AUTOMATIC RECOGNITION OF PRINTED AND HANDWRITTEN MATHEMATICAL EXPRESSIONS

BY

UTPAL GARAIN

Computer Vision & Pattern Recognition Unit

INDIAN STATISTICAL INSTITUTE

203, B. T. ROAD

KOLKATA 700 108

INDIA

A THESIS SUBMITTED TO

THE INDIAN STATISTICAL INSTITUTE

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Dedicated to my parents,

Shri Sahadeb Garain and Smt. Tulasi Garain.

ACKNOWLEDGEMENTS

At the very beginning, I would like to thank my supervisor Prof. B. B. Chaudhuri for introducing me to the topic of Optical Character Recognition, his advise, guidance, and his insistence on being meticulous in different aspects.

I express gratitude towards my colleagues Prof. S. K. Parui, Dr. U. Pal, Dr. M. Mitra, and U. Bhattacharya with whom I have had frequent discussion on several aspects of the work embodied in this thesis. Dr. J. Mitra, Dr. A. Ray Chaudhuri, H. V. K. Swamy, T. Pal, and R. P. Ghosh helped me a lot for machine implementation of several modules proposed in the thesis. Also, they are my co-authors for some of the papers published/communicated based on the work described in this thesis. I thank them all.

I reserve special thanks for Prof. D. Blostein of the School of Computing, Queen's University, Kingston, Canada, for her kind suggestions on the development of a corpus of mathematical expressions. She also took the pain of presenting one of our papers at the 4th IAPR International Workshop on Graphics Recognition (GREC), Canada, 2001. I gratefully acknowledge the help I received from Dr. M. Okamoto of Department of Information Engineering, Shinshu University, Japan and Dr. D.-Y. Yeung of Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong. They, upon my request through email, not only sent me copies of their publications but also pointed out many useful references for my work.

My friends and colleagues at the Computer Vision and Pattern Recognition (CVPR) Unit of Indian Statistical Institute, Kolkata helped me in many ways in the course of my work. Many of my colleagues and students whose cooperation has been acknowledged later in respective chapters helped me a lot by providing useful references, generating and checking the corpus (of expressions) content and scrutinizing the thesis for minute typographical mistake with utmost care. Also, I thank all office staffs of the CVPR unit, who provided me all possible logistics help to complete my work.

I express gratitude towards members of my family - my parents, my brothers- Ashis and Biplab, and others, for their help and encouragement. I reserve special acknowledgement for my wife, Sabita, for her great support, cooperation and inspiration during the entire phase of the research. The same goes for my little son, Udbhas.

May 9, 2005

UTPAL GARAIN
CVPR UNIT, ISI
KOLKATA

ABSTRACT

This thesis presents a systematic study on recognition of printed and handwritten mathematical expressions. Automatic recognition of printed expressions is an essential requirement for efficient Optical Character Recognition (OCR) of scientific paper documents. On the other hand, recognition of handwritten expressions has been tried for online environment. Here expressions are written using electronic data tablet/stylus providing a convenient alternative to keyboard or mouse used for data entry into a computer.

The previous studies dealing with different aspects of expression recognition are, at first, reviewed. Next, the scope of the present thesis, its layout and contributions are outlined. Discussion on OCR of printed expressions starts with constructing a representative corpus of scientific documents taken from various branches of science. Methods for groundtruthing expressions contained in the documents, statistical analysis of the corpus, etc. are presented to facilitate research on expression recognition.

Next, issues related to recognition of expressions are elaborately discussed in a chapter wise manner. In case of printed documents, identification of expression zones is considered for smooth upgradation of the existing OCR systems to properly handle documents containing expressions. Such an identification task keeps the main OCR engine undisturbed while a specially designed module can work for recognition of expressions. Online recognition of handwritten expressions assumes expressions are entered in isolation and therefore, no component for identification of expression zones is needed under online environment.

Recognition of expressions under any environment (printed or handwritten) involves two major stages: (i) symbol recognition and (ii) interpretation of expression structure. Techniques to realize these stages are presented for both the printed and handwritten expressions. All processing modules are methodically tested on a large dataset to attest the feasibility of the proposed approaches.

Errors encountered in different modules are analyzed in detail and a set of error-correcting rules is formulated. The design of rules exploits several contextual information to improve the overall expression recognition accuracy for both the printed and handwritten expressions. A method for evaluating performance of an expression recognition system has been presented. The proposed performance measure considers several non-trivial issues related to an expression recognition task and provides a single figure of merit to judge the efficiency of a system. The thesis has been concluded with a summary of its achievements and a discussion on future extension of the present study.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

Automatic recognition of mathematical expressions (hereafter, referred as expressions) is one of the challenging pattern recognition problems of significant practical importance. Such a recognition task is required while converting scientific documents from printed to electronic form or to aid reading of scientific documents for the visually impaired persons. On the other hand, recognition of online handwritten expressions facilitates the users to enter expressions through a data tablet and thereby provides a convenient alternative to the input methods like typing $\TeX$ syntax for expressions or using an equation editor like the one available with Microsoft Word.

This thesis is devoted to the development of a system for recognition of both printed and online handwritten expressions. Recognition of expressions involves two major components namely (i) symbol recognition and (ii) structure interpretation. Symbol recognition is difficult because a large character set (Roman letters, Arabic digits, Greek letters, Operator symbols, etc.) with a variety of typefaces (regular, italic, bold), and a large number of different font sizes may be used to generate the expressions. Moreover, certain symbols (e.g. *integration*, *summation*, *product*, *brackets*, etc.) are elastic in nature and have a wide range of possible scales.

Interpretation of structure is particularly difficult for expressions due to the subtle use of space that often defines the relationship among symbols. For instance, unlike plain text (which is written linearly from left to right), symbols in an expression can be written above, below, and one inside another. Therefore, understanding of the spatial relationship among symbols is crucial to the interpretation of structure of an expression. This means that even if all the characters are correctly recognized, there still remains the non-trivial problem of interpreting the two-dimensional structure of an expression. Moreover, several symbols (e.g. *horizontal line*, *dot*, etc.) have multiple meanings depending on the context and such ambiguous role of symbols makes the interpretation task more difficult.

As far as printed scientific documents are concerned, an additional processing is needed for the identification and extraction of expression zones. The expressions may appear in two modes namely, (i) embedded (also called in-line expressions) i.e. mixed with normal text and (ii) displayed (also called isolated expressions) i.e. typed in distinct line. Since the presence of expressions disturbs an existing Optical Character Recognition (OCR) system (not trained for expression recognition), the identification and extraction of expression zones, therefore, may help in efficient conversion of scientific paper documents into electronic form. Such a framework permits an existing OCR engine to process the normal text portion as usual whereas the extracted expressions can be processed by a system specially designed for expression recognition.

On the other hand, handwritten expressions are more complex since each writer has his/her own writing style and the system should be able to recognize different shapes for the same symbol. Moreover, many writers tend to connect and abbreviate the strokes for various symbols in different ways and significant variation in stroke number and order is observed in handwriting. Moreover, the ambiguity in spatial relationships among symbols is substantially increased for handwritten expressions, adding further complication to the interpretation of expression structures.

Work on an expression recognition system needs another problem to be addressed. The quantitative evaluation of expression recognition results is a non-trivial problem since recognition scheme involves two major stages: symbol recognition and structural analysis. The stages are tightly coupled and therefore, if evaluation in one stage is done independent of the other, then it may not reflect true performance of the system. Moreover, error in the symbol recognition stage affects the structure analysis result. This calls for an integrated evaluation mechanism for judging the performance of a system dealing with expression recognition.

1.1 Review of Related Works

Studies on recognition of mathematical expressions date back to late sixties of the last century when Anderson [2] proposed a syntax-directed scheme for recognition of hand-printed expressions. Several studies have been reported and surveyed in [8, 13, 31]. The review presented here is categorized according to different environments (printed and handwritten) under which the expressions are recognized.

1.1.1 Recognition of Printed Expressions

In case of printed scientific documents, expressions are typically appear in two modes, either as distinct (or displayed) expressions or embedded into text lines. Thus, identifying the expression zones in the input document is considered as the first step in printed expression recognition. Next, expression symbols are recognized and finally, arrangement of symbols is analyzed to interpret the expression's structure.

Identification of Expression Zones

Studies dealing identification of expression zones are few in number as the most of the previous works assume that expressions are available in isolated form. Among the existing techniques, method proposed by Lee and Wang [69, 70] labels text lines in a document as either *TEXT* (to denote normal text) or *EXP* (to denote displayed expression) based on two properties (i) isolated expressions are taller and (ii) the line spaces above and below them are larger than those between text lines that contain no mathematical expressions. The technique for locating embedded expressions initially recognizes characters in a text line from left to right direction and then converts them to a stream of tokens. A token is decided to belong to an embedded expression according to some basic expression forms which considers presence of special mathematical symbols (e.g. horizontal line, summation, product, etc.), super-scripting, or matrix structures. Symbols that are adjacent to the above tokens are heuristically attached to form an embedded expression.

Fateman [33] presented a three-pass algorithm that initially recognizes all connected components in a scanned document and separates them into two bags, math and text. The text bag contains all Roman letters, italic numbers and the math bag includes punctuations, special symbols, italic letters, Roman digits, and other marks (e.g. horizontal lines, dots), etc. Next, components in the math bag are grouped into zones according to their proximity. Symbols that are left ungrouped and appeared to be too far from other math symbols are moved to the text bag. Symbols in the text bag are similarly joined up into groups according to proximity. Text words (hopefully include words like "sin", etc.) that are relatively isolated from other text but within any previously identified math zone are moved to the math bag. Segmentation result is finally reviewed by human assistance to correct errors, if any.

The method proposed by Inoue *et. al.* [52] isolates expressions contained in Japanese scientific document by assuming that the OCR recognizes Japanese characters with high confidence whereas expression symbols are either rejected or recognized (rather misrecognized) with low confidence. In another approach, Toumit *et. al.* [98, 99, 100] locate embedded expressions by finding special symbols like "=", "+", "<", ">", etc. and some specific context propagation from these symbols is done. For example, for parenthesis and brackets, symbols between them are checked; for horizontal bars, symbols above and below them are investigated; etc.

Later on, Kacem *et. al.* [57, 58] proposed a two-pass scheme which does not put much emphasis on symbol recognition. Initially, expressions are separated from the text lines using a primary labeling which uses fuzzy logic based model to identify some mathematical symbols. Later on, a secondary labeling uses some heuristics to reinforce the results of the primary labeling and locates super/sup-scripts inside the text. An evaluation strategy has been presented to judge the expression extraction technique and a success rate of about 93% has been reported on a combined test set of 300 displayed and embedded expressions. A similar technique is used in [95] to locate mathematical expressions in printed documents.

Recently, Chowdhury *et. al.* [24] proposed a recognition-free approach that exploits the usual spatial distribution of the black pixels in math zones. Experimental results show that the method works well for segmenting displayed expression (with a success rate of 97.69%) but gives only 68.08% accuracy for extraction of embedded expression. In another recognition-free technique reported by Jin *et. al.* [55], embedded expressions are extracted based on the detection of two-dimensional structures. However, the authors of [24, 55] concluded that the extraction of embedded expressions is quite difficult without doing character recognition.

Recognition of Expression Symbols

As far as printed expressions are concerned, majority of previous studies have put emphasis on interpretation of expression structures. In several experiments, an error-free symbol recognition is assumed before formulating methods for symbol arrangement analysis. In controlled research environment, it is possible to bypass the symbol-recognition step and concentrate on structure analysis phase. However, design of a symbol-recognition module is essential to realize a complete expression recognition system.

The approaches proposed in previous studies on recognition of printed mathematical symbols can be broadly classified into two categories namely, (i) template matching and (ii) feature extraction and classification. Okamoto *et. al.* [81, 82] followed template matching approach where two sets of dictionaries for normal and script type symbols are maintained. Symbols are normalized to the predefined size prior to classification. Based on this proposed method, an accuracy of 98.96% has been reported in [84]. Also, the authors have addressed the problem of touching characters in expressions and presented a segmentation technique [83] which is based on projection profiles of a given binary image and minimal points of a blurred image obtained by applying a Gaussian kernel to the original image.

Fateman *et. al.* [31, 32] proposed another template matching technique where a symbol template is represented by a vector of features. Bounding box of gray-level character image is divided into 5 by 5-rectangular grids and percentage of gray values in each grid is computed. The feature vector is made up of this set of gray-values along with two more data items, the height-to-width ratio and the absolute height in pixels of the bounding box. During classification of symbols, the authors used a Euclidean metric to define the distance between characters.

Lee and Lee [67, 68] proposed a feature extraction based classification scheme where 13 features are utilized to represent each symbol. Next, a coarse classification algorithm is applied to reduce the number of candidates. For each input symbol, the character with the highest similarity is selected as the candidate symbol. The recognition accuracy reported in [68] is 84.80%. The method presented by Lee and Wang [69, 70] initially divides the symbol set into three classes based on the aspect ratio of bounding box of symbols. For recognizing a symbol within a class, the symbol image is divided into 4x4 non-uniform blocks and a 4-dimensional direction feature vector is computed from each image block. It gives a 64-dimensional feature vector representation for each symbol. The authors achieve an accuracy of 96.18%. Ha *et. al.* [47] also adopted a feature extraction based approach, but their classification is done through neural network.

Suzuki *et. al.* [95] designed a recognition engine that tries to distinguish 564 symbol categories. The number of classes is so large because the authors considered several categories for a single character to tackle font and style variation. For instance, 6 categories have been considered for the character 'B' to take care of its *regular*, *italic*, *bold*, *calligraphic*, etc. versions. A three-step coarse-to-fine classification strategy has been employed for recognition of symbols. The features like aspect ratio, crossing counts, directional, peripheral and mesh features have been used for classification of symbols. Experiments conducted on a set of 476 scientific pages showed an accuracy of 95.18% for recognition of expression symbols.

Interpretation of Expression Structure

Studies dealing with interpretation of expression structure are quite a few in numbers. One of the earliest contributions in this area is by Anderson [2, 3]. His syntax-directed technique is essentially a top-down parsing approach based on a co-ordinate grammar. Though the partitioning strategy used there do not show satisfactory results in many cases, this work is regarded as pioneering one. Afterwards, Chang [17] proposes an algorithm based on operator precedence and operator dominance, but did not clearly explain the algorithm as well as its performance in practical scenario.

Later on, several studies have been proposed to analyze arrangement of symbols in expressions. Among them, some consider expressions in printed form whereas others assume handwritten input. In a few cases (e.g. [113]), structure analysis in both the printed as well as handwritten data has been attempted by a single approach. The techniques proposed for symbol-arrangement analysis in printed expressions are outlined below and the approaches presented for online environment are reviewed in the next section.

In 1989, Chou [23] presented a stochastic context-free grammar to understand two-dimensional (2-D) structure of expressions. A 2-D probabilistic version of Cocke-Younger-Kasami parsing algorithm [1] is proposed to find the most likely parse of the observed image. The author demonstrated that such a stochastic framework can recognize images of noisy equations and learn the noise probabilities. However, very little is discussed about the construction of the training set and the experimental results. Among other related approaches, Hull [50] computed probabilities by which two sub-expressions are in a particular relationship. The algorithm attempts to enumerate all possibilities by using an A-star search and prunes away the unlikely ones. Later on, Miller and Viola [76] tried to limit the number of potentially valid interpretations by decomposing the expressions into a sequence of compatible convex regions. A lower bound estimate on the cost to reach the goal is also provided. However, the authors felt the need for further improvement of the system.

Twaakyondo and Okamoto [103] presented a technique that uses notational conventions in typing expressions. Structure of an expression is analyzed by *projection-profile cutting* and a top-down strategy is used to analyze the horizontal and vertical relation between sub-expressions. To analyze nested structure such as subscripts, superscripts, etc. a bottom-up strategy is invoked that begin with the smaller sized symbols. The authors also provided an automatic approach [84] for evaluating their method. An accuracy of 98.04% is reported for recognition of 4,701 elementary expression structures like *scripts*, *limit*, *fraction*, etc.

The method proposed by Lee and Lee [67, 68] uses a procedure-oriented bottom-up approach to translate a 2-D expression into a 1-D character string. Initially, smaller symbol groups are formed around seven special mathematical symbols (i.e. Σ, Π, *fraction*, etc.) which may deviate from the typographical center of an expression. Next, symbol groups are ordered from left to right based on their center y-coordinates. Matrices are handled separately [70]. The authors consider 105 expressions for training and 50 expressions for testing. An error rate of about 2% is reported but the approach for computing the error rate is not presented.

Fateman *et. al.* [6, 32] described a *recursive decent* parser where an additional stage (called *linearization*) is used in between lexical analysis and the conventional parsing. In the linearization phase, adjacency relations among the tokens are detected. Several data-dependent heuristics are used. The experiments put emphasis on parsing of integrals. In another study [47], J. Ha *et. al.* outlined a system that uses a recursive X-Y decomposition to understand the geometric layout of an expression. A top-down approach is followed to construct an expression tree, which is checked next for syntax errors.

Toumit *et. al.* [99, 100] assigned different priority levels to symbols in order to present a tree representation of the input expression. An alternative approach based on graph representation of an expression image is proposed by Grbavec and Blostein [45]. Nodes in the graph represent symbols or sub-expressions and edges represent relationship between the sub-expressions. A graph-rewriting rule replaces one sub-graph by another. Experiment on 13 expressions is reported, but details of test results are not presented. Later on, Lavirotte and Pottier [65, 66] used context-sensitive graph grammar technique which attempts to add context in the graph-rewriting rules so that some ambiguities are removed as automatically as possible.

Eto and Suzuki [30] proposed a concept of virtual link network to recognize expression structures. In their approach, a network with vertices representing the symbols is constructed first. Vertices link each other by several labeled edges and the cost representing possible relations of the pair of symbols. Next, a minimum cost spanning tree of the network is generated to encode the entire expression. The proposed technique was tested using 123 expressions, of which 110 are properly recognized. The same was incorporated in [95] and experiment conducted on a larger dataset containing 12,493 expressions reported a structure recognition accuracy of 89.6%.

Garcia and Couasnon [44] proposed a generic method, DMOS (Description and MOdification of Segmentation) for recognition of musical scores, tables, forms, etc. They applied this technique to recognize the structure of mathematical formulae and some symbols made of line segments. Using DMOS, the authors found some possibilities to improve symbol recognition accuracy as well as to overcome segmentation problems occurring in old mathematical formulae. The proposed method was tested using 60 formulae but details of the test results were not available.

More recently, Zanibbi *et. al.* [113] described an algorithm consisting of multiple passes for (i) constructing a baseline structure tree describing the 2-D arrangement of symbols, (ii) grouping tokens comprised of multiple symbols and (iii) arranging symbols in an operator tree which describes the order and scope of operations in the input expressions. Experiment using 73 expressions of UW-III database [87] showed a recognition (at expression level) accuracy of 38% (at most). However, the authors presented an in-depth analysis of the performance of their proposed method for structure analysis.

1.1.2 Recognition of Online Handwritten Expressions

In case of online expression recognition as input comes from a person writing on data tablet, where text and expressions are generally not mixed. Therefore, identification of expression zones does not arise in such a case. Consequently, recognition of online expression concentrates only on symbol recognition and analysis of symbol-arrangement.

The earliest paper in this area is due to Anderson [2] who assumed an error-free symbol recognition and presented a co-ordinate grammar for analyzing the 2-D structures of hand printed expressions. A partitioning strategy was used for rules with two non-terminal syntactic units on their right side and each partition might require considerable processing. Later on, Anderson [3] proposed techniques to improve efficiency of the system, but did not provide recognition rate or performance evaluation results.

In the system proposed by Belaid and Haton [4], handwritten symbols are segmented into basic primitives to be fed into recognition engine. The syntactic approach proposed by Anderson is modified and two parsers (top-down and bottom-up) are used to interpret the expression structures. Eight expressions written by ten persons four times to create a dataset of 320 expressions are used to achieve a symbol recognition accuracy of 93% with 5% rejection. The system correctly recognized all structures except six cases with two confusions and four rejections.

Studies presented by Koschinski *et. al.* [59] and Winkler *et. al.* [107, 108, 109] used Hidden Markov Model (HMM) for recognition of handwritten symbols and incorporated a soft-decision approach for structural analysis of expressions. Alternate solutions are generated during the analysis process if the relation between two symbols within the expression is ambiguous. Finally, a string representing the input expression is generated and syntactically verified for each alternative. Strings failing this verification process are considered invalid. The authors considered 82 symbols written 50 times by a subject and used 40 versions for training the HMMs. Maximum writer-dependent recognition accuracy of 96.9% has been reported.

Among other HMM based approaches, Kosmala *et. al.* [60, 61, 62] employed several online and offline features which are fed to discrete NN-HMMs (NN: Neural Net). HMMs of different number of states have been proposed to recognize the symbols. For structure analysis, the authors used a graph grammar approach [66] that generates a tree-structure for the input expression. However, the detailed experimental results were not presented.

Xuejun et. el. [110] proposed a recognition method that analyzes the structures of 94 commonly used expression symbols. Symbol matching is done by an improved Kohn-Munkres algorithm. The approach has been tested with 94 symbol classes written 5 times by 20 different persons. A writer-dependent recognition rate of 90.52% has been reported. However, the analysis of expression structures is not presented there.

Sakamoto *et. al.* [93] used a 16-directional coding scheme to capture writing directions of a stroke. A stroke is further labeled as up-stroke and down-stroke depending on the writing direction. A dynamic programming is used for segmentation of a sequence of strokes into character units. In a related study [38], Fukuda *et. al.* have used 3×5 mesh directional element features and some additional features for recognition of symbols. For analyzing expression structure, the authors in [93, 38] have employed the same technique [52] that chooses one of the nine pre-defined relations for a pair of expression symbols. Four expressions are used in the experiment. Twenty persons have written each expression twice, thus generating a set 160 expressions. A character recognition accuracy of 99.35% is reported. Here, the structure analysis module has been evaluated by finding correct identification of relations between a pair of symbols. The technique shows an efficiency of 98.46% while identifying such relationships.

Chan and Yeung [14] presented a syntactic approach to understand expression structure. A Definite Clause Grammar (DCG) is used as a formalism to define a set of replacement rules for parsing the expressions. The authors improved the efficiency of the parsing process by reducing the number of backtrackings used by a DFG. The symbols are recognized by following a flexible structure matching approach [12]. Experiments are done on 60 commonly used expressions taken from four domains, namely, elementary algebra, trigonometric functions, geometry, and indefinite integrals. Performance evaluation of the system is presented in a different paper [15] where effectiveness of both the symbol recognition and structural analysis stages is demonstrated by a single measure. Experimental results show that recognition speed ranges from 0.73 to 6 seconds per expression on a modest workstation.

Toyozumi *et. al.* [101] proposed a system that recognizes each stroke by Freeman chain code. Several strokes are combined into a character based on their positions and combinations. Structural analysis is done by dividing an expression into blocks, but details of the method and dataset used are not presented. Reported recognition rates are 80%, 92%, and 69% for symbols, mathematical structures and matrix structures, respectively.

Later on, Zanibbi *et. al.* [112, 113] described a tree transformation based method to understand expression structures. The approach makes use of search functions that exploit the left-to-right reading order of expression notations and operator dominance to recursively and efficiently extract baselines in an expression. Recognition of symbols is achieved through another system [111]. Five fairly complicated expressions written by 27 participants are used in the experiment. No details of the experimental results for processing online expressions are available but it is reported that the participants found the output to be useful.

More recently, Tapia and Rojas [96] outlined a system that involves support vector machines for recognition of handwritten symbols and the reconstruction of formulas is based on baseline structure analysis [113]. The proposed system supports use of 43 distinct symbols for writing expressions and for recognition of symbols an accuracy of more than 99% has been reported .

1.2 Motivation for the Present Work

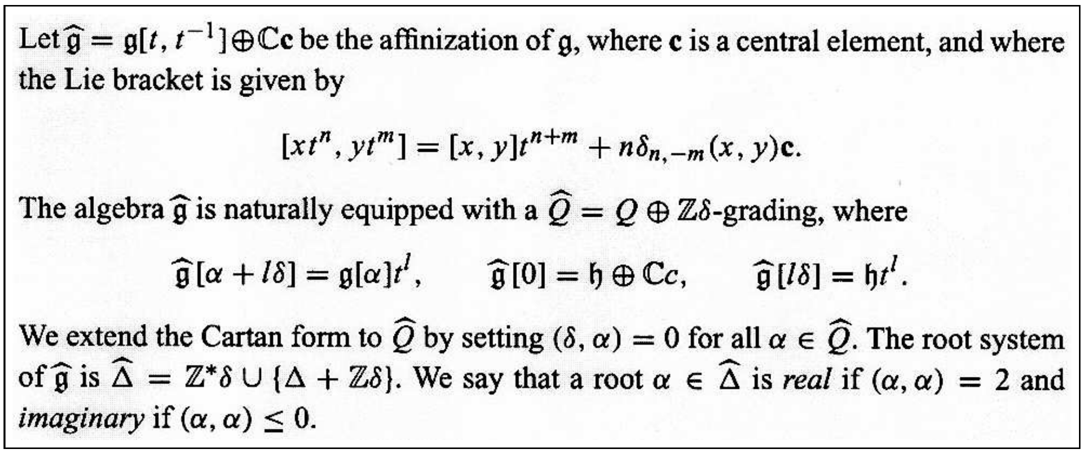
The existing OCR systems show severe limitations for converting scientific papers into corresponding electronic form. Figure 1.1 demonstrates one such example obtained from one the popular OCR systems. This is so because current systems fail to recognize mathematical expressions that often appear in scientific documents. On the other hand, review of the previous studies dealing with recognition of expressions reveals that most of the studies concentrate on different sub-problems instead of providing a complete solution. The work embodied in this thesis is motivated to fill this gap.

The proposed study focusing on recognition of online handwritten expressions is aimed at providing a better man-machine interface for entering mathematics while preparing scientific documents. With the advent of pen based devices (e.g. PDAs, tablet PCs, etc.) research on online handwriting recognition has attained a considerable attention in recent past. Though there exists several works on recognition of handwritten text, the studies dealing with handwritten expressions are a few in number. Therefore, systems extending a support for online entry of mathematical expressions are still quite immature for the commercial market. Our present effort is directed to this end so that the users preparing scientific documents are facilitated with a convenient alternative to input methods such as typing expressions following TEX syntax or using an equation editor like the one available with Microsoft Word.

1.3 Contributions of the Thesis

As far the state of the art is concerned, this thesis has several contributions for development of a system for recognition of expressions. Some of the major contributions are briefly discussed below:

- At first, the thesis deals with the development of a corpus of expressions. This



Let $\widehat{\mathfrak{g}} = \mathfrak{g}[t, t^{-1}] \oplus \mathbb{C}\mathbf{c}$ be the affinization of $\mathfrak{g}$, where $\mathbf{c}$ is a central element, and where the Lie bracket is given by

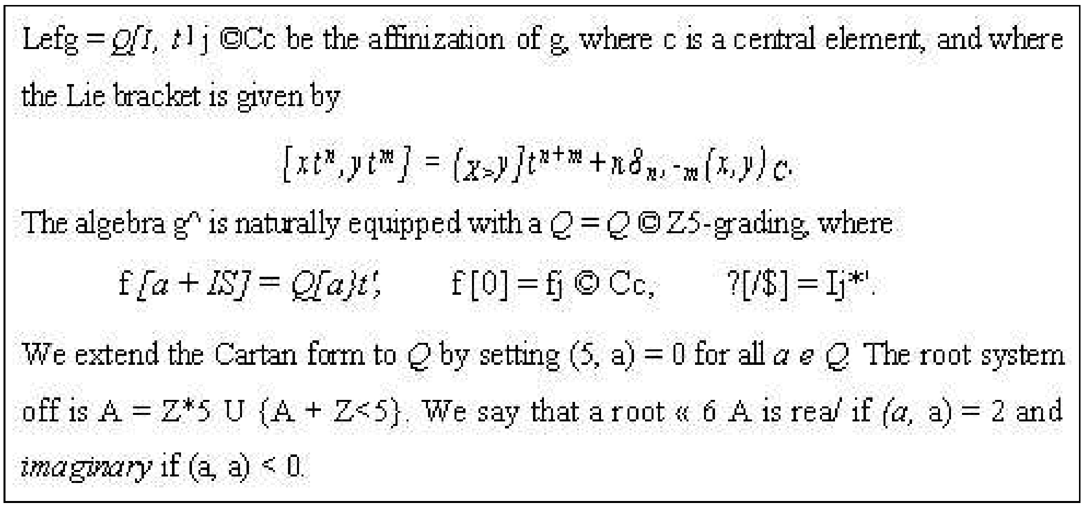
$$[xt^n, yt^m] = [x, y]t^{n+m} + n\delta_{n,-m}(x, y)\mathbf{c}.$$

The algebra $\widehat{\mathfrak{g}}$ is naturally equipped with a $\widehat{Q} = Q \oplus \mathbb{Z}\delta$-grading, where

$$\widehat{\mathfrak{g}}[\alpha + l\delta] = \mathfrak{g}[\alpha]t^l, \qquad \widehat{\mathfrak{g}}[0] = \mathfrak{h} \oplus \mathbb{C}c, \qquad \widehat{\mathfrak{g}}[l\delta] = \mathfrak{h}t^l.$$

We extend the Cartan form to $\widehat{Q}$ by setting $(\delta, \alpha) = 0$ for all $\alpha \in \widehat{Q}$. The root system of $\widehat{\mathfrak{g}}$ is $\widehat{\Delta} = \mathbb{Z}^*\delta \cup \{\Delta + \mathbb{Z}\delta\}$. We say that a root $\alpha \in \widehat{\Delta}$ is *real* if $(\alpha, \alpha) = 2$ and *imaginary* if $(\alpha, \alpha) \leq 0$.

(a)



Lefg = *Q/I, t* 1 j ©Cc be the affinization of g, where c is a central element, and where the Lie bracket is given by

[xtn, ytm] = (x>y]tn+m + nδn, -m(x,y) c.

The algebra g^ is naturally equipped with a Q = Q © Z5-grading, where

f *[a + IS]* = *Q[a}t',* f [0] = fj © Cc, ?[/$] = Ij*'.

We extend the Cartan form to Q by setting (5, a) = 0 for all *a e* Q The root system off is A = Z*5 U {A + Z<5}. We say that a root « 6 A is *rea/* if *(a,* a) = 2 and *imaginary* if (a, a) < 0.

(b)

Figure 1.1: OCR of scientific documents: an example (a) image (b) recognition results.

study is motivated to facilitate research on automatic recognition of expressions. Moreover, unavailability of suitable corpora of expressions has so far prompted the researchers to define their own dataset for testing their algorithms. As a result, replication of experiments and comparison of performance among different methods have become difficult tasks. The proposed corpus that is available on request will substantially contribute to this end.

- Unlike most of the previous studies on recognition of printed expressions, the approach presented in this thesis does not assume that the expressions are available in isolated form. In reality, expressions typically appear in documents, either as isolated (displayed) expressions or embedded directly into text lines. Therefore, the proposed technique for identification and extraction of expression zones in scientific documents helps to successfully upgrade the existing OCR systems (not trained for expression recognition) for converting scientific paper documents into their electronic form.

- A general framework based on multifactorial analysis has been presented to solve problems where the solution depends on several factors. The approach has been applied for two different cases namely, extraction of expression zones and segmentation of touching characters. The results obtained by using this approach strongly attest its potential and it is likely that this framework will find applications in other document analysis problems.

- A multiple classifier system proposed for recognition of expression symbols provides promising performance. Unlike previous studies, the present classifier deals with a large number symbol classes (274 and 198 types of symbols for printed and handwritten expressions, respectively) appearing in variety of mathematical expressions. Moreover, the classifiers are not only restricted to the recognition of expression symbols only. Their capabilities have also been checked for other character recognition problems like ones in [39, 40].

- The structure recognition scheme describes an easily computable technique for parsing the expressions. The context-free grammar presented in this thesis is schematically simple but still able to successfully process a large variety of expressions found in different branches of science. Moreover, the same parsing technique shows its capability to interpret structures of printed as well as handwritten expressions with suitable modification.

- Recognition of online handwritten expressions supports recognition of a symbol set of 198 different characters that a user may employ while writing the expressions. Moreover, the proposed technique allows thirteen two-dimensional structures including *matrices*, *scripts*, *root*, *limit expressions*, etc. without imposing any restriction on the nesting of one structure into another. These facilitate entry of various types of expressions appearing in different branches of science.

- A new performance measure has been presented to assess the system performance. In case of printed scientific documents, a method to judge the efficiency for extraction of expression zones has been formulated. On the other hand, a performance-index is computed to evaluate an expression recognition system. The proposed index integrates the results of symbol recognition and structure interpretation, maintaining a proper balance on both aspects and provides a single figure of merit to evaluate the overall recognition performance.

1.4 Organization of the Thesis

The content of the thesis can be broadly divided into three major parts: (i) *Recognition of Printed Expressions*: Four chapters (from Chapter 2 to Chapter 5) deal with recognition of printed expressions; (ii) *Recognition of Handwritten Expressions*: Chapter 6 is devoted for this purpose. However, some techniques presented in Chapter 4 (mainly the techniques for combination of classifiers) and Chapter 5 (parsing technique) are reused with suitable modification. (iii) *Post-processing and Performance Evaluation*: Chapter 7 presents discussions related to post-processing and performance evaluation.

A chapter-wise break up of the thesis is briefly given below:

Chapter 2. This chapter is concerned about the construction of a representative corpus of technical and scientific documents. The proposed database contains 400 images of documents containing embedded as well as displayed expressions. Both real and synthetic (generated by TEX or Microsoft Word) documents are present in the dataset. A format has been proposed to groundtruth embedded and displayed expressions appearing in documents.

A statistical analysis of the corpus content is presented next. The occurrence frequencies of expression symbols are computed. Other statistical investigations are carried out and the usefulness of analysis results demonstrated in the related research problems namely, (i) identification and segmentation of expression zones from the rest of the document, (ii) recognition of expression symbols, (iii) interpretation of expression structures, and (iv) performance evaluation of an expression recognition system.

Chapter 3. This chapter deals with identification and extraction of expression zones contained in scientific paper documents. As the displayed and embedded expressions impose different level of complexities, separate techniques have been proposed for their extraction.

Identification of displayed expressions is done using some image-level features. However, identification of an expression zone is confirmed by checking the presence of one (or more) of the symbols that appear in the expressions with quite high frequencies. On the other hand, the method for locating embedded expressions initially use an existing OCR system to recognize the input document and then the expression zones are pinpointed by exploiting the inability of ordinary OCR system to handle expressions and by using some common typographical features noted in mathematical expressions.

Chapter 4. In this chapter, a multiple classifier system has been presented for recognition of printed expression symbols. A group of four classifiers having different capabilities are arranged hierarchically in two levels. The classifier used at the top level employs stroke-based technique to recognize the symbols that are simple in shape but appear with high occurrence frequencies. Symbols not recognized at the first level are passed to the second level that employs a combination of three classifiers where feature descriptors like run-number or crossing count, density of black pixels and wavelet decomposition are employed. Several combination techniques have been attempted to integrate the second level classifiers to achieve high recognition accuracy.

The presence of connected (or touching) symbols sometimes disturbs recognition of symbols. Therefore, an approach for segmentation of connected symbols is outlined in this chapter. Several image level features are considered and a multifactorial analysis [72] is implemented to find appropriate cut positions. A quantitative comparison among the exiting recognition approaches is also presented to show the distinctiveness of the recognition technique presented in this thesis.

Chapter 5. This chapter deals with interpretation of the geometric structure of expressions. A simple grammar-based approach has been presented to recognize complex arrangement of expression symbols. The proposed technique is based on symbol identities and their positional information. Initially, symbols in an expression are arranged in a number of hierarchical levels based on their size and positional information. The results obtained from the statistical analysis presented in Chapter 2 are used to define the symbol levels and several geometric properties of printed expressions.

Initially, the entire expression image is partitioned into some vertical and horizontal stripes based on pixel projection. This partition is done recursively until an atomic stripe is obtained. Each stripe represents a token or lexical group. Next, a bottom-up approach is followed where two or more tokens are combined together to form a sub-expression. Finally, the sub-expressions are merged to form the final expression string. Space and time complexity analysis for the proposed approach is also presented.

Chapter 6. This chapter aims at recognition of online handwritten mathematical expressions. The techniques for pre-processing of the raw data recorded by an electronic tablet are presented first. Next, recognition of handwritten symbols is discussed. The recognition technique is based on human motor model. The model tries to grab the essence of the idea used to teach children to write characters. A multiple classifier approach consisting of both parametric and non-parametric classifiers has been adopted to implement the proposed model. For arrangement of symbols the technique proposed in Chapter 5 has slightly been modified to work under online environment. The modified version considers online features to identify certain spatial relationship among symbols.

Construction of a database of handwritten expressions has been presented to test the proposed system. One hundred and seventy five (175) expressions are taken from different branches of science including school and college level books. Forty writers belonging to different categories are involved in writing each of the selected expressions twice. Altogether the database contains 5500 handwritten samples for 175 expressions. Samples are groundtruthed following a specific format. Method for a semi-automatic evaluation of recognition performance is also outlined in this chapter.

Chapter 7. This chapter outlines some general aspects that are common to both printed and online environment. An error detection and correction module has been designed to improve the overall expression recognition results. The types of error that occur during symbol recognition and structure interpretation are studied in details and techniques incorporating contextual information are presented to correct these errors.

Next, strategies for performance evaluation of an expression recognition system are discussed. At first, the proposed method separately calculates accuracy for recognition of symbols and structures. Geometric (or structural) complexity of an expression has been defined and considered for evaluation of structure recognition accuracy. Next, a performance-index is computed to integrate the results of symbol recognition and structure interpretation, maintaining a proper balance on both aspects. The proposed index provides a single figure of merit to evaluate the overall recognition performance.

Chapter 8 concludes the thesis and outlines the scope of future work.

CHAPTER 2

A CORPUS FOR OCR RESEARCH ON MATHEMATICAL EXPRESSIONS

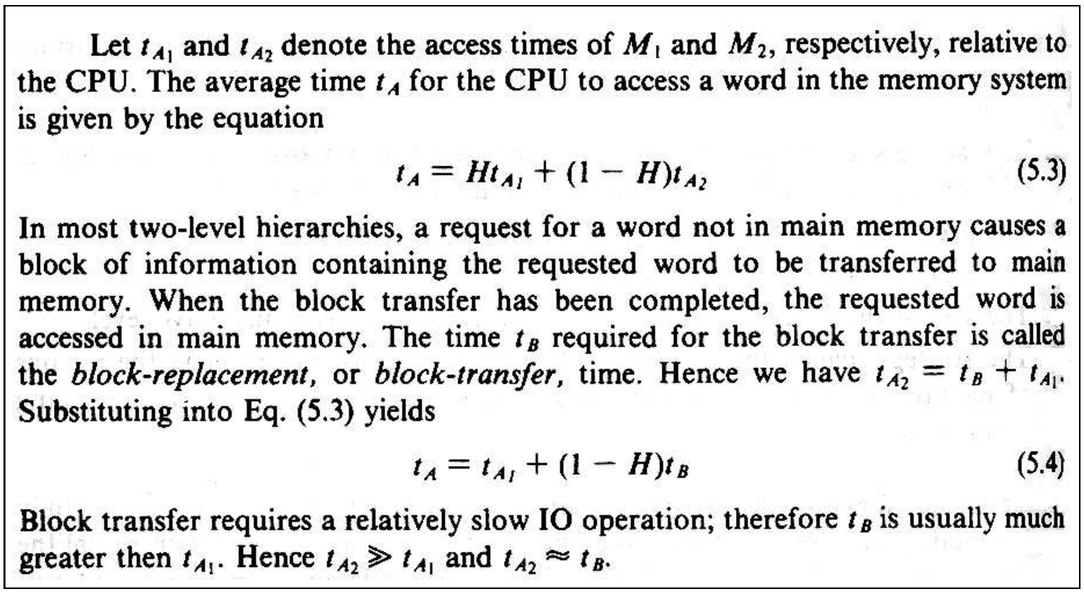
2.1 Introduction

Automatic transcription of printed scientific and technical documents into their electronic format largely depends on the success in recognizing the typeset mathematics. Several studies dealing with recognition of printed mathematics have been reported in the literature. These research efforts have been surveyed in [8, 13, 31]. From these reports it is understood that unavailability of a suitable corpus of expressions has prompted the researchers to define their own data set for testing their algorithms. As a result, replication of experiments and comparison of performance among different methods has become a difficult task.

In this chapter, we present a corpus (or database) of mathematical expression images that would facilitate research in automatic understanding of expressions. The only relevant database available so far is the University of Washington English/Technical Document Database III (UW-III) [87]. However, the database is mainly constructed for general OCR (Optical Character Recognition) research and contains 25 groundtruthed (into TEX) document pages containing about 100 expressions. Therefore, it does not seem to be a representative corpus for the respective research. Another drawback of this data set is that groundtruthing of expressions into TEX only does not support an in-depth analysis of recognition performance [15, 84, 113]. Another freely available source of expressions is the set used by Raman for his Ph.D. work [91]. However, the expressions available here are synthetic (generated by TEX) and isolated (i.e. not a part of any document) in nature.

This chapter discusses about the contents of a corpus of printed scientific documents collected at the Computer Vision & Pattern Recognition Unit of the Indian Statistical Institute, Kolkata, India and describes how this database addresses various research considerations related to recognition of printed mathematical expressions. This work is an extension of our earlier effort presented in [43].

In printed documents, expressions appear in two modes, namely, *embedded* (mixed with text and also referred to as in-line expression) and *displayed* (typed in a separate line). Figure 2.1 shows a typical example of such a document. The content of the database, therefore, is arranged into a two-level hierarchy. At the top level, there are 400 scientific and technical document images containing mathematical expressions. For each document, its *embedded* and *displayed* expressions are collected into two different files. Correspondence between a top-level document with its lower level files storing embedded and displayed expressions is maintained through the naming convention for files. Expressions are groundtruthed following a specified format explained later.



Let t_{A_1} and t_{A_2} denote the access times of M_1 and M_2, respectively, relative to the CPU. The average time t_A for the CPU to access a word in the memory system is given by the equation

$$t_A = Ht_{A_1} + (1 - H)t_{A_2} \tag{5.3}$$

In most two-level hierarchies, a request for a word not in main memory causes a block of information containing the requested word to be transferred to main memory. When the block transfer has been completed, the requested word is accessed in main memory. The time t_B required for the block transfer is called the *block-replacement*, or *block-transfer*, time. Hence we have $t_{A_2} = t_B + t_{A_1}$. Substituting into Eq. (5.3) yields

$$t_A = t_{A_1} + (1 - H)t_B \tag{5.4}$$

Block transfer requires a relatively slow IO operation; therefore t_B is usually much greater then t_{A_1}. Hence $t_{A_2} \gg t_{A_1}$ and $t_{A_2} \approx t_B$.

Figure 2.1: A sample page containing *embedded* as well as *displayed* expressions.

However, the present study doesn't only discuss about the content of the database and its organization. It is also focussed on some other issues like measuring the comprehensiveness of the corpus, analyzing it for several research considerations and designing an automated tool for testing and evaluating the performance of an expression recognition system.

The rest of this chapter is organized as follows. The structure of the proposed database, its content, sampling procedure, generation of groundtruth are outlined in Section 2.2. Section 2.3 presents a statistical study of the database used to measure the comprehensiveness of the proposed corpus. Also, this section contains a linguistic analysis to assist identification of embedded expressions in scientific documents. Next, Section 2.4 demonstrates how the proposed corpus contributes towards testing of an expression recognition system. Section 2.5 summarizes the chapter.

2.2 Organization of the Database

The proposed database contains 400 document images containing 2459 displayed and 3101 embedded expression fragments. Both real (297 pages) and synthetic (generated by TEX or Microsoft Word) documents (103 pages) are present in the data set. Real documents are collected from (i) books of various branches of science, (ii) science journals, (iii) proceedings of technical conferences, (iv) question papers (College/University level examination), etc.

Synthetic documents are selected from sources that are available in Microsoft Word or TEX format. Several electronically available journals, conference proceedings, Internet sites related to various science subjects were considered for this purpose. A few pages were selected from the technical articles published by the members of our research unit. Documents in the database are categorized into three groups depending on the abundance of expressions in the documents. Group I refers to those documents where the number of expressions per page is relatively less compared to other two categories. Similarly, group II points to those pages where density of expressions is higher than that of group-I pages and documents under group III show the highest density of expressions. A summary of the collected samples is given in Table 2.1.

Several factors influence the choice of materials, some of which are described below:

- *Relative frequency of expressions*: The documents show variation in the number of expressions contained in them. Sample documents are divided into three groups based on the number of displayed expressions and percentage of sentences (per page) containing embedded expressions (see Table 2.1).

- *Nature of expressions*: Documents are selected from various branches of science to cover a wide range of expressions that may appear in the literature. The details of the specific topics covered in the database are discussed later. Pages containing expressions having varying geometric structures and layouts are considered to make the data set a representative one.

- *Variations in typeset*: The data set contains documents published using old mechanical typeset as well as those printed by offset and other modern machines.

- *Page layout*: Documents may be printed in single or multi-column format. Apart from text and expressions, they may contain graphs, charts, illustrations, half-tone pictures, etc.

Table 2.1: Coverage of the Document Database

Group label	Source	#Samples (pages)	Avg. no. of displayed exps per page	% of sentences containing embedded exps
Group I	Real	116	2.16	7.61%
	Synthetic	44	3.07	8.82%
Group II	Real	101	6.24	19.23%
	Synthetic	32	7.11	17.05%
Group III	Real	80	11.45	40.57%
	Synthetic	27	10.61	38.12%
Total		400	6.11	20.04%

- *Aging effect*: Many important scientific/technical materials which are not available in electronic form are older than one hundred years. OCR conversion of these documents into electronic form remains a big challenge. The data set contains samples from older as well as recently published materials to reflect this aging effect.

- *Other degradations*: Photocopied versions, pages from bound journals, damaged documents, etc. are also present in the data set to represent different levels of degradation and distortion.

2.2.1 Digitization of Samples

HP flatbed scanners (Scanjet 5470C and Scanjet ADF) are used to digitize real samples into `TIFF` files. Documents are scanned in Gray scale and resolution varies from 150 dpi to 600 dpi. Old materials and documents printed in smaller font sizes are scanned with higher resolutions. Scanning at different dpi-values helps to study the effect of resolution on recognition performance. On the other hand, no scanning is involved for synthetic documents and therefore, corresponding images are free from common digitization errors. These noise-free binary images are generated either by TEX or by Microsoft Word editor.

2.2.2 Format of the Groundtruthed Data

In the current database, each sample page (say, `docxxx.tif`) generates 3 files with extensions `.atr`, `.emb` and `.dis`, respectively, as shown in Figure 2.2. The `docxxx.atr` file contains a set of attributes defined for the page `docxxx.tif`. Some of the attributes are the `page ID`, `publication information`, `page type` (*book/journal/question paper/etc.*), etc. Attributes like `degradation type` (*original/photocopy*), `salt/pepper noise` (*yes/no*), `blurred` (*yes/no*), etc. refer to the page condition and describe the visual condition of a given sample page. Many of these attributes are identical to those used in UW Document Image Database [87].

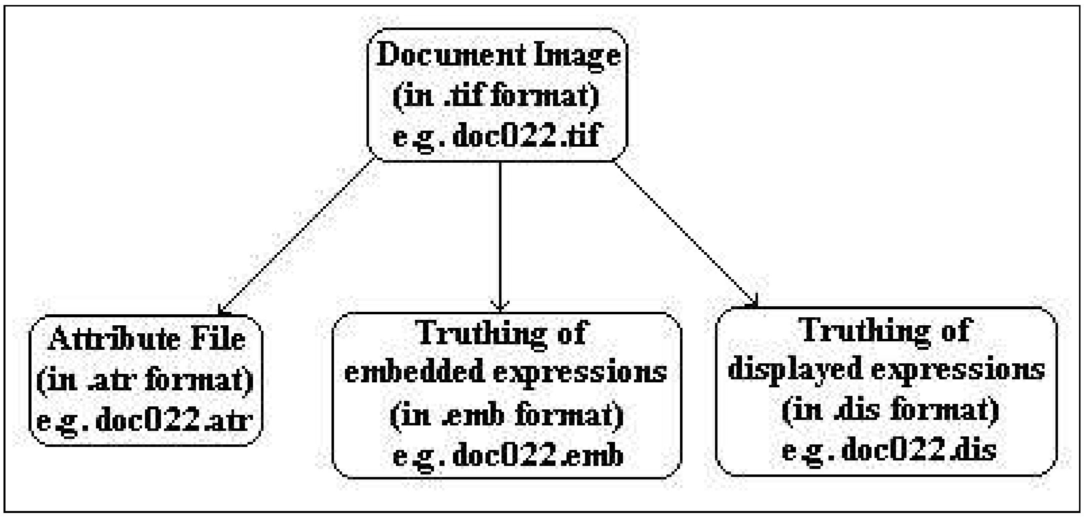
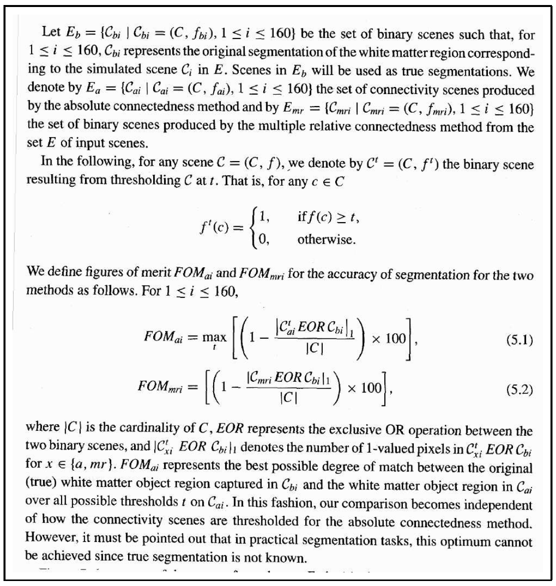


Figure 2.2: Groundtruthing of scientific document images.

Documents in the database show variety in their page layout. Some of the documents are in single column while others are in multi-column format. Several documents contain graphs, charts, illustrations, half-tones, etc. Figure 2.3 shows a few documents present in the database. However, entire page is not considered for generation of groundtruth and only the blocks containing text and mathematical expressions are considered. Several techniques (a summary of them can be found in [54]) have been proposed for automatic analysis of page layout and a modified version of the technique proposed by Pavlidis and Zhou [86] has been employed in our system to locate the blocks containing text and expressions.

Groundtruthing of Embedded Expressions: The file, `docxxx.emb` describes truth for the embedded expressions contained in the page `docxxx.tif`. Embedded expressions are recorded along with the sentences containing them. A sentence is said to have one or more embedded expressions if it would need the use of `math mode` had the sentence been prepared using $\TeX$. Approaches presented in [24, 33, 55, 58, 69, 95, 100] were studied to develop an automatic way of identifying zones containing embedded expressions in the documents. But it is observed that frequent manual intervention is needed to make



Let $E_b = \{\mathcal{C}_{bi} \mid \mathcal{C}_{bi} = (C, f_{bi}), 1 \leq i \leq 160\}$ be the set of binary scenes such that, for $1 \leq i \leq 160$, $\mathcal{C}_{bi}$ represents the original segmentation of the white matter region corresponding to the simulated scene $\mathcal{C}_i$ in E. Scenes in E_b will be used as true segmentations. We denote by $E_a = \{\mathcal{C}_{ai} \mid \mathcal{C}_{ai} = (C, f_{ai}), 1 \leq i \leq 160\}$ the set of connectivity scenes produced by the absolute connectedness method and by $E_{mr} = \{\mathcal{C}_{mri} \mid \mathcal{C}_{mri} = (C, f_{mri}), 1 \leq i \leq 160\}$ the set of binary scenes produced by the multiple relative connectedness method from the set E of input scenes.

In the following, for any scene $\mathcal{C} = (C, f)$, we denote by $\mathcal{C}^t = (C, f^t)$ the binary scene resulting from thresholding $\mathcal{C}$ at t. That is, for any $c \in C$

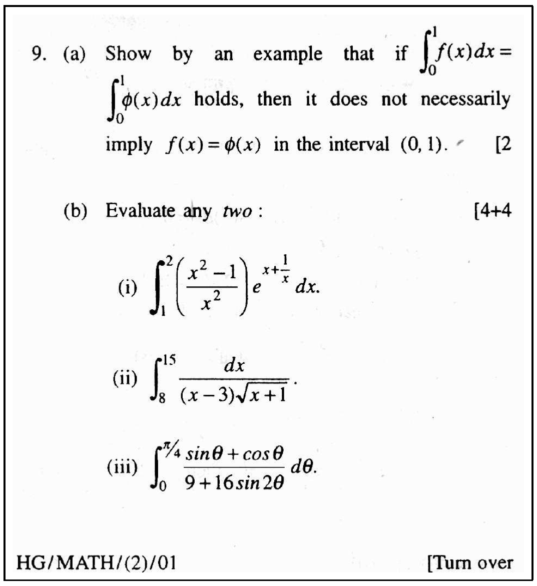
$$f^t(c) = \begin{cases} 1, & \text{if } f(c) \geq t, \\ 0, & \text{otherwise.} \end{cases}$$

We define figures of merit FOM_{ai} and FOM_{mri} for the accuracy of segmentation for the two methods as follows. For $1 \leq i \leq 160$,

$$FOM_{ai} = \max_t \left[\left(1 - \frac{|\mathcal{C}^t_{ai}\, EOR\, \mathcal{C}_{bi}|_1}{|C|} \right) \times 100 \right], \tag{5.1}$$

$$FOM_{mri} = \left[\left(1 - \frac{|\mathcal{C}_{mri}\, EOR\, \mathcal{C}_{bi}|_1}{|C|} \right) \times 100 \right], \tag{5.2}$$

where $|C|$ is the cardinality of C, EOR represents the exclusive OR operation between the two binary scenes, and $|\mathcal{C}^t_{xi}\ EOR\ \mathcal{C}_{bi}|_1$ denotes the number of 1-valued pixels in $\mathcal{C}^t_{xi}\ EOR\ \mathcal{C}_{bi}$ for $x \in \{a, mr\}$. FOM_{ai} represents the best possible degree of match between the original (true) white matter object region captured in $\mathcal{C}_{bi}$ and the white matter object region in $\mathcal{C}_{ai}$ over all possible thresholds t on $\mathcal{C}_{ai}$. In this fashion, our comparison becomes independent of how the connectivity scenes are thresholded for the absolute connectedness method. However, it must be pointed out that in practical segmentation tasks, this optimum cannot be achieved since true segmentation is not known.



9. (a) Show by an example that if $\int_0^1 f(x)dx = \int_0^1 \phi(x)dx$ holds, then it does not necessarily imply $f(x) = \phi(x)$ in the interval $(0, 1)$. [2

(b) Evaluate any *two* : [4+4

(i) $\displaystyle\int_1^2 \left(\frac{x^2-1}{x^2}\right) e^{x+\frac{1}{x}} dx.$

(ii) $\displaystyle\int_8^{15} \frac{dx}{(x-3)\sqrt{x+1}}.$

(iii) $\displaystyle\int_0^{\pi/4} \frac{\sin\theta + \cos\theta}{9 + 16\sin 2\theta} d\theta.$

HG/MATH/(2)/01 [Turn over

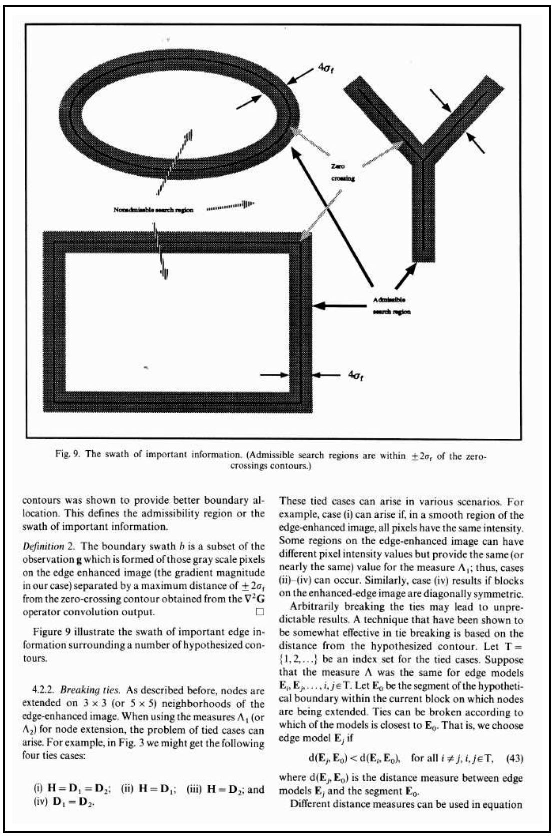

$4\sigma_f$
Zero crossing
Nonadmissible search region
Admissible search region
$4\sigma_f$

Fig. 9. The swath of important information. (Admissible search regions are within $\pm 2\sigma_f$ of the zero-crossings contours.)

contours was shown to provide better boundary allocation. This defines the admissibility region or the swath of important information.

Definition 2. The boundary swath b is a subset of the observation **g** which is formed of those gray scale pixels on the edge enhanced image (the gradient magnitude in our case) separated by a maximum distance of $\pm 2\sigma_f$ from the zero-crossing contour obtained from the $\nabla^2 \mathbf{G}$ operator convolution output. □

Figure 9 illustrate the swath of important edge information surrounding a number of hypothesized contours.

4.2.2. *Breaking ties.* As described before, nodes are extended on 3×3 (or 5×5) neighborhoods of the edge-enhanced image. When using the measures Λ_1 (or Λ_2) for node extension, the problem of tied cases can arise. For example, in Fig. 3 we might get the following four ties cases:

(i) $\mathbf{H} = \mathbf{D}_1 = \mathbf{D}_2$; (ii) $\mathbf{H} = \mathbf{D}_1$; (iii) $\mathbf{H} = \mathbf{D}_2$; and (iv) $\mathbf{D}_1 = \mathbf{D}_2$.

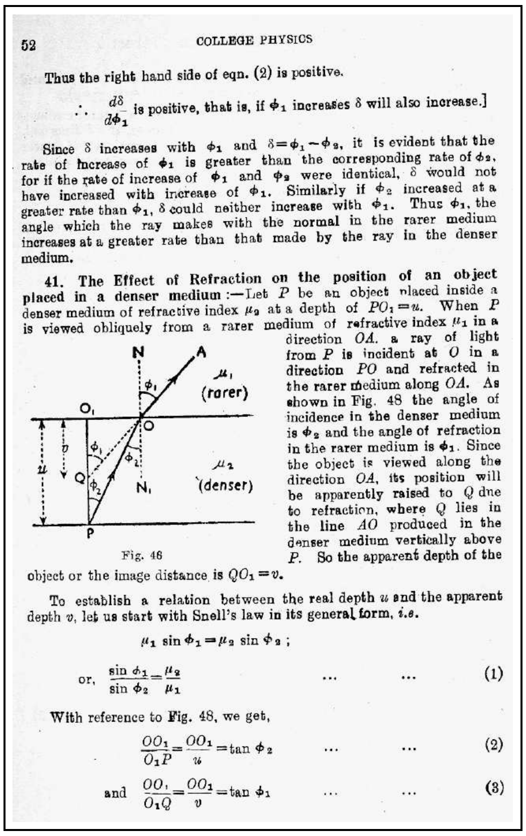
These tied cases can arise in various scenarios. For example, case (i) can arise if, in a smooth region of the edge-enhanced image, all pixels have the same intensity. Some regions on the edge-enhanced image can have different pixel intensity values but provide the same (or nearly the same) value for the measure Λ_1; thus, cases (ii)–(iv) can occur. Similarly, case (iv) results if blocks on the enhanced-edge image are diagonally symmetric.

Arbitrarily breaking the ties may lead to unpredictable results. A technique that have been shown to be somewhat effective in tie breaking is based on the distance from the hypothesized contour. Let $\mathrm{T} = \{1, 2, \ldots\}$ be an index set for the tied cases. Suppose that the measure Λ was the same for edge models $\mathbf{E}_i, \mathbf{E}_j, \ldots, i, j \in \mathrm{T}$. Let $\mathbf{E}_0$ be the segment of the hypothetical boundary within the current block on which nodes are being extended. Ties can be broken according to which of the models is closest to $\mathbf{E}_0$. That is, we choose edge model $\mathbf{E}_j$ if

$$\mathrm{d}(\mathbf{E}_j, \mathbf{E}_0) < \mathrm{d}(\mathbf{E}_i, \mathbf{E}_0), \quad \text{for all } i \neq j,\ i, j \in \mathrm{T}, \tag{43}$$

where $\mathrm{d}(\mathbf{E}_j, \mathbf{E}_0)$ is the distance measure between edge models $\mathbf{E}_j$ and the segment $\mathbf{E}_0$.

Different distance measures can be used in equation



52 COLLEGE PHYSICS

Thus the right hand side of eqn. (2) is positive.

$\therefore \dfrac{d\delta}{d\phi_1}$ is positive, that is, if ϕ_1 increases δ will also increase.]

Since δ increases with ϕ_1 and $\delta = \phi_1 - \phi_2$, it is evident that the rate of increase of ϕ_1 is greater than the corresponding rate of ϕ_2, for if the rate of increase of ϕ_1 and ϕ_2 were identical, δ would not have increased with increase of ϕ_1. Similarly if ϕ_2 increased at a greater rate than ϕ_1, δ could neither increase with ϕ_1. Thus ϕ_1, the angle which the ray makes with the normal in the rarer medium increases at a greater rate than that made by the ray in the denser medium.

41. The Effect of Refraction on the position of an object placed in a denser medium :—Let P be an object placed inside a denser medium of refractive index μ_2 at a depth of $PO_1 = u$. When P is viewed obliquely from a rarer medium of refractive index μ_1 in a direction OA. a ray of light from P is incident at O in a direction PO and refracted in the rarer medium along OA. As shown in Fig. 48 the angle of incidence in the denser medium is ϕ_2 and the angle of refraction in the rarer medium is ϕ_1. Since the object is viewed along the direction OA, its position will be apparently raised to Q due to refraction, where Q lies in the line AO produced in the denser medium vertically above P. So the apparent depth of the object or the image distance is $QO_1 = v$.

Fig. 46

To establish a relation between the real depth u and the apparent depth v, let us start with Snell's law in its general form, *i.e.*

$$\mu_1 \sin\phi_1 = \mu_2 \sin\phi_2 ;$$

$$\text{or,} \quad \frac{\sin\phi_1}{\sin\phi_2} = \frac{\mu_2}{\mu_1} \quad \ldots \quad \ldots \quad (1)$$

With reference to Fig. 48, we get,

$$\frac{OO_1}{O_1P} = \frac{OO_1}{u} = \tan\phi_2 \quad \ldots \quad \ldots \quad (2)$$

$$\text{and} \quad \frac{OO_1}{O_1Q} = \frac{OO_1}{v} = \tan\phi_1 \quad \ldots \quad \ldots \quad (3)$$

Figure 2.3: A few document images present in the database.

the results error-free. However, addition of some n-gram based linguistic properties (explained later) did help a lot to improve the identification process.

The truthed data for a page is contained within <page> and </page> tag pairs. The image file name is stored within tags namely, <imagefile> and </imagefile>. A sentence containing embedded expressions is recorded within the tag pairs <sentence> and </sentence>. In a single page, since multiple sentences can contain embedded expressions, there are multiple instances of <sentence> and </sentence> tag pairs. On the other hand, a sentence may consist of multiple text lines and therefore, a line content is enclosed by <line> and </line> tag pair. An upper level tag structure for groundtruthing of embedded expressions is presented below:

```
<page>
 <imagefile> ... </imagefile>
 <sentence>
   <line>
    Bounding box of the line is recorded.
    Next, text and math portion, if any are
    recorded separately.
   </line>
   <line>
    ...
   </line>
   .
   .
   .
 </sentence>
 <sentence>
   ...
 </sentence>
 .
 .
 .
</page>
```

Each text line of a sentence containing an expression is separately marked with a pair of (x, y) coordinates for top-left and bottom-right corners of the minimum upright rectangular box (called *bounding box* and recorded within <bbox> </bbox> tag pairs) enclosing that text line. The expression zones within a text line are highlighted by the bounding box coordinates of the expression zone. The text and the expression portions of a text line are separately truthed within <text> </text> and <math> </math> tag pairs, respectively. Tag structure for a line is shown below:

```
<line>
 <bbox> ... </bbox>
 <text>
  ...
 </text>
 <math>
   ...
 </math>
 .
 .
 .
</line>
```

For each embedded expression, its enclosing bounding box is recorded within <bbox> </bbox> tag pairs. Next, a <GC> </GC> tag pairs is used to indicate the Geometric Complexity (`GC`) of the expression. Geometric complexity of an expression is measured by the number of horizontal lines (on which expression symbols are arranged) found in that expression. For example, symbols of expression shown in Figure 2.4(a) are arranged on `9` horizontal lines as shown in Figure 2.4(b). Therefore, `GC` becomes `9` for this expression. The dominant baseline [113] of an expression is treated as `level 0` and `level` number increases above and decreases below the baseline.

Different zooming factor and resolutions may give different looks for the same expression, but we view that GC is not much affected by these changes. Therefore, this value is recorded along with each expression to identify expression's geometric complexity. However, it is true that the value of GC for an expression in rare occasion can vary from its actual value and we have observed it for expressions showing unusual typography. Such errors are attributed to non-uniformity in typography found in documents printed with very old and non-standard technology that lacks in proper layout in typing expressions (Section 5.4 of Chapter 5 demonstrates this problem). We experience that occurrence of such problem is really rare in number.

Next, content of the embedded expression is presented. For this purpose, MathML[1]

[1]see W3C Math Home at http://www.w3.org/Math/

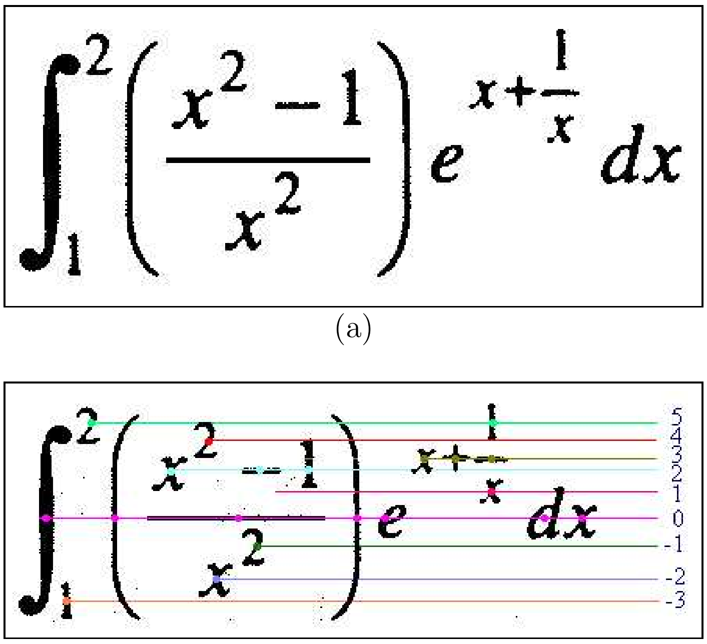


Figure 2.4: Geometric complexity of expressions: an example.

presentation tags are used. However, for a symbol (where it is operator, identifier, or numeral) three additional tag pairs namely, <level> </level>, <style> </style> and <truth> </truth> are used. For example, if t is an identifier used in the expression, its MathML representation (i.e. <mi> t </mi>) is extended as follows:

```
<mi>
 <level> ... </level>
 <style> ... </style>
 <truth> ... </truth>
</mi>
```

The `level` indicates horizontal level number at which the symbol appears. For example, symbols of the expression in Figure 2.4(a) appear in one of the nine horizontal levels numbered with `-3` to `5` including `0`. The `style` indicates the type style (`n`: normal, `b`: bold, `i`: italic, `bi`: bold-italic) of the symbol. The identity of the symbol is given within the <truth> </truth> tag pairs.

Algorithm can be designed for automatic computation of `GC` and symbol `levels`. One such algorithm [77] is explained in section 5.2 of Chapter 5. On the other hand, detection of symbol `style` is done by following the algorithms outlined in [19, 21]. It is noted that the expression symbols are often typed with a font style, which is different from the dominant style of the document text. In several cases, variations in font faces are also observed, i.e. the expressions are printed in a font different from the one used for remaining text of the document. Statistics regarding different type styles used in expressions are presented in Table 2.2. For computing type style of an expression it is found that the style of all expression symbols, in many cases, may not be unique and therefore, the results presented in Table 2.2 show dominant (determined by the majority of symbols) style in the expressions.

Table 2.2: Statistics on Type Styles

Style	Expression Level Statistics	
	Embedded	Displayed
Regular	186 (6%)	271 (11%)
Italics	2,078 (67%)	1,254 (51%)
Bold	558 (18%)	516 (21%)
Bold-Italics	279 (9%)	418 (17%)

In case of a page having multi-column layout, expressions are truthed following the normal reading sequence determined manually. In database, 1084 (out of 5402) sentences are found to have embedded expressions and groundtruth for each of these sentences is available in the corpus. The CDROM attached with this thesis contains groundtruth for five sample images. Groundtruth of embedded expressions contained in a document image (say, `sample1.tif`) is available in the corresponding `.emb` file (e.g. `sample1.emb` corresponds to `sample1.tif`).

Groundtruthing of Displayed Expressions: The file, `docxxx.dis` truths the displayed expressions contained in the image `docxxx.tif`. Automatic identification of displayed expressions in document images has been achieved by the algorithm proposed in [20]. For a sample page, all displayed expressions are truthed within <page> </page> tag pairs. Each expression is truthed under <math> </math> tags. For a multi-line expression, two consecutive expression lines are linked together with a *Thread* pointer, if they are part of the same expression. So, if the *Thread* pointer (which can have only binary values) for any expression is `1` then the current expression continues to the next line. In case of a page having multi-column layout, expressions are truthed following the normal reading sequence (manually determined).

Within a pair of <math> </math> tags, truthing starts with <bbox> </bbox> tags giving the bounding box coordinates for the expression. Next, geometric complexity of the expression is recorded using the tag pairs <GC> </GC>. Expression sequence

Table 2.3: Coverage of the Expression Dataset

Source	Area	Expressions	Remarks
Expressions found in 390 technical and scientific documents collected under Part I of the proposed database.	Algebra	439	This data set shows various image level noise like degradation due to aging, digitization errors, etc. Expressions are scanned with resolution varying from 75 to 600 dpi.
	Calculus	205	
	Differential equations	313	
	Integrals	351	
	Logic and set theory	186	
	Statistics and probability	176	
	Trigonometry and geometry	171	
	Vector	254	
	Miscellaneous	304	
AsTeR data set [91]	Series	5	This data set is freely available at `http://www.cs.cornell.edu/info/people/raman/aster/`. These TEXgenerated expressions are free from any digitization errors. The resolution is 300 dpi.
	Logarithms	4	
	Fractions	9	
	Roots	4	
	Sums	3	
	Superscripts and Subscripts	9	
	Limits	2	
	Matrix	1	
	Trigonometry	8	
	Integrals	6	
	Miscellaneous	9	
Total		2,459	

number, if any exists, is given under the <expno> </expno> tags. Content of the expression is truthed in a similar way used for truthing the embedded expressions. Only difference is the additional presence of <bbox> </bbox> tag pairs for each symbol of a displayed expressions.

In the corpus, there are 2399 displayed expressions and groundtruth data for each expression is available in 400 `docxxx.dis` files. In addition, 60 synthetic expressions are selected from the data set used in AsTeR research [91]. Reasons for this choice are (i) it contains examples from various branches of mathematics. The expressions show wide variations in their structural complexity making the data set a representative one; (ii) for each expression its corresponding correct TEX string is available and these strings are used to generate the groundtruth; (iii) the data set is free and globally available through the Internet, thus allowing comparative study. Therefore, the number of expressions present in Part-II, in total, is 2459. Table 2.3 gives an idea of how the expressions are divided among different topics and purposes.

The CDROM attached with this thesis contains groundtruth for five sample pages. Groundtruth of displayed expressions contained in a document image (say, `sample1.tif`) is available in the corresponding `.dis` file (e.g. `sample1.dis` corresponds to `sample1.tif`).

2.2.3 Implementation of the Groundtruthing Process

From the above discussion it follows that the proposed groundtruthing process consists of several steps as outlined next. Implementations of all these steps were not fully automated and frequent manual interventions were required for realisation of many steps. Computation of some information was done automatically, whereas others required semi-automatic approach. Execution mode for each step is described below:

1. Scanning of samples,

Comment: manually done. Automatic document feeder (ADF) facility of scanners didn't help much as samples were collected from various sources and could not be given in bulk.

2. Filling up the attributes for sample pages (generation of `.atr` files),

Comment: manually done. Basically a template containing different attribute fields was prepared and this template form was filled in for each sample.

3. Marking of text blocks (expressions are treated as part of text blocks),

Comment: Semi-automatic. A modified version of the approach described in [86] was used and results were accepted after manual inspection.

4. Identification (in form of bounding box information) of embedded and displayed expressions in each page,

Comment: Semi-automatic. Displayed expressions were located by following the approach in [20] and embedded expressions were identified by following an approach designed after consulting the existing literature. In both cases, identification results were displayed using graphical interface and manual intervention was involved to check the results and correct (by adjusting the bounding box) them whenever required. Later on. it is found that the approach presented in the Chapter 3 of this thesis can substantially improve the identification results.

5. Groundtruthing of embedded expressions:

   a. Marking of sentences containing embedded expressions,

   Comment: Semi-automatic. Once embedded expressions are marked, sentences containing one or more embedded expressions and ending with a full stop mark are (in majority of cases) located automatically. However, boundary of a sentence containing expression fragments but not ending with a full-stop mark are identified manually.

   b. For each such sentence, computation of bounding box information for text lines,

   Comment: Automatic. Once sentences containing embedded expressions are marked, constituent text line boundaries are located automatically.

   c. In each text line recording of OCR results for (i) text and (ii) expression parts

   Comment: OCR of text part is done automatically. A commercial OCR system was used for this purpose. Execution details for OCR of expression parts is given later.

   d. Generation of final marked-up description (`.emb` file)

   Comment: Manual. Values for all tag pairs are ready at this stage but they are organized manually.

6. Groundtruthing of displayed expressions:

   a. OCR of expression,

   Comment: Execution details for OCR of expression parts is given later.

   b. Generation of final marked-up description (`.dis` file)

   Comment: Manual. Values for all tag pairs are ready at this stage but their integration was done manually.

Two steps namely, (i) OCR of expressions (steps 5.c.(ii) and 6.a) and (ii) generation of marked-up description files involve different sub-tasks important in the context of overall groundtruthing process. OCR of an expression involves following steps:

i) Computation of expression's geometric complexity (GC field),

Comment: Semi-automatic. Approach presented in [77] was used to determine an expression's `GC` which is manually inspected for final acceptance.

ii) For each expression symbol, computation its position, `level` and `style`,
Comment: Semi-automatic. Position i.e. bounding box info is computed automatically. The `level` and `style` are determined by following approaches described in [77] and [19], respectively, and verified manually.
iii) Symbol recognition,
Comment: Semi-automatic. Classifier described in [20] is used for symbol recognition. Mis-classified and rejected symbols are identified manually. The multiple-classifier approach presented in Chapter 4 of this thesis substantially improves the classification results.
iv) Structure analysis and generation TEX string for the expression,
Comment: Semi-automatic. Approach described in [77] is used for this purpose and results are manually checked and corrected (if needed).
v) TEX to MathML conversion.
Comment: Automatic. A freely available tool[2] is used to convert TEX strings in their corresponding MathML (using presentation tags) versions.
vi) Insertion of expression number.
Comment: Manual. For a displayed expression the expression sequence number, if any exists, is entered manually.

Similarly, generation of final marked-up description files (steps 5.d and 6.b) requires integration of all necessary information (i.e. tag values) together. This is done by adding non-MathML tags (for which values are computed at different steps) within and outside the MathML description obtained after OCR of expressions. For example, tags like <page> and </page>, <imagefile> and </imagefile> etc. are positioned outside the MathML tags, whereas tags likes <level> and </level>, <style> and </style>, etc. are integrated inside the MathML representation.

Four persons voluntarily contributed in the groundtruthing process, mainly for implementation of semi-automatic steps described above. Contributors (acknowledged later) were students of Computer Science or allied fields and did not require extensive training to understand the process. Initially, the goal of this study was clearly stated to them and then all necessary programs along with some graphical interfaces (like image display with associated controls like drawing of bounding box, etc., interface to check symbol recognition results and to correct them if necessary, utility to check final TEX string for an input expression, etc.) were supplied to the students who were explained about their

[2]Many such tools are available in the net. The conversion utility used in this experiment was found at http://hutchinson.belmont.ma.us/tth/mml/ttmmozform.html

respective role/responsibility involved in the process.

Volunteers mainly contribute towards checking of results obtained by executing different program modules and then correcting the results whenever required. Results for missed parts (e.g. an expression zone, recognition of any symbol, etc.) are added manually. Students also help in generating the final marked-up description of a page and this step requires extensive manual intervention as for each page MathML and non-MathML tags with values were manually arranged according to the proposed format as described in the preceding section (i.e. section 2.2.2).

It was difficult to estimate the exact man-power that was required to generate the groundtruthed data for 400 pages as the effort was voluntary in nature and the work was done in a sporadic way rather than in a continuous manner. However, considering the tools (most of them are semi-automatic) available with us, a rough estimation indicates that if the entire work to have been done at a stretch then a time period of 6 man-month would have been required. This estimates does not include the time required for validation of the data as discussed next.

2.2.4 Validation of the Truthed Data

As correctness of the truthed data is very much important for any benchmark database, we attempted to validate the generated data. Since frequent manual intervention was involved at several steps of the groundtruthing process, further checking of the generated data reveals many errors. A preliminary-level effort has been made to remove many of such errors and further attempt to make the dataset error-free is in progress.

Two persons[3] (apart from those involved in the groundtruthing process) did help us for the validation of the truthed data. As the final description of an input is coded in a marked-up language, it is difficult to check the validity of the data by looking at `.emb`/`.dis` files. Therefore, we did design an additional utility program to generate a global description of a page from its corresponding truthed data. For a page, there are two upper level descriptions, one is generated for the embedded expressions (using the associated `.emb` file) and another one for displayed expressions (using the `.dis` file). Expressions[4] contained in a page were given in the description file in their corresponding TEX format.

[3]These two persons were employed in one of our departmental projects on document image analysis and partially involved in the process of validating the truthed data.

[4]As groundtruthed files contain expressions in MathML format, MathML to TEX conversion is done following a freely available utility found at http://www.orcca.on.ca/MathML/texmml/mmltotex.html.

The description files are in `.tex` style. Compilation of such a description file (say, `sample1_emb_desc.tex` generates a corresponding `.dvi` (or `.ps`/`.pdf`) format which provides better convenience to check the validity of the truthed data[5]. This is done by visually comparing the content and appearance of the expressions in the image (say, `sample.tif`) as well as in TEX format. Moreover, such an upper level description provides ways to further check the `GC` value of an expression, the `style`, `level` and identity (`truth`) data for expression symbols. OCR accuracy for the text portion is also checked at this level.

Apart from these, the description files contain information for bounding box coordinates of different entities (e.g. line, expression, expression symbol, etc.) and one can check the validity of such information by looking at the coordinate values of an entity's bounding box and its position (by a rough estimation or if needed, by checking *cursor*'s position on the image) in the corresponding image file. However, as this way of checking bounding box information lacks in convenience, design of a more user-friendly interface for this purpose is in progress.

First level validation task is nearly completed and errors are mostly found in some cases of the following entries:
(i) The *level* information associated with the symbols of displayed expressions: errors generated by the program used to capture *level* of expression symbols have not been properly corrected in many cases.
(ii) Truthing of text parts: a commercial OCR was used to recognize text portions and OCR made some mistakes. However, such mistakes are less serious than the other type of errors as a pure expression recognition system has little to do with the recognition of text parts.
(iii) Spelling of some user-defined tags: spellings of some tags were found wrong, e.g. <bbox> in some cases appears as <box>, <truth> as <truth>,etc. Though templates were used for such tags and therefore such errors are unexpected. But these errors occurred because of casual editing during integration of non-MathML and MathML tags. It is mentioned earlier that the final organization of tags was done manually using an text editor. However, occurrences of such errors were less in number and detected automatically by the utility program used for generation description files (e.g. `sample1_emb_desc.tex`). This program gives "invalid tag" message whenever it encounters such wrongly spelt tags

[5]For each sample image, two TEX files are available in the CDROM attached with this thesis. For example, `sample1_emb_desc.tex` and `sample1_dis_desc.tex` are associated with the image `sample1.tif`. The file `sample1_emb_desc.tex` gives an upper level description of the lines containing embedded expressions whereas the file `sample1_dis_desc.tex` provides description of the displayed expressions.

(i.e. not present in the pre-compiled tag list).
(iv) Placement of punctuation marks: Punctuation mark (e.g. ‘,’/‘.’/etc.) placed at the end of an embedded expression are sometimes coded along with the expression (i.e. within <math> </math> tags) and sometimes coded as text (i.e. within <text> </text> tags). An attempt has been made to maintain a uniformity for this purpose.

2.3 Statistical Investigation of the Corpus

Initially, the data set is investigated to measure its comprehensiveness for expression recognition research. Next, some additional studies are presented for identification of embedded expressions and issues related to testing an expression recognition system.

Table 2.4: Occurrence rate of symbol categories

Symbol (#Classes)	#Occurrences in		Total Count
	Embedded	Displayed	
Roman Letter (52)	9,687	20,361	30,048 (0.36)
Arabic Numeral (10)	3,544	6,882	10,426 (0.13)
Greek Symbol (41)	1,654	3,493	5,147 (0.06)
Calligraphic Letter (16)	71	309	380 (0.01)
Mathematical Operator (39)	1,541	10,907	12,448 (0.15)
Relational Operator (39)	954	4,693	5,647 (0.07)
Arrow Symbol (32)	612	2,123	2,735 (0.03)
Bracket Symbol (06)	2,127	4,314	6,441 (0.08)
Misc. Symbol (33)	945	3,511	4,456 (0.05)
Punctuation Mark (06)	2,268	2,695	4,963 (0.06)
Total (274)	23,403	59,288	82,691 (1.00)

2.3.1 Study of Comprehensiveness

Comprehensiveness of the proposed corpus is measured against some basic aspects, which are important in any expression recognition research. The presence of different symbols, their occurrence frequencies, appearances of different two-dimensional (2-D) structures (e.g. *super/subscript*, *root*, *fraction*, etc.) are studied to understand the corpus coverage.

For each 2-D structure, an idea of Degree of Nestedness (`DoN`) is introduced. For example, consider three structures (i) e^x, (ii) e^{x^x} and (iii) $\sqrt{1+\frac{1}{x}}$. First two *superscript* structures show values for `DoN` equal to `1` and `2`, respectively. On the other hand, the third structure is a mixed one and its `DoN` equals to `2` since a *fraction* structure is nested inside the *root*. For a 2-D structure, its `DoN` value is automatically computed from the operator tag pairs associated with that structure. For example, e^x uses only one pair of MathML presentation tags namely, <msup> </msup> giving its `DoN` equals to 1 whereas, $\sqrt{1+\frac{1}{x}}$ uses one pair of <mfrac> </mfrac> tags within <msqrt> </msqrt> tags giving `DoN` equals to 2.

It is observed that for a 2-D structure `DoN` value doesn't (substantially) vary with change of parameters like font style, etc. In general, it is true that two identical expressions (clearly speaking an expression structure like scripts, fraction, root, etc.) written in two different font style will have the same DoN. This is because symbols within a structure convey their meaning based on their spatial arrangement or layout which doesn't change with different font faces, type style, character sizes, etc., otherwise, mathematical meaning of a structure would have been changed.

The investigation results are outlined below.

Observation 1. *Expression Symbols*: Symbols occurring in expressions are quite different from those occurring in normal text. Apart from the Roman letters, symbols like Arabic digits, Greek symbols, Mathematical operators, function words, etc. are frequently used to write expressions. Analysis shows that the symbols present in the corpus can be partitioned into 10 categories, namely, (i) Arabic Numerals (*AN*), (ii) Roman Letters (*RL*), (iii) Greek Symbols (*GS*), (iv) Calligraphic Letters (*CL*), (v) Mathematical Operators (*MO*), (vi) Relational Operators (*RO*), (vii) Arrow Symbols (*AS*), (viii) Bracket Symbols (*BS*), (ix) Miscellaneous Symbols (*MS*) e.g. prime('$\prime$'), for all ('$\forall$'), there exists ('$\exists$'), etc., (x) Punctuation Marks (*PM*).

Frequency at which these symbol categories occur in embedded as well as displayed expressions are given in Table 2.4. Distinct number of symbol classes under each category is shown within braces under the first column. Table 2.4 gives an idea about the extra symbols an OCR has to classify for recognizing expressions. Availability of this large number of samples helps one to design suitable classification scheme for expression symbols. For this purpose, training and test set of symbols are marked separately in the corpus.

Next, occurrence frequencies for symbols are computed. The list of symbol frequencies gives an idea about the relative abundance of different symbols in expressions. Based on these statistics, one may design a prototype library where reference symbol are arranged according to their occurrence frequencies to speed up the symbol-recognition process. Moreover, occurrence statistics concerning mathematical operators helps to identify a small group of symbols that show high occurrence frequencies. Table 2.5 presents a few of these operators and their abundance (not the occurrence frequencies) in the set of

Table 2.5: Occurrence Rate of some Operators

No.	Symbol	Abundance (% of expressions in which the symbol appears)	Occurrence Frequency
1	=	94%	12.07%
2	+		8.71%
3	- (Minus)	93%	4.65%
4	/		0.35%
5	(	60%	4.77%
6	)		4.02%
7	Fraction Line	51%	3.42%
8	[	35%	1.57%
9	]		1.57%
10	{	20%	0.74%
11	}		0.61%
12	$<$	16%	0.56%
13	$>$		0.17%

displayed expressions. The high occurrence and shape simplicity of these operators may help one to quickly identify zones containing displayed expressions.

Observation 2. *Operator Structures*: Symbols in expressions are arranged around different operators and form different geometric layouts, some of which are one-dimensional (1-D) in nature while others are two-dimensional (2-D). For example, operators like '+', '-', '=', function words like *sin*, *cos*, *log*, etc. induce 1-D structures while superscripts, subscripts, operators (like '$\sum$', '$\prod$', '$\int$', etc.) with limit expressions, matrix, etc. form the 2-D layout.

Properties of such structures along with their occurrence statistics have been studied in the proposed corpus. Twelve elementary structures have been detected which are 2-D in nature. They are: (i) *Superscript*, (ii) *Subscript*, (iii) *Fraction*, (iv) *Root*, (v) *Overline*, (vi) *Underline*, (vii) *Overbrace*, (viii) *Underbrace*, (ix) *Ellipses*, (x) *Accent*, (xi) *Matrix* and (xii) *Stacking of symbols*.

The number of such structures observed in the data set are reported in Table 2.6. However, sometimes it is observed that structures appear in *nested* mode. A structure is called *nested* if it contains another structure within it. With each class of structures, the observed `DoN` (representing degree of nestedness) are listed in the 3rd column of Table 2.6.

Observation 3. *Structural Complexity of Expressions*: It is discussed earlier that geometric (or structural) complexity, `GC` of an expression is defined by the number of horizontal lines on which the constituent symbols are arranged. Expressions in the database are investigated to check their complexity levels and they are grouped into different

Table 2.6: Occurrence of Operator Structures

Structure	#Occurrences	DoN Values
Superscript	4,267	1, 2, 3, 4
Subscript	3,986	1, 2, 3
Fraction	2,063	1, 2, 3, 4, 6
Root	227	1, 2, 3, 5
Overline	60	1, 2, 3
Underline	13	1, 2
Overbrace	47	1, 2, 4
Underbrace	19	1, 3
Ellipses	828	1
Accent	341	1, 2, 3
Matrix	73	1, 2, 3
Stacking of symbols	154	1, 2

classes based on their `GC` value. Frequencies of these classes in the database are reported in Table 2.7. Results show that embedded expressions are less complex in their structure than that of displayed ones. The highest complexity observed in embedded expressions is 5 whereas for displayed expressions the value of `GC` can be as high as 15.

2.3.2 Linguistic Properties of Sentences Containing Embedded Expressions

A linguistic analysis of the sentences present in scientific documents indicates that a word level N-gram model could be of great help to categorize a sentence in a technical document into one of the two categories, namely, with or without expressions. This analysis is motivated by Zipf's law [115] which is formally defined as $P_r \sim 1/r^a$, where P_r is the frequency of occurrence of the r-th ranked item and a is close to 1.

In other words, the law says that the n-th most common word in a human language text occurs with a frequency inversely proportional to n. The implication of this law is that there is always a set of words that are specific to a particular context. In our approach, use of embedded expression is treated as the context and based on this, sentences with and without expressions are labeled with one of the two categories, namely, (i) Category-I (C_1) in which sentences do not contain any expression and (ii) Category-II (C_2) where each sentence contains one or more expression fragments. To generate N-gram profiles of a category the following steps are performed.

Step 1. Digits, punctuation marks, expression portions (in case of sentences containing

Table 2.7: Geometric Complexity of Expressions

Geometric	Number of Expressions	
Complexity (`GC`)	Embedded	Displayed
1	695	347
2	1,498	489
3	481	628
4	374	427
5	53	109
6	–	202
7	–	93
8	–	70
9	–	31
10	–	16
11	–	23
12	–	12
13	–	7
14	–	3
15	–	2

expressions) are discarded so that sentences contain only words.

Step 2. Each sentence is scanned to generate all distinct word level N-grams (for N = 1 to 3) and their frequencies are recorded. To compute bigram and trigram frequencies, each sentence is padded with sufficient blanks on both sides. Stop words are discarded while generating the N-grams.

Step 3. Relative frequency of each N-gram ($w_{1,N}$) is computed as the ratio of its associated count (i.e. the number of its occurrences) to the total number of similar (based on the value of N) N-grams. For example, let $c^i_{1,N}$ be the number of occurrences of i-th N-gram $w^i_{1,N}$ and T_N be the total number of distinct N-grams, then relative frequency $f^i_{1,N}$ of $w^i_{1,N}$ is computed as

$$f^i_{1,N} = \frac{c^i_{1,N}}{\sum_{j=1}^{T_N} c^j_{1,N}} \text{ for } N = 1 \text{ to } 3 \tag{2.1}$$

Step 4. When steps 1–3 are completed, a list of all N-grams along with their frequencies is produced.

Step 5. Sort the list in descending order of frequencies. The resulting list then represents an N-gram frequency for a category.

In our study, a profile for category-I (C_1) is built using 2,655 sentences having nearly 35 thousand words. On the other hand, 870 sentences containing about 12 thousand words have been used to generate the profile for category-II (C_2). A number of observations are followed from an inspection of the N-gram profiles of the categories.

- Many of the stop words show high occurrence frequency in both category profiles. However, note that a list of stop words in this application is different from the standard stop word list used in text retrieval applications. This is so because several words like "if," "then," "where," etc. show significance in the present purpose are important for the purposes of the present study, though in general they are treated as stop words. Therefore, the list of stop words mentioned in step 2 above is specially computed for the current application.

- It is observed that top 150 or so N-grams are highly representative for a particular category. Of course, there is nothing special about rank 150 itself and it is chosen mostly by inspection. In fact, Zipf's law provides a very smooth distribution curve and one could always do more elaborate statistics and choose an optimal cut-off rank.

- The top ranking N-grams in category-I are mostly unigrams that reflect the distribution of the words in the sentences of that category whereas for category-II, presence of frequent bigrams (e.g. *such that*, *the following*, etc.) and trigrams (e.g. *let us consider*, *is given by*, etc.) is also observed. In both categories, there is, of course, a long tail to the distribution of N-grams which goes well beyond 150. These N-grams represent terms which do not show much distinguishing power as far as classification is concerned.

A given sentence is classified by a similarity measure implemented as follows:

- Step 1. All N-grams are generated for a target sentence (S) all N-grams are generated. Let L be the number of such N-grams.

- Step 2. Each N-gram found ($w^k_{1,N}$, for k = 1 to L) in S is searched in the category profile of C_1 and C_2. Let p_k and q_k be the relative frequency of $w^k_{1,N}$ in C_1 and C_2, respectively. If any $w^k_{1,N}$ is not found in the profile of C_1 (or in C_2) then p_k (or q_k) is set to zero.

- Step 3. S is classified as follows:

$$\text{If} \quad \textstyle\sum_{k=1}^{L} p_k > \sum_{k=1}^{L} q_k \quad \text{then } S \in C_1$$

$$\begin{aligned} &\text{Elseif} \quad \textstyle\sum_{k=1}^{L} p_k < \sum_{k=1}^{L} q_k \quad \text{then } S \in C_2 \\ &\text{Otherwise,} \quad \text{Indeterminate} \end{aligned} \tag{2.2}$$

Table 2.8: Correct Classification of Sentences using N-gram Model

Sentence Length (#Words)	< 10	< 10	< 10	< 10	≥ 10	≥ 10	≥ 10	≥ 10
Profile Length (#N-grams)	50	100	125	150	50	100	125	150
Sentences without Exp.	$\frac{217}{239}$	$\frac{225}{239}$	$\frac{232}{239}$	$\frac{234}{239}$	$\frac{398}{424}$	$\frac{407}{424}$	$\frac{416}{424}$	$\frac{419}{424}$
Sentences with Exp.	$\frac{60}{68}$	$\frac{63}{68}$	$\frac{65}{68}$	$\frac{66}{68}$	$\frac{139}{146}$	$\frac{142}{146}$	$\frac{144}{146}$	$\frac{145}{146}$
Accuracies	90.2%	93.8%	96.7%	97.7%	94.2%	96.3%	98.3%	98.9%

This approach for sentence categorization has been tested with 877 sentences (which have not been used to generate category profile of either C_1 or C_2) and the classification results are reported in Table 2.8. These results show some interesting patterns:

- The classification method performs a little better for longer sentences.
- The accuracy improves with the increased length (in terms of the number of N-grams) of the profiles. Overall, the method yields its best performance at a profile length of 150. At this stage, the system misclassified only 13 sentences out of 877, producing an overall classification accuracy of 98.52%.
- The result shows that N-gram model provides important indications to label sentences containing embedded expressions. This feature, therefore, could be integrated with other aspects for robust identification of embedded expressions.

2.4 Issues related to System Testing

Choosing a right set of data is always an important aspect to test any system performance. In case of expression recognition, test data must represent the variability in terms of expression symbols, structures, geometric complexity etc. Selecting adequate number of test samples is another important criterion. The proposed corpus provides necessary support in this direction. Four types of queries can be made by the designers to choose the proper test samples. Queries are of the following nature:

- Type I: *Query for Symbols–* Query can be made to search for documents and in particular, embedded and displayed expressions which contain a particular query *symbol.* The word *symbol* refers to identifiers, constants, operators, etc. that occur in expressions. A query made on document level is replied with a set of document IDs that contain the query symbol. Similarly, individual expressions tagged with document IDs are returned when query is made on expression level. A query processor is implemented by maintaining two different forms of indexing – one at the individual expression level and another at the document level. Symbol name and its relative occurrence frequency are used as index keys. Relative frequency of a symbol ($\mathcal{S}$) refers to an expression level feature and is computed as,

$$\frac{\text{No. of occurrences of } \mathcal{S} \text{ in an expression, } \mathcal{E}}{\text{Total no. of symbols in } \mathcal{E}} \tag{2.3}$$

  A document level indexing is just an aggregation of the results obtained for its constituent expressions. The resultant expressions (or documents) are ranked according to (in descending order) the occurrence frequency of the query symbol in them.

- Type II: *Query for Function Words–* Expressions (or documents) containing a particular function word (e.g. sin, cos, log, max, etc.) can also be queried. This facility is the same as one in Type I explained above. Necessary index structures are maintained for this purpose.

- Type III: *Query for structures*: The presence of structures (e.g., *fraction*, *root*, *script*, *sum*, *product*, etc.) can also be queried. The query result for a particular structure returns the number of expressions (or documents if the query is made at the document level) , respective or document IDs containing the expressions along with a value representing the degree of nestedness `DoN` for each occurrence of the structure. Through this query facility one may be assured that the test set contains samples for the structure one is looking for. The geometric complexity of the structures is also reflected in the associated `DoN` values.

- Type IV: *Query for geometric complexity*: Each expression is tagged with its geometric complexity (`GC`), as explained before. Therefore, one may make pose a query to find expressions (and the documents containing them) having a particular complexity level. A support for such types of queries helps one to understand the geometric variability of expression structures. Moreover, based on the geometric complexity of expressions, one may choose the right data set to test one's expression recognition algorithm designed for expression recognition.

2.5 Summary

In this chapter, the development of a corpus of printed scientific and technical documents is discussed. This study is motivated to facilitate research on automatic recognition of expressions. A large number of samples are collected from various branches of science to make the dataset a representative one. Groundtruthing of expression images follows a new format useful for several research considerations. Later on, Chapter 7 shows how the proposed format helps in a semi-automatic evaluation of expression recognition results. A statistical analysis of the corpus content is presented to show the comprehensiveness of the proposed corpus. Several utilities are provided to assist designing and testing of an expression recognition system.

Assuming expressions are a special kind of visual language, the corpus presented here is expected to help in automatic deduction of language definitions and several probabilistic contextual information (as in the case of natural language processing [18]). This chapter demonstrates the usefulness of such an analysis for identification of embedded expressions. In future, we plan for more similar studies e.g. generation of a probabilistic approach for interpretation of expression structures. The studies presented in [23, 50, 76] have demonstrated, though in a limited domain, the usefulness of a stochastic technique over a deterministic one to resolve ambiguous meanings imposed by different expression symbols. However, formulation of any such probabilistic framework (e.g. Probabilistic Context-Free Grammar, *PCFG*) requires a large representative training data. The present corpus, therefore, is expected to play an important role in this direction.

The issue related to the dissemination of the corpus is treated as an immediate future activity as mentioned later in Chapter 8. Execution of first level checking is nearly completed and a second (and final) level checking of the truthed data is being planned to further check the validity of the truthed data. In this phase, two groups of volunteers (may be hired as in one of our departmental projects we have some budgetary provision for this purpose) will be employed. Two copies of the set of groundtruthed files (say, $\mathcal{G}$) will be generated and let $\mathcal{G}_1$ and $\mathcal{G}_2$ denote these two copies. One group will work on $\mathcal{G}_1$, manually check the validity of turthed data contained in $\mathcal{G}_1$, and correct the errors. Let $\mathcal{G}_1'$ denote the corrected version of $\mathcal{G}_1$. Similarly, other group will concentrate on $\mathcal{G}_2$ to produce $\mathcal{G}_2'$. Apart from the existing tools (developed so far), design of some more user-friendly interface is in progress to assist this manual correction. Next, the contents of $\mathcal{G}'_1$ and $\mathcal{G}'_2$ are to be compared by automatic file comparison utilities like `cmp`, `diff`, etc. Differences will be marked and corrected to produce the final version of the truthed data, $\mathcal{G}'$). The completion of the first level checking took about 3 man-month and based on this experience, it is estimated that another 3 man-month will be required to complete the final validation of the truthed data.

Special Ackowledgement: We express our sincere gratitute towards our students R.P. Ghosh, Md. S. Chowdhury, Alok Datta, and Mangal Chakraborty for their active participation in the groundtruthing process. Partial involvement of our colleagues (who were employed in our departmental projects) Pradip Dey and Udayan Kar is much appreciated.

A paper based on the preliminary version of the study presented in this Chapter was communicated to *the 4th IAPR Int'l. Workshop on Graphics Recognition* (GREC-2001) [43] and we sincerely thank the reviewers for their comments and suggestions on that communication. Later on, the complete work as presented in this Chapter was sent to the *Int'l Journal on Document Analysis and Recognition* (IJDAR) for possible publication. We express our sincere gratitude towards the anonymous reviewers of this paper[6] for their detailed comments and a number of valuable suggestions that enormously helped us to improve the quality of the work.

[6]U. Garain and B.B. Chaudhuri, "A Corpus for OCR research on Mathematical Expressions," *Int'l Journal on Document Analysis and Recognition* (IJDAR), accepted (in press) and an Internet version has been appeared in March 2005 in the IJDAR website.

CHAPTER 3

IDENTIFICATION OF MATHEMATICAL EXPRESSIONS IN DOCUMENTS

3.1 Introduction

Mathematical expressions typically appear in documents, either as (a) displayed (isolated) expressions or (b) expressions embedded into (i.e. mixed with) the text lines. Figure 2.1 shows a typical document containing both embedded and displayed expressions. Therefore, the first step in recognition of printed expressions is to identify where expressions are located on document image. Moreover, since the presence of expressions disturbs an existing OCR system not yet trained for expression recognition, identification and extraction of expressions will allow an existing OCR engine to process the normal text portion as usual, whereas the extracted expressions can be processed by an OCR specially designed for expression recognition. Such a framework may result in an efficient conversion process for scientific paper documents.

As far as automatic extraction is concerned, displayed and embedded expressions impose different level of complexities. This is so because the displayed ones are typed in separate lines and exhibit several image-level features that distinguish them from normal text lines. On the contrary, embedded expressions are generally small expression fragments which are difficult to isolate from the text portion with which expressions are mixed.

Review of the existing studies reveals that most of the work assume that expressions are available in isolated form and only a few of them deal with expression extraction. Summary of these extraction methods have been presented under the section 1.1 of Chapter 1.

In our approach, separate techniques are employed for identification and extraction of displayed and embedded expressions. A framework based on multifactorial analysis is presented, which integrates several factors to successfully pinpoint the expressions. The module for detection of displayed expressions initially considers several image level features to suspect text lines containing isolated expressions. Next, presence of one or more frequently occurring mathematical symbols is checked to validate the identification results. On the other hand, approach for finding embedded expression initially invokes an existing OCR to recognize the input document. Several features including word N-grams are computed on sentence level to spot sentences containing expressions (this is possible because a statistical analysis of a corpus of scientific documents reveals that the word level N-gram profile for sentences containing embedded expressions is quite different from that of the sentences without any expression). Expression zones are pin-pointed by exploiting inability of OCR to handle expressions and by using some common typographical aspects followed in typing mathematical expressions. Experimental results on a considerable size of dataset show high efficiency of the proposed technique.

The rest of this chapter is organized as follows. Section 3.2 presents a general discussion on multifactorial analysis that has been used to formulate our proposed approach for extraction of expressions. Section 3.3 presents the techniques for locating embedded as well as displayed expressions in a document. Experimental results are outlined in section 3.4 which also presents a measure of efficiency for extraction of expressions and a comparative study to show the distinctiveness of our proposed approach against the existing ones. Section 3.5 summarizes the chapter.

3.2 Multifactorial Analysis

In 1982, Wang and Sugeno [105] first defined the concept of factor spaces and applied it to the study of artificial intelligence [106]. In factor space theory, the word *factor* is the primary term that is denoted by a noun. It has properties like *state* denoted by a numeral and *characteristic* by an adjective. The factor *weight*, for instance, is a noun; 120lbs, 150lbs, etc are its state; heavy, light, etc. are its characteristics.

Li and Yen [72] have discussed four types of factors namely: (i) *Measurable Factors* - factors like time, height, weight, etc. which are measurable and for which the state space (i.e the range of values) can be represented as a discrete subset of an interval. For example, state space for the factor *weight* can be represented by subset $(1, 2, \cdots, 300)$. (ii) *Nominal Factors* - factors that are qualitative in nature and whose state space are sets of terms. For instance, the state space of the factor *religion* is given by (Hinduism, Christianity, ..., Islam). (iii) *Degree Factors* - factors for which the state spaces are usually the interval [0, 1]. Degree of similarity, feasibility, portability, etc. are the examples under this category. (iv) *Switch Factors* - factors for which only two values *yes*, *no* are possible. Because of their boolean characteristics, we call class (iv) as boolean factors. In our problem, we employ class (iii) factors and in this connection, a brief discussion on fuzzy multifactorial analysis is given below.

Consider a decision-making problem where we try to find an optimal solution or solution-set against some objectives. At first, objects are identified based on the objectives to be satisfied. Objects are related to a number of factors and the identification of an object depends of the states of these factors. For instance, an employee in an office has his/her own data: name Mr. X, age 30 years, sex male, and department is accounts, etc. So, an employee is being determined by specifying the states of each factor relevant to him/her. By specifying a number of factors Mr. X can be uniquely identified and described by them. Hence, an object like a point in Cartesian space can be viewed as residing in a space constructed by the factors.

However, an object may not have significance to an arbitrary factor. For example, it is meaningless to discuss factors such as *time* or *mass* for identification of mathematical expressions in documents. So a factor f is called relevant to an object o only if f has significant importance to o. By specifying the factors relevant to the objects identified under the problem specification, our goal is to uniquely determine the object(s) that gives the optimal solution.

After the objects and the factors relevant to them are identified, we go for evaluation of the factors against each object. Suppose n-mutually independent factors $f_1, f_2, \cdots, f_n$ are identified as relevant to an object o and say, m such objects $o_1, o_2, \cdots, o_m$ are located under the problem to be solved. Let $f = f_1, f_2, \cdots, f_n$ be the set of n-factors. Then for each of the m objects o_i; $(i = 1, 2, \cdots, m)$ we get n-different values (or states) corresponding to the n-factors and in total, m-different evaluations are obtained one per object. Let $E = e_1, e_2, \cdots, e_m$ be the set of m-evaluations for m objects and for each e_i there are n-values $v_{i1}, v_{i2}, \cdots, v_{in}$ one for each of the n-different factors. We represent these values in a matrix called evaluation matrix V,

$$V = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1m} \\ v_{21} & v_{22} & \cdots & v_{2m} \\ \vdots & \vdots & \cdots & \vdots \\ v_{n1} & v_{n2} & \cdots & v_{nm} \end{bmatrix} \tag{3.1}$$

where n is the number of factors, m is the number of evaluations, and each column in V represents one evaluation.

Now from this matrix it is very difficult to choose any evaluation as the solution because each of them consists of n-different values for n-different factors. So, we design a function M_n that maps a n-dimensional vector $f = (f_1, f_2, \cdots, f_n)$ into a one-dimensional scalar. Mathematically, it is represented as: $M_n(f) = M_n(f_1, f_2, \cdots, f_n)$. Since M_n is a function of factors, it is called a multifactorial function.

Here we are interested in class (iii) factors for which the state space of the factor

$(f_i)_{i=1}^{n}$ is represented by closed unit intervals [0, 1]. We also restrict the design of M_n so that the synthesized value should never be greater than the largest (represented by the operator, $\bigvee$) of the component values and should never be less than the smallest (represented by the operator, $\bigwedge$) of the component values, i.e.

$$\bigwedge_{i=1}^{m} \leq M_n(f_1, f_2, \cdots, f_n) \leq \bigvee_{i=1}^{n} f_i \tag{3.2}$$

In the literature, function like M_n that satisfies the above condition in equation 3.2 is called *Additive Standard Multifactorial* (ASM) functions [73]. Next, by applying such an ASM function M_n on V we make a multifactorial evaluation as follows:

$$V' = (v_1, v_2, \cdots, v_m) \tag{3.3}$$

where $v_i = M_n(v_{1i}, v_{2i}, \cdots, v_{ni})$; for $i = 1$ to m.

As we deal with degree factors, all the v_{ij}'s are in [0, 1]. Since M_n is an ASM function, the values of all v_i's for $(i = 1, 2, \cdots, m)$ are in [0, 1]. Next, a simple search operation is performed to select v_k which is the maximum over all v_i's, i.e. $v_k = max(v_1, v_2, \cdots, v_n)$ to indicate that the evaluation corresponding to v_k (i.e. the k-th column in matrix v) should be adopted as the optimal solution. However, in our problem, use of v_is is slightly modified to make the approach fit to our application, as explained in the next section.

3.3 Extraction of Expression Zones

The multifactorial analysis described above is used to extract expression zones contained in a scientific document. However, depending on the nature of the factors that are involved in the analysis, the extraction techniques are different for displayed and embedded expressions, as described below.

3.3.1 Displayed Expressions

Displayed expressions appear as isolated text lines and they exhibit some image level features that distinguish them from a normal text line. For instance, the statistical investigation on the corpus of 400 pages, as described in Chapter 2 reveals that the mean value of the white spacing between two normal text lines is about 0.4 times the text height whereas the mean value of the white spacing above and below the displayed expressions is nearly 1.8 times the text height. By text height we mean the average height of the text lines of a document. In general, normal text of point-size 10 and 12 are found in the technical documents. However, so far as text height is concerned displayed expressions are often taller (along vertical direction) than any normal text line.

Another important characteristic of displayed expressions is attributed to the spatial arrangement of the constituent symbols. The y-coordinates of the lower-most black pixels of expression symbols are generally scattered over the expression zone, whereas such pixels of the symbols of a normal text line predominantly lie on one or two straight lines. So, if the standard deviation (σ_y) among the y-coordinates of the lower-most black pixels of the expression symbols is calculated, a value much larger than the similar SD-value calculated for a normal text line is obtained.

Based on these observations, the following four factors (f_{ws}, f_{ds}, f_{mh} and f_{mo}) are defined and integrated through multifactorial analysis for the extraction of displayed expressions. The factor, f_{ws} captures the feature related to the white space surrounding (above and below) a text line and is measured as

$$f_{ws} = 1 - e^{(-\frac{r}{r_\mu})} \tag{3.4}$$

where r denotes the average of the white space (measured in number pixel rows) above and below a text line and r_μ denotes the mean of the white space between two consecutive text lines. In case of the first line, only the line below it is considered to measure r (similarly, for the last text line its preceding line is considered).

The second factor (f_{ms}) is designed to measure the scatteredness of the constituent symbols in a text line and is defined as

$$f_{ms} = 1 - e^{-\sigma_y} \tag{3.5}$$

where σ_y denotes the standard deviation among the y-coordinates of the lower-most pixels of the symbols (i.e. connected components) of a text line.

The factor, f_{mh} measures the height (h) of a text line in terms of pixel rows and compares it to the mean (h_μ) of all h-values. The value of f_{mh} is computed as follows:

$$f_{mh} = 1 - e^{(-\frac{h}{h_\mu})} \tag{3.6}$$

The fourth factor (f_{mo}) keeps track of the occurrence of a few mathematical operator symbols that often appear in expressions. Only 13 operators listed in Table 2.5 are considered for computation of f_{mo}. If k_1 denotes the number of operators identified in a text line and p_i denotes the probability of occurrence of the i-th symbol then f_{mo} is defined as

$$f_{mo} = 1 - e^{-k_1 \sum_{i=1}^{K} p_i} \tag{3.7}$$

Next, these factors involve in a multifactorial analysis as follows. Next, let m_1 be the total number of text lines in any document image. For all the m_1-lines the above four factors (f_{ws}, f_{ms}, f_{mh}, and f_{mo}) are evaluated and a $4 \times m_1$ one-factor evaluation matrix is formed as follows:

$$V_s = \begin{bmatrix} f_{ws}^1 & f_{ws}^2 & \cdots & f_{ws}^{m_1} \\ f_{ms}^1 & f_{ms}^2 & \cdots & f_{ms}^{m_1} \\ f_{mh}^1 & f_{mh}^2 & \cdots & f_{mh}^{m_1} \\ f_{mo}^1 & f_{mo}^2 & \cdots & f_{mo}^{m_1} \end{bmatrix} \tag{3.8}$$

Each column in matrix V_s represents each text line in the image and consists of a 4-dimensional vector that reflects four different aspects for that line to be a displayed expression. Next, these aspects get combined and mapped into a one-dimensional scalar by an ASM-function M_s. The function M_s transforms the $4 \times m_1$ matrix V_s into a $1 \times m_1$ matrix V_s' as follows:

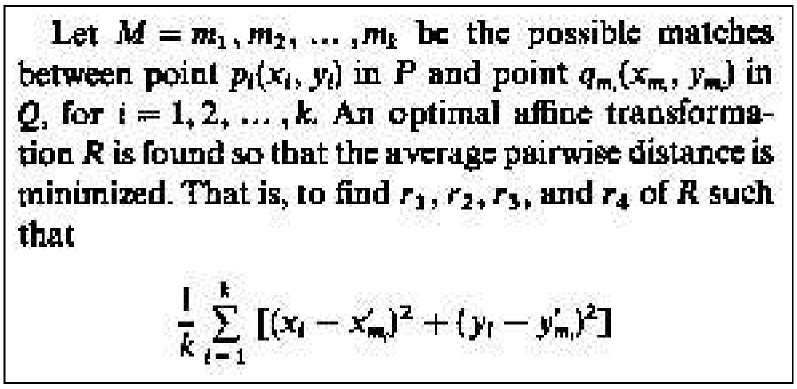
$$V_s' = (f_s^1, f_s^2, \cdots, f_s^{m_1}) \tag{3.9}$$

where $f_s^i = M_s(f_{ws}^i, f_{ms}^i, f_{mh}^i, f_{mo}^i) = \frac{1}{4}(f_{ws}^i + f_{ms}^i + f_{mh}^i + f_{mo}^i)$; i varies from 1 to m_1.

Since the state spaces of all the four factors are theoretically bounded by the interval [0, 1] and M_s retains the property of an ASM-function (discussed in section-3), the state space for f_s is also bounded by the interval [0, 1]. Actually, M_s is used to give each text line a degree of membership, f_s^i that reflects the possibility of i-th line to be a displayed expression. Higher f_s-value indicates larger possibility for that line to be a displayed expression.

Looking at the f_s^i values in V_s', the text lines representing displayed expressions are selected against a threshold (τ_1) determined empirically. The i-th text line is assumed to be a displayed expression if $f_s^i > \tau_1$. In our experiment, it is observed that $\tau_1 = 0.75$ successfully identifies the displayed expressions.

To demonstrate the proposed technique, consider the figure 3.1 which contains seven text lines among which one is a displayed expression. The V_s of equation 3.8 and V_s' of equation 3.9 obtained for the document in figure 3.1 are shown below:



Let $M = m_1, m_2, \ldots, m_k$ be the possible matches between point $p_i(x_i, y_i)$ in P and point $q_{m_i}(x_{m_i}, y_{m_i})$ in Q, for $i = 1, 2, \ldots, k$. An optimal affine transformation R is found so that the average pairwise distance is minimized. That is, to find r_1, r_2, r_3, and r_4 of R such that

$$\frac{1}{k}\sum_{i=1}^{k}[(x_i - x'_{m_i})^2 + (y_i - y'_{m_i})^2]$$

Figure 3.1: Extraction of displayed expressions: an example.

$$V_s = \begin{bmatrix} 0.821 & 0.714 & 0.562 & 0.291 & 0.692 & 0.851 & 0.981 \\ 0.682 & 0.611 & 0.607 & 0.532 & 0.679 & 0.0 & 0.944 \\ 0.538 & 0.569 & 0.541 & 0.598 & 0.539 & 0.505 & 0.886 \\ 0.147 & 0.272 & 0.210 & 0.0 & 0.0 & 0.0 & 0.941 \end{bmatrix} \quad (3.10)$$

$$V'_s = (0.547, 0.542, 0.480, 0.355, 0.478, 0.339, 0.938) \quad (3.11)$$

It is to be noted that the value corresponding to the seventh element of V'_s clearly identifies the displayed expression which is the seventh text line of the document in figure 3.1.

3.3.2 Embedded Expressions

Unlike extraction of displayed expressions, identification of embedded expressions needs extra computational effort. This is so because embedded expressions are mixed with the normal text and often it becomes difficult to locate the expression fragments in a text line that predominantly contains normal text characters. In our approach, design of an extraction technique started studying the corpus (described in Chapter 2) of 400 scanned pages of scientific documents containing 3101 embedded expressions. Two existing commercial OCR systems (that are often used commercially for converting papers into electronic form) are invoked to recognize these pages. Recognition results for sentences with and without expressions are separated for further investigation. The following observations are important in this context.

- Sentences without expressions are recognized with almost no error.

- Also, high recognition accuracy is obtained for normal text words in sentences with expressions.
- Some of the expression symbols (e.g. Roman letters, digits, symbols like '+', '-', '=', punctuation marks, etc.) are often recognized properly. However, for majority of these symbols, the OCRs, on recognition, associate suspicion marks with them to indicate that either these symbols in isolation (excepting characters like 'a', etc.) do not form any valid word (e.g. isolated characters, characters with scripts, words like "sin", "log", etc.) or to reveal poor OCR confidence during their recognition (sometimes due to italic or bold characters).
- Other expression symbols (mostly Greek letters, majority of mathematical operators, special symbols, etc.) are either rejected (signaled by some special symbol) or misrecognized with a suspicion mark.
- If word level N-grams are computed for two categories of sentences (with and without expressions) then such an N-gram based category profiles markedly differ from one another. Section 2.3 of Chapter 2 presents an elaborate discussion in this regard. Let C_E denotes the category of sentences containing embedded expressions.

In our approach, the above observations are captured through defining suitable factors that involve in a multifactorial analysis. For a given document, the proposed method, at first, invokes an existing OCR to recognize the page content in its capability. Next, for each sentence, the word N-grams are computed and their categories are determined against the pre-computed statistics. In fact, instead of categorizing a sentence, a probability (p_i) that the i-th sentence belongs to the category C_E is computed.

Within a sentence, a word is defined as a sequence of characters delimited by a space and let w_{ij} denote the j-th word of i-th sentence. Initially, for all words, two degree factors (f_{mc} and f_{ce}) are evaluated to identify the words that are part of embedded expressions. In true sense, these are not words, rather a sequence of some symbols forming expressions. Identification of these words are further verified by evaluating three more factors (f_{ts}, f_{ms}, and f_{cd}). The five factors (f_{mc}, f_{ce}, f_{ts}, f_{ms}, and f_{cd}) are defined as described below:

The factor, f_{mc} measures the confidence of OCR in recognition of the word, w_{ij} and defined as

$$f_{mc} = 1 - e^{-\frac{c_{off}}{c}} \tag{3.12}$$

where c denotes the confidence of the OCR for recognition of the word, w_{ij} and c_{off} is the offset confidence level based on which the OCR either accepts a recognition result or suspects it to be wrong. The value of c or c_{off} depends on several aspects like average character level similarity measure (if the recognition scheme is based on template matching) obtained for the constituent characters of a word, the presence of the word in the dictionary or lexicon used by an OCR for the post-processing purpose, etc. For function words like '*sin*', '*log*', etc. high character level similarity measures are obtained but still the OCR may suspect their recognition since they do not appear in a common dictionary.

The second factor, f_{ce} measures the inclination of the sentence containing the word (w_{ij}) towards belonging to the category C_E and it is set to p_i as discussed before.

The factor, f_{ts} records the type style of a word. Type style (i.e. regular, italic, and bold) is checked at the character level following the approach presented in [19, 21]. To obtain the binary image of a word, image level bounding box information (i.e. the four corner coordinates) is made available for each word along with its recognition result. Character level type styles determine the value of f_{ts} for a word as follows:

$$f_{ts} = 1 - e^{-k_2} \tag{3.13}$$

where k_2 is the number of characters in the word for which italic or bold styles are detected.

The factor, f_{ms} measures the scatteredness of the constituent symbols in a word and is computed following the equation 3.5. Another factor, f_{cd} measures the inter character distance within the word. Let g denote the average of the inter-character gaps for the word under evaluation and g_μ denotes the same for a normal text word then f_{cd} is computed as

$$f_{cd} = 1 - e^{(-\frac{g}{g_\mu})} \tag{3.14}$$

The value of g_μ is computed considering a few normal text words for which the OCR shows high confidence.

Initially, f_{mc} and f_{ce} are evaluated for each word and next, a multifactorial function M_{id} is used to map the 2-D vector into a scalar quantity (f_{sus}) as follows:

$$f_{sus} = M_{id}(f_{mc}, f_{ce}) = 1/2(f_{mc} + f_{ce})$$

Basically, M_{id} assigns a degree of membership to each word to be *mathematical* i.e. part of an embedded expression. If this degree of membership of a word is above a threshold (τ_2), then the word is deemed as *mathematical*.

The identifications of such *mathematical* words (let m_2 be the number of such words) are further verified for their final acceptance. For this purpose, other three factors (i.e. f_{ts}, f_{ms}, and f_{cd}) are evaluated for each of the m_2 words and a $3\times m_2$ one-factor evaluation matrix is formed as follows:

$$V_w = \begin{bmatrix} f_{ts}^1 & f_{ts}^2 & \cdots & f_{ts}^{m_2} \\ f_{ms}^1 & f_{ms}^2 & \cdots & f_{ms}^{m_2} \\ f_{cd}^1 & f_{cd}^2 & \cdots & f_{cd}^{m_2} \end{bmatrix} \tag{3.15}$$

Next, these factors get combined and mapped into a one-dimensional scalar by an ASM-function M_{em}. The function M_{em} transforms the $3 \times m_2$ matrix V_w into a $1 \times m_2$ matrix V'_w as follows:

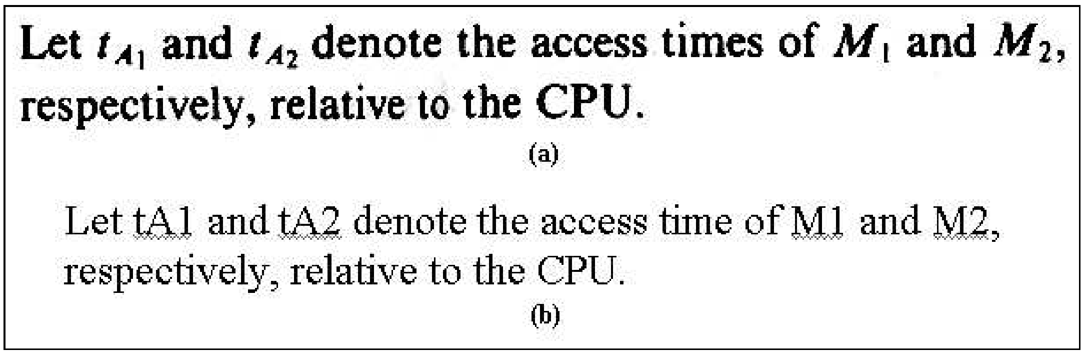
$$V'_w = (f_w^1, f_w^2, \cdots, f_w^{m_2}) \tag{3.16}$$

where $f_w^i = M_{em}(f_{ts}^i, f_{ms}^i, f_{cd}^i) = \frac{1}{3}(f_{ts}^i + f_{ms}^i + f_{cd}^i)$; i varies from 1 to m_2.

Since the state spaces of all the five factors are theoretically bounded by the interval [0, 1] and the function M_{id} and M_{em} retain the property of ASM-functions (discussed in section 3.2), the state space for f_w is also bounded by the interval [0, 1].

Looking at the f_w value, the identification of a word representing embedded expression is accepted against a threshold (τ_3) determined empirically. A word is identified as *mathematical* if $f_{sus} > \tau_2$ and this identification is accepted if $f_w > \tau_3$. The extraction of embedded expressions is partitioned into two stages (i.e. (i) identification of words to be mathematical and (ii) then verification of identification results) to narrow down the space for finding the embedded expressions. A large number of normal text words not participating in any expression are ignored by looking at their f_{sus} values at the first stage. In our experiment, it is observed that the choice of $\tau_2 = 0.5$ and $\tau_3 = 0.7$ successfully identify the embedded expression fragments.

To demonstrate the proposed technique for extraction of embedded expression, consider an example sentence shown in figure 3.2(a). The sentence contains seventeen words, among which four are expression fragments. The OCR output for this sentence is shown in figure 3.2(b). Words for which the OCR shows less confidence are underlined in figure 3.2(a). The sentence shows its inclination towards the category C_E mainly because of the presence of the words *Let* and *denote*. Word-level investigation reveals that confidence scores for recognizing '*tA1*', '*tA2*', '*M1*', and '*M2*' are less than the average confidence for other words and thereby, high f_{sus} values for such words identify them as part of embedded expressions.



Let t_{A_1} and t_{A_2} denote the access times of M_1 and M_2, respectively, relative to the CPU.

(a)

Let tA1 and tA2 denote the access time of M1 and M2, respectively, relative to the CPU.

(b)

Figure 3.2: Extraction of embedded expressions: an example.

Next, the V_w of equation 3.15 and V'_w of equation 3.16 obtained for these four suspected words are shown below:

$$V_w = \begin{bmatrix} 0.863 & 0.863 & 0.630 & 0.630 \\ 0.949 & 0.949 & 0.878 & 0.939 \\ 0.950 & 0.917 & 0.950 & 0.629 \end{bmatrix} \tag{3.17}$$

$$V'_w = (0.921, 0.910, 0.819, 0.733) \tag{3.18}$$

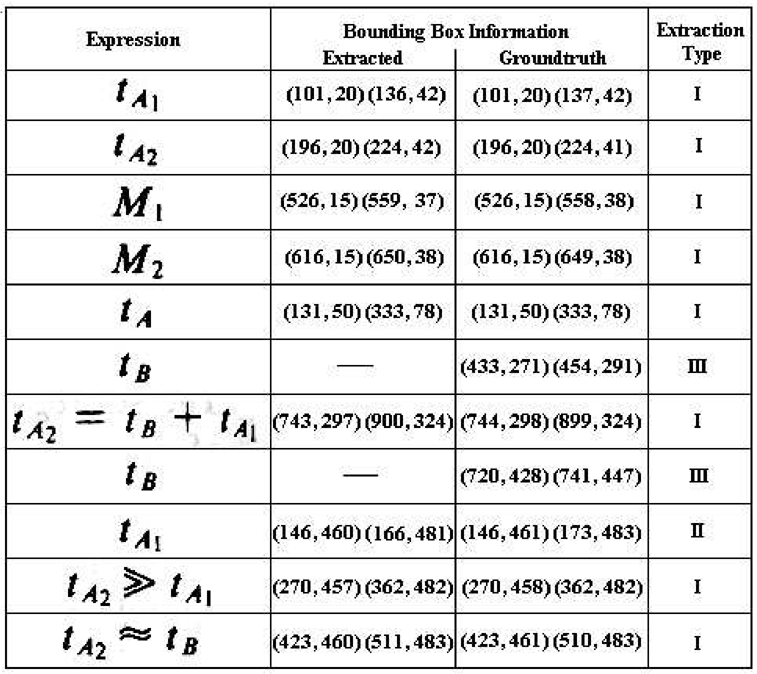
It is to be noted that in V'_w, the values corresponding to all the four suspected words are above the threshold (assuming $\tau_3 = 0.7$) and hence, they are finally accepted as representing expression fragments. In our approach, if more than one word in a sentence are labeled as mathematical then they are grouped together according to their positional proximity to form embedded expressions.

3.4 Experimental Results

The experiment has been carried out on 400 scanned pages available in the corpus described in Chapter 2. These pages contain 2399 displayed expressions and 1084 sentences with 3101 expression fragments embedded in them. A sentence is said to have one or more embedded expressions if it would need the use of math mode had the sentence been prepared using TEX. As explained in Chatper-2 for each page (say, docxxx.tif), displayed and embedded expressions contained in the page are groundtruthed in separate files called docxxx.dis and docxxx.emb, respectively. Along with other information (see Appendix-A and B) each expression is tagged with its bounding box corner (top-left and bottom-right) coordinates.

Extraction of expressions is evaluated against the bounding box information available in the groundtruthed data. One of the following four cases can occur during extraction of expressions: Case-I. An extraction is correct i.e. the bounding box corresponding to the extracted zone finds a match in the groundtruth. Case-II. An extraction is partially correct i.e. the extracted zone shows a bounding box that partially matches the groundtruth. Case-III. An expression is missed i.e. is not extracted at all and Case-IV. False identification i.e. an extracted zone does not actually contain any mathematics at all.

Though the extraction of embedded expression shows all the four cases stated above, extraction of displayed expressions do not exhibit the problem stated under Case-II since for a displayed expression the entire text line is extracted and that line may or may not contain mathematics. In our approach, each line of a multi-line expression is initially extracted as separate displayed expression and later on, positional proximity of the extracted zones connects them together. Figure 3.3 shows the extraction results along with their types (which corresponds to one of the four cases described above) for the embedded expressions contained in the document shown in figure 2.1.



Expression	Bounding Box Information		Extraction Type
	Extracted	Groundtruth	
t_{A_1}	(101,20)(136,42)	(101,20)(137,42)	I
t_{A_2}	(196,20)(224,42)	(196,20)(224,41)	I
M_1	(526,15)(559, 37)	(526,15)(558,38)	I
M_2	(616,15)(650,38)	(616,15)(649,38)	I
t_A	(131,50)(333,78)	(131,50)(333,78)	I
t_B	—	(433,271)(454,291)	III
$t_{A_2} = t_B + t_{A_1}$	(743,297)(900,324)	(744,298)(899,324)	I
t_B	—	(720,428)(741,447)	III
t_{A_1}	(146,460)(166,481)	(146,461)(173,483)	II
$t_{A_2} \gg t_{A_1}$	(270,457)(362,482)	(270,458)(362,482)	I
$t_{A_2} \approx t_B$	(423,460)(511,483)	(423,461)(510,483)	I

Figure 3.3: Extraction of embedded expressions from the document shown in figure 2.1.

3.4.1 Computation of Extraction Efficiency

To evaluate the extraction results, a score (for example, α, β, and γ for case-I, II and III, respectively) is associated with each type of extraction. Let T_1 be the total number of expressions (embedded as well as displayed) properly extracted, T_2 be the number for which partial extraction is done, T_3 be the number of expressions, which are missed (i.e. not extracted) and T_4 be the number of zones that do not contain any expression (false identification). The extraction efficiency (E) for an input page is computed as

$$E = \frac{\alpha T_1 + \sum_{i=1}^{T_2} \beta_i + \gamma T_3}{\alpha T} - \left(1 - e^{-\delta \frac{T_4}{T}}\right) \tag{3.19}$$

where, $T = T_1 + T_2 + T_3$. In our evaluation strategy, α and γ are set to 1 and 0, respectively and β_i is computed as

$$\beta_i = 1 - \frac{C_e - C_a}{C_a} \tag{3.20}$$

where C_e is the number of components found within the i-th extracted zone and C_a is the number of components actually present in the zone. The value of C_a is obtained from the groundtruth where MathML presentation tags are available for each expression.

The last term in the equation 3.19 i.e. $\left(1 - e^{-\delta \frac{T_4}{T}}\right)$ induces penalty due to false identification. Here, the number of false identifications (i.e. T_4) is compared to the total number of actual zones containing expressions (i.e. T). Note that the lower bound of the exponential function used to impose penalty due to false identification is 0, whereas the upper bound (always ≤ 1) is controlled by the parameter δ. In our system, δ is set to 1, however, an empirically chosen value for δ may penalize false identifications is a more judicious manner.

If a document (D_i) shows an extraction accuracy of E_i, then an average efficiency E_{av} is computed in a dataset of N documents as follows:

$$E_{\text{av}} = \frac{1}{N} \sum_{i=1}^{N} E_i \tag{3.21}$$

Following equation 3.19, extraction efficiency for each of the 400 pages is computed separately. As an example, for the page portion in figure 2.1, the following extraction results are obtained: out of thirteen expressions (two displayed and eleven embedded), ten are properly extracted, two embedded expressions are missed and one embedded expression is partially extracted (results for extraction of embedded expressions are in figure 3.3). For partially extracted expression, the extracted zone contains two out of three actual components giving $\beta_i = \left(1 - \frac{1}{3}\right) = 0.67$. Therefore, the extraction efficiency (E) for this document equals to 0.821 $\left[\frac{1\times10+0.67+0\times2}{1\times13}\right]$ is obtained. The equation 3.21 combines page-level efficiency scores to give an overall measure for a given set of documents. In our experiment, a value 0.971 for E_{av} has been obtained for extracting expressions contained in 400 pages. Details of extraction results are given in Table 3.1.

Table 3.1: Summary of Extraction Results

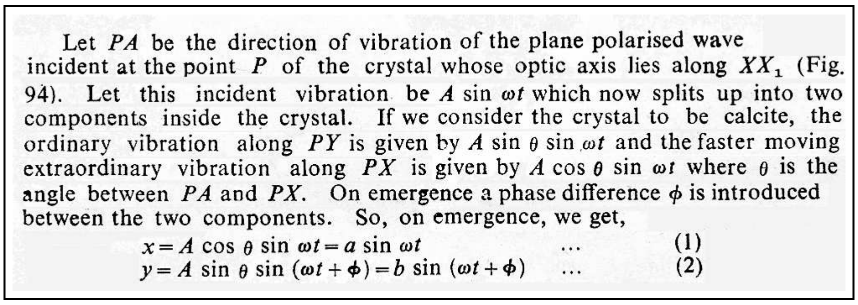
Nature of Extraction	Accuracy		Remarks
	Embedded	Displayed	
Case-I (Perfect)	2904/3101 (93.65%)	2358/2399 (98.29%)	Detection of *Perfect*, *Partial* and Missed is done against the groundtruthed data.
Case-II (Partial)	169/3101 (5.45%)	NIL	
Case-III (Missed)	28/3101 (0.9%)	41/2399(1.71%)	
Case-IV (False)	62	28	In this case, some text portions are wrongly extracted as expressions.

Error Analysis

Analysis of extraction results shows that the errors occurring for extraction of displayed expressions are mostly (1.71%) due to false identification (i.e. Case-IV as stated above). The chapter or section title, table or figure caption, etc. are generally separated by large white space giving high value of the factor, f_{ws}. Sometimes, false recognition of a few letters as mathematical operator (e.g. 'C' as '(', *hyphen* as *minus*, etc.) increases the value of the factor, f_{mo}. Identification of displayed expression is missed in rare occasions where the expressions do not exhibit the features used for their extraction. The expressions that consist of only a few symbols and are densely typed with a line spacing similar (or less than) to that between successive text lines create some problem during extraction. Figure 3.4a shows such an example where the proposed technique fails to identify the displayed expression.

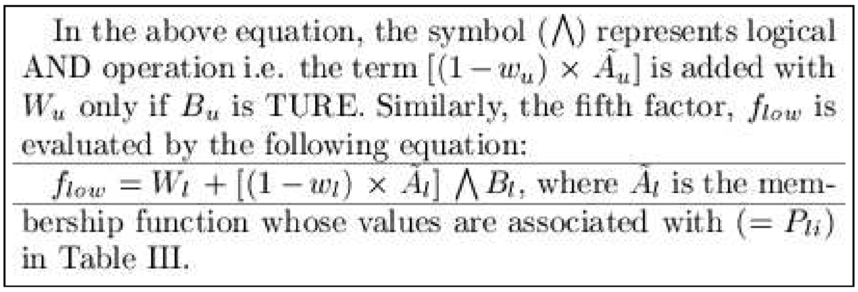
In a few cases, text lines consisting of embedded expressions are identified as displayed expressions. Figure 3.4b demonstrates one such case. However, in these cases text line contains more mathematics than normal text and therefore, as far as OCR of the entire document is concerned, identification of the mathematics intensive text lines as displayed expressions is not a severe error. This is because normal OCRs can hardly recognize such lines and special module trained for expression recognition are anyway needed.

In case of finding embedded expressions, major extraction errors fall under Case-II or III, as stated before. These errors occur for some of the short expression fragments


Let PA be the direction of vibration of the plane polarised wave incident at the point P of the crystal whose optic axis lies along XX_1 (Fig. 94). Let this incident vibration be $A \sin \omega t$ which now splits up into two components inside the crystal. If we consider the crystal to be calcite, the ordinary vibration along PY is given by $A \sin \theta \sin \omega t$ and the faster moving extraordinary vibration along PX is given by $A \cos \theta \sin \omega t$ where θ is the angle between PA and PX. On emergence a phase difference ϕ is introduced between the two components. So, on emergence, we get,

$$x = A \cos \theta \sin \omega t = a \sin \omega t \quad \ldots \quad (1)$$
$$y = A \sin \theta \sin (\omega t + \phi) = b \sin (\omega t + \phi) \quad \ldots \quad (2)$$

(a)


In the above equation, the symbol ($\bigwedge$) represents logical AND operation i.e. the term $[(1 - w_u) \times \tilde{A}_u]$ is added with W_u only if B_u is TURE. Similarly, the fifth factor, f_{low} is evaluated by the following equation:

$f_{low} = W_l + [(1 - w_l) \times \tilde{A}_l] \bigwedge B_l$, where $\tilde{A}_l$ is the membership function whose values are associated with $(= P_{li})$ in Table III.

(b)

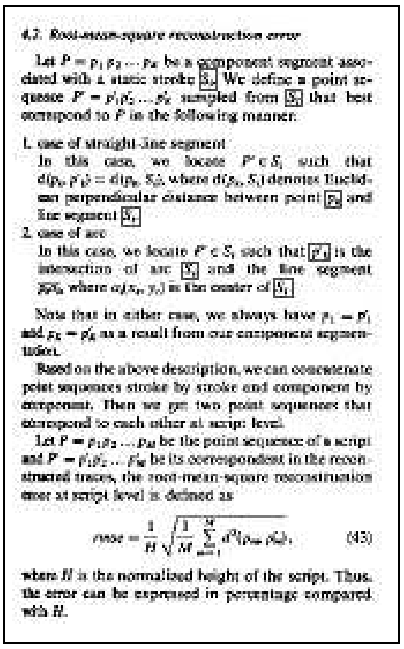
Figure 3.4: Error in extracting displayed expressions: two examples (a) Identifications of the displayed expressions occurring on the last two lines are missed, (b) A text line containing embedded expression is detected as displayed expression.

containing only 2 or 3 symbols. On several occasions, it is difficult to distinguish them from a normal text word. Figure 3.5b shows occurrence of such errors when extraction of expressions is attempted on the document in figure 3.5a. False identifications (i.e. a text portion is wrongly identified as embedded expression) are encountered for documents with excessive degradation due to aging, improper digitization, etc. where large number of broken and merged characters appear and disturb the extraction process.

The performance measure presented in equation 3.19 also considers false extraction along with the accepted and rejected extractions. In our experiment, analysis of test results shows that 62 and 28 zones fall under this false extraction category for extraction of 3101 embedded and 2399 displayed expressions, respectively. These false identifications induce a penalty of 0.016 (negative) in the overall performance measure.

3.4.2 Comparison with the Previous Studies

Though most of the previous studies dealing with expression recognition assume that the expressions are available in isolated form, a few address the problem of finding expressions in scientific documents. Section 1.1 of Chapter 1 presents a brief survey of the



4.2. Root-mean-square reconstruction error

Let $P = p_1 p_2 \cdots p_K$ be a component segment associated with a static stroke $\boxed{S_i}$. We define a point sequence $P' = p'_1 p'_2 \dots p'_K$ sampled from $\boxed{S_i}$ that best correspond to P in the following manner.

1. case of straight-line segment
   In this case, we locate $P' \in S_i$ such that $d(p_k, p'_k) = d(p_k, S_i)$, where $d(p_k, S_i)$ denotes Euclidean perpendicular distance between point $\boxed{p_k}$ and line segment $\boxed{S_i}$.
2. case of arc
   In this case, we locate $P' \in S_i$ such that $\boxed{p'_k}$ is the intersection of arc $\boxed{S_i}$ and the line segment $\overline{p_k o_i}$, where $o_i(x_i, y_i)$ is the center of $\boxed{S_i}$.

Note that in either case, we always have $p_1 = p'_1$ and $p_K = p'_K$ as a result from our component segmentation.

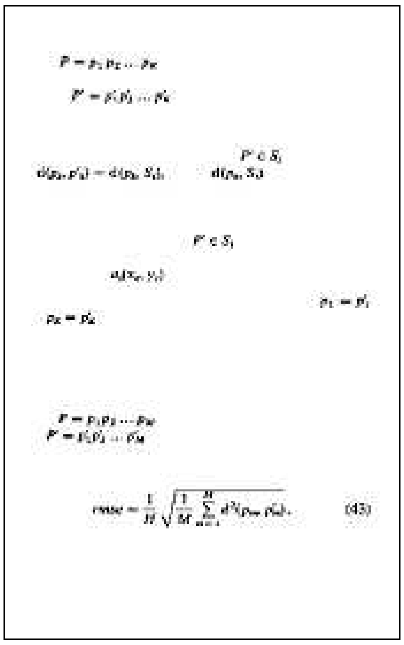
Based on the above description, we can concatenate point sequences stroke by stroke and component by component. Then we get two point sequences that correspond to each other at script level.

Let $P = p_1 p_2 \dots p_M$ be the point sequence of a script and $P' = p'_1 p'_2 \dots p'_M$ be its correspondent in the reconstructed traces, the root-mean-square reconstruction error at script level is defined as

$$rmse = \frac{1}{H}\sqrt{\frac{1}{M}\sum_{m=1}^{M} d^2(p_m, p'_m)}, \qquad (43)$$

where H is the normalized height of the script. Thus, the error can be expressed in percentage compared with H.

(a)



$P = p_1 p_2 \cdots p_K$

$P' = p'_1 p'_2 \cdots p'_K$

$P' \in S_i$

$d(p_k, p'_k) = d(p_k, S_i)$ $d(p_k, S_i)$

$P' \in S_i$

$o_i(x_i, y_i)$

$p_1 = p'_1$

$p_K = p'_K$

$P = p_1 p_2 \cdots p_M$

$P' = p'_1 p'_2 \cdots p'_M$

$$rmse = \frac{1}{H}\sqrt{\frac{1}{M}\sum_{m=1}^{M} d^2(p_m, p'_m)}, \qquad (43)$$

(b)

Figure 3.5: Errors in extracting embedded expressions: (a) Input page (b) Extraction results. Extractions of embedded expression fragments marked with rectangular boxes in (a) are missed in (b).

approaches proposed for identification of expressions in printed documents. Any quantitative comparison of these methods is difficult, since each work defines its own dataset to test the extraction technique. Here, we attempt to outline a qualitative comparison of the studies in terms of the approach adopted, nature of the test data and accuracy. Table 3.2 presents such a comparative study.

Looking at Table 3.2 it is noted that some of the studies do not present any experimental detail for their proposed approach. Fateman [33] describes the proposed algorithm in details and demonstrates it with examples. Chowdhury et. al. [24] compute separate accuracies for extraction of displayed and embedded expressions whereas Jin et. al.[55] provide experimental results for identification of displayed expressions and demonstrate extraction of embedded expressions only by an example. On the other hand, the paper by Kacem [57, 58] presents extraction results with some details and also, formulates a measure for evaluating performance of the propose technique. The work by Suzuki *et. al.*[95] involves a large set of expressions for their experiment but does not provide any accuracy measure for finding expressions in printed documents.

Compared to these previous studies our approach has attempted to present experimental results on a dataset having representative samples of various scientific documents.

Extraction of displayed and embedded expressions has been tested separately and the errors are analyzed in details. Also, an automatic performance evaluation technique is used that utilizes the groundtruth format described in Chapter 2. A technique for computing the efficiency of extraction method is presented to measure the overall (displayed as well as embedded) extraction accuracy.

3.5 Summary

A technique for the extraction of mathematical expressions contained in scientific documents is presented in this chapter. Separate approaches are proposed for identification of embedded and displayed expressions. Several aspects are considered to formulate the extraction methods. These aspects are captured in the form of factors that participate in a multifactorial analysis which forms the core of the extraction technique.

The experiment involving a test set of considerable size shows encouraging results. Techniques for extraction of displayed and embedded expressions are evaluated separately. Next, an integrated performance measure is presented to verify the efficiency of the overall extraction process. A comparative study of the previous approaches as well as the one described in this chapter is also presented.

Integration of the proposed approach with one of the existing OCR systems and evaluation of its performance in recognizing scientific documents will be considered in a future work.

Special Ackowledgement: A paper[1] based on the study discussed in this Chapter was presented in *the 17th International Conf. on Pattern Recognition* (ICPR) and we sincerely thank the anonymous reviewers for their valuable comments and suggestions based on which we did incorporate several modifications in the work presented in this Chapter.

[1]U. Garain, B.B. Chaudhuri, and A. Ray Chaudhuri, "Identification of Embedded Mathematical Expressions in Scanned Documents," *17th Int'l Conf. on Pattern Recognition* (ICPR), pp. 384-387, Cambridge, UK, 2004.

Table 3.2: Comparison of the Studies on Extraction of Mathematical Expressions

No.	Authors	Approach	#Test Samples	Accuracy
1.	Lee and Wang [69, 70]	Image level features are used for extraction of displayed exps. OCR results along with the knowledge about expression layout are used for detection of embedded exps.	NA	NA
2.	Fateman [33]	Grouping of characters as text and math based on their recognition and some post-processing.	NA (Approach is demonstrated with an exam.)	NA
3.	Toumit *et. al.* [99, 100]	Level features for identification of displayed expressions and recognition of a few mathematical operators for embedded expressions.	NA	NA
4.	Kacem *et. al.* [57, 58]	Recognition of a few mathematical operators is done. Spatial arrangement of expression symbols is also exploited. A fuzzy logic based approach is used to formulate the extraction process.	100 scientific documents containing 300 formulas.	93% (overall)
5.	Chowdhury *et. al.* [24]	Spatial distribution of connected components is captured at the image level. No character recognition is attempted.	197 scientific document images.	97.69% (displayed) 68.08% (embedded)
6.	Jin *et. al.* [55]	Recognition free approach is adopted. Extraction of expressions is achieved by using image level features. Existence of 2-D structures is checked to detect embedded expressions.	93 pages with 1370 displayed exps. Tested on 1233 exps.	91.65% (displayed)
7.	Suzuki *et. al.* [52, 95]	Two OCRs are used to recognize normal characters and mathematical symbols. The approach used in [58] is used to pin-point the expressions.	476 documents containing 12,493 exps.	NA
8.	Our Approach	Both image level features and character recognition results are used. Limitation of existing OCR systems has been exploited. A multifactorial analysis is used to integrate several aspects.	400 pages containing 3101 embedded and 2399 displayed expressions	98.29% (displayed) 93.65% (embedded) 95.67% (overall) Efficiency: 0.971, following the Eq. 3.21

NA: Not Available, Exps: Expressions, Exam: Example.

CHAPTER 4

RECOGNITION OF PRINTED MATHEMATICAL SYMBOLS

4.1 Introduction

Symbol recognition in mathematical expressions is difficult because there is a large character set (Roman letters, Greek letters, operator symbols, etc.) with a variety of font styles (regular, bold, italic) and a range of font sizes (scripts, limit expressions, etc.). Certain symbols have an enormous range of possible scales (e.g. brackets, parentheses, symbols like $\int$, $\sum$, $\prod$, $\cup$, etc.). Symbols also vary substantially in their shape characteristics. Therefore, recognition of mathematical symbols is considered as an important pattern recognition problem.

Review of existing studies as presented in section 1.1 of Chapter 1 shows that the studies dealing recognition of symbols are a few in number. Most of the published papers have put emphasis on analysis of two-dimensional structures appearing in expressions. In several experiments, an error-free symbol recognition is assumed before formulating any method for symbol arrangement analysis. In controlled research environment, it is possible to bypass the symbol-recognition step and concentrate on structure analysis phase. However, design of a symbol-recognition module is essential to realize a complete expression recognition system. The work presented in this chapter is directed to this end.

As mentioned earlier, the symbols appearing in mathematical expressions are quite large in number and show wide variety in shape, size and style. Symbols like dot, comma, colon, etc. are small in size and they have little shape signature. Symbols like equal to, plus, minus, fraction bar, vertical bar, greater than, less than, brackets, etc. are not much complex in shape and recognition of such strokes is not very difficult. On the other hand, symbols like Roman and Greek letters, etc. involve relatively complex stroke patterns and recognition of these symbols needs some amount of extra effort.

Based on these observations, a multiple classifier system is adopted here for recognition of symbols. A group of four classifiers of different capabilities are arranged hierarchically in two levels. The classifier used at the top level employs stroke-based classification technique to recognize symbols showing high occurrence frequencies. Symbols not recognized at the first level are passed to the second level that employs a combination of three classifiers. The classifiers placed at this level make use of different feature descriptors namely, run-number or crossing counts, density of black pixels and wavelet decomposition. Different combination techniques have been attempted to integrate the second level classifiers to achieve high recognition accuracy. The presence of connected (or touching) symbols may disturb the recognition of symbols. Therefore, a module for segmentation of connected symbols is designed. Several image level features are considered and a multifactorial analysis as discussed in section 3.2 of Chapter 3 is implemented to find appropriate cut positions.

The rest of this chapter is organized as follows. Section 4.2 presents the description of the classifiers and different combination techniques for their fusion. Segmentation of touching characters is discussed in section 4.3. Section 4.4 reports the experimental results and a comparison of the related studies including the one proposed in this chapter. Section 4.5 summarizes the chapter.

4.2 The Classifiers and their Combination

For recognition of symbols, a multiple-classifier concept is adopted by considering wide variations in shape and size of the symbols appearing in different expressions. Some of the symbols like dot, comma, colon, etc. are small and they have little shape characteristics. Symbols like *equal to*, *plus*, *minus*, *fraction bar*, *vertical bar*, *greater than*, *less than*, *brackets*, etc. are not much complex in shape and recognition of such symbols is relatively easy. On the other hand, symbols like Roman and Greek letters involve comparatively complex stroke patterns and recognition of these symbols needs some amount of extra effort.

Based on these observations, a group of four classifiers of different capabilities is used in our system. The classifiers are arranged according to the figure 4.1. Initially, classifier-1 (C_1) is invoked to recognize quite a few operator symbols that are simple in shape but appear quite frequently in the expressions (frequencies for some of these symbols are provided in Table 2.5). Symbols not recognized by the classifier-1 are passed on to the next level where they are classified by each of the three classifiers in isolation and then the results obtained from these classifiers are combined to improve the final recognition results. The features used by individual classifiers are briefly described below.

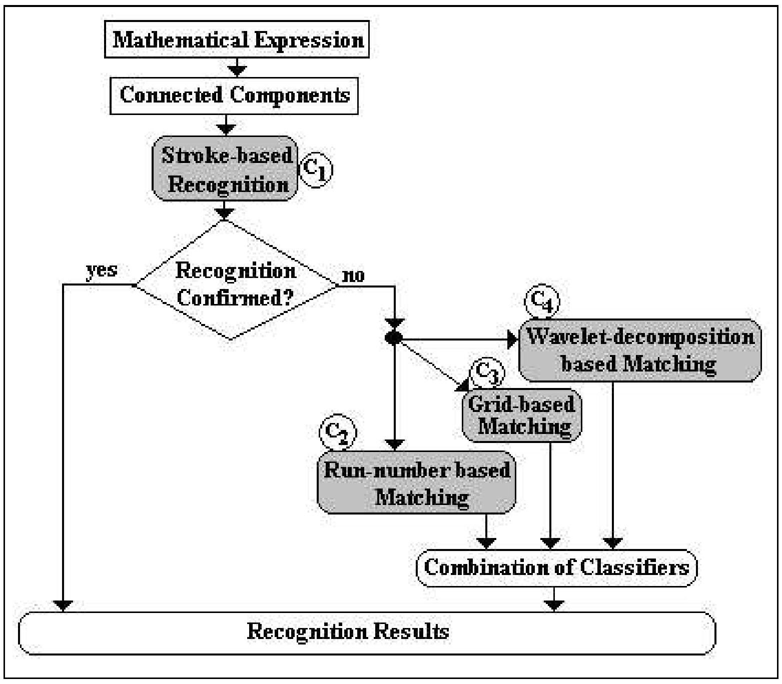
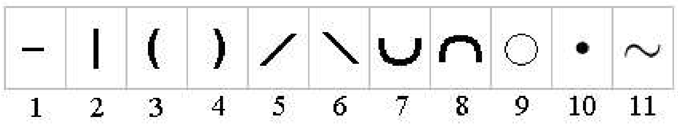


Figure 4.1: Arrangement of the classifiers.

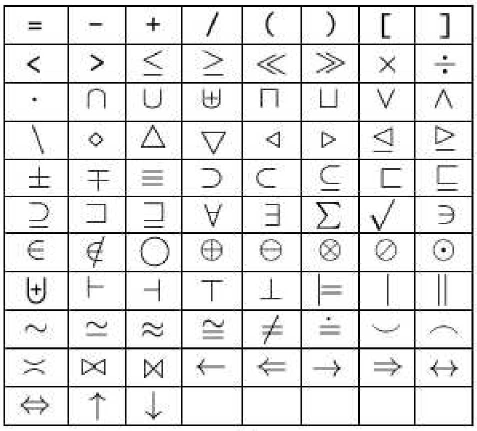
4.2.1 Classifier-1

Classifier-1 looks for the presence of certain strokes to recognize symbols. These strokes are shown in figure 4.2a. Since the strokes are simple in shape, robust detection of these strokes is achieved with reasonable computational effort. Using the stroke features, classifier-1 tries to recognize 83 symbols, as shown in figure 4.2b. It can be noted that classifier-1 mostly recognize symbols which are used as operators in expressions. Some of these operators exhibit high frequency of occurrence in expressions.

For a given symbol, at first, the recognition engine tries to detect one or more number of these 11 strokes as listed in figure 4.2a and then geometric orientation of the identified strokes are checked against a set of rules maintained in the system. The rules used the classifier-1 are of two types: shape description rules and stroke merging rules. The shape description rules are formed to provide unique definition of the symbols recognized by the classifier. For example, the symbol '[' is recognized by identifying one vertical and two horizontal strokes and looking at their relative positions. The rules providing shape description for the symbols take care of the variations that can be found in the definition of several symbols. For instance, the '+' and '-' parts of the symbol, '±' may or may not be connected to each other depending on specific type faces. Therefore, separate description rules are kept in the rule base to tackle such variations.



(a)



=	−	+	/	(	)	[	]
<	>	≤	≥	≪	≫	×	÷
·	∩	∪	⊎	⊓	⊔	∨	∧
\	⋄	△	▽	◁	▷	⊴	⊵
±	∓	≡	⊃	⊂	⊆	⊏	⊑
⊇	⊐	⊒	∀	∃	∑	√	∋
∈	∉	○	⊕	⊖	⊗	⊘	⊙
⊎	⊢	⊣	⊤	⊥	⊨	|	∥
∼	≃	≈	≅	≠	≐	⌣	⌢
≍	⋈	⋈	←	⇐	→	⇒	↔
⇔	↑	↓					

(b)

Figure 4.2: Classifier 1: (a) Stroke features used to recognize symbols shown in (b).

On the other hand, merging rules, whenever possible combines several disconnected strokes into a meaningful symbol. For instance, in case of '=' sign, two disconnected horizontal strokes are initially identified and then '=' is recognized as the combination of these two strokes satisfies the rule corresponding to *equal to* i.e. horizontal strokes of almost equal length are placed one above the other and are separated by a vertical space of height less than the horizontal width of the strokes.

However, rules for merging multiple strokes into symbols consider one important aspect explained as follows. Several symbols recognized by classifier-1 contains another valid symbol as their part. For example, '=' symbols is part of other symbols like '$\equiv$', '$\cong$', '$\doteq$', etc. Therefore, merging rules assume higher priority for merging of maximum possible strokes into a valid symbol to avoid detection of sub-part of symbols. For example, detection of '=' is attempted after the rules corresponding to '$\equiv$', '$\cong$', and '$\doteq$' are tried.

To recognize 83 symbols listed in figure 4.2b, classifier-1 uses 59 shape description and 26 stroke merging rules. Out of these 83 symbols, since many appear frequently in expressions, classifier-1 often recognizes sizeable number of symbols of any expression. The amount of contribution and the recognition accuracy provided by classifier-1 are discussed in detail in section 4.4.

4.2.2 Other Classifiers

The symbols (mostly the Roman letters and digits, Greek symbols, calligraphic letters, some mathematical operators, etc.) that are not recognized by Classifier-1 are passed to a group of three classifiers. Each of these three classifiers computes different features on a target symbol to recognize it. In our system, for a given symbol each classifier ranks the symbol classes, correct obtains the highest rank. Later on, individual classifier's results are combined to improve the rank of the true class in the final classification. Features used by the individual classifiers are summarized below.

- **Classifier-2**: This classifier uses run-number based feature vectors to classify symbols. Consider a symbol (S_1) enclosed by a minimum upright rectangle, called the bounding box. Let (S_1) be scanned horizontally from top to bottom and let N be the total number of scan lines. For each scan line, the number of black to white transitions is counted. This count is known as *run-number* or *crossing count.* Figure 4.3 shows the run numbers for some of the horizontal and vertical scans for the symbol, β. For i-th row let $R_{S_1}[i]$ be the number of black runs. The sequence $\{R_{S_1}[i]; i = 1, 2, \ldots, N\}$ may be considered as a vector of N integer components. We can call it as horizontal run count vector of S_1.

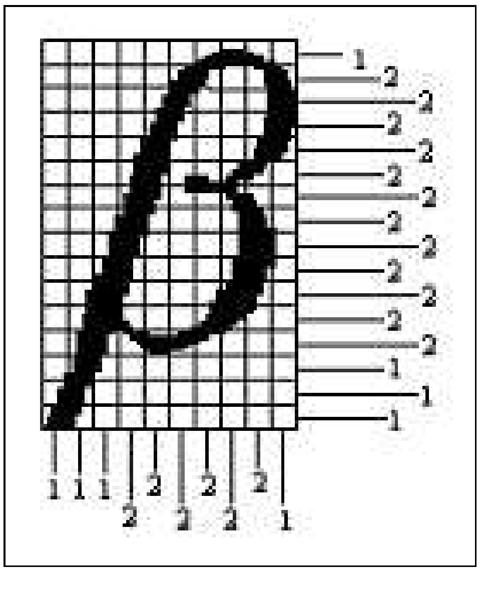


Figure 4.3: Computation of horizontal and vertical run numbers.

The run count vector may be given an abbreviated notation by observing that the run count remains unaltered over a sequence of several scan lines. Each such sequence may be represented as a pair like (m_k, n_k), where m_k denotes the number of scan lines for which the run count is a constant n_k. Note that $\sum_k m_k = N$. To normalize the sum, we can divide the individual m_k's by N. Let $w_k = m_k/N$. Clearly, $\sum w_k = 1$. Hence, the symbol S_1 is represented by a normalized horizontal run count vector defined as

$$V_H(S_1) = \{w_k, n_k; k = 1, 2, \ldots, K\} \text{ where } \sum_{k=1}^{K} w_k = 1 \tag{4.1}$$

Similarly, a vertical run count vector $V_V(S_1)$ can also be computed.

In the classification phase, feature vectors V_H and V_V for a target symbols are computed and matched with the stored prototypes. Matching is done by defining a distance measure explained below:

Let $V_H(S_1)$ and $V_H(S_2)$ be the normalized horizontal run-count vectors of symbols S_1 and S_2, respectively. $V_H(S_1)$ and $V_H(S_2)$ are represented as

$$V_H(S_1) = \{w_k, n_k; k = 1, 2, \ldots, K\} \text{ where } \sum_{k=1}^{K} w_k = 1 \tag{4.2}$$

$$V_H(S_2) = \{w_j, n_j; j = 1, 2, \ldots, J\} \text{ where } \sum_{j=1}^{J} w_j = 1 \tag{4.3}$$

Next, define $W_k(S_1) = \sum_{i=1}^{k} w_i(S_1)$ and $W_j(S_2) = \sum_{i=1}^{j} w_i(S_2)$. Then the union of $\{W_k(S_1); k = 1, 2, \ldots, K\}$ and $\{W_j(S_2); j = 1, 2, \ldots, J\}$ is sorted in increasing sequence. Let this sequence of numbers be $\{W_r; r = 1, 2, \ldots, R\}$ where $W_R = 1$. It is clear that the run-count of symbol S_1 is constant over $W_r - W_{r-1}$ for any $r = 2, \ldots, R$. The same is true for the run count of the symbol S_2.

Now, we can re-define the run-counts of S_1 and S_2 as $w_r n_r(S_1)$ and $w_r n_r(S_2)$, respectively; where $r = 1, 2, \ldots, R$. The distance measure is now formulated as

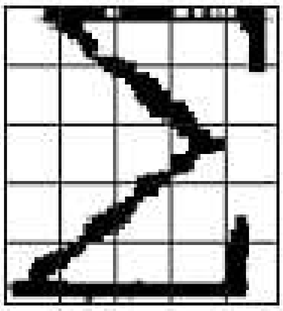
$$J_H(S_1, S_2) = \sum_{r=1}^{R} w_r \left| n_r(S_1) - n_r(S_2) \right| \tag{4.4}$$

It may be understood that J_H satisfies metric property (a formal proof can be found in [39]) since $\sum w_r = 1$. In a similar way, we can scan the characters vertically within the bounding box and find the run-count dissimilarity measure $J_V(S_1, S_2)$. The overall dissimilarity measure may be defined as

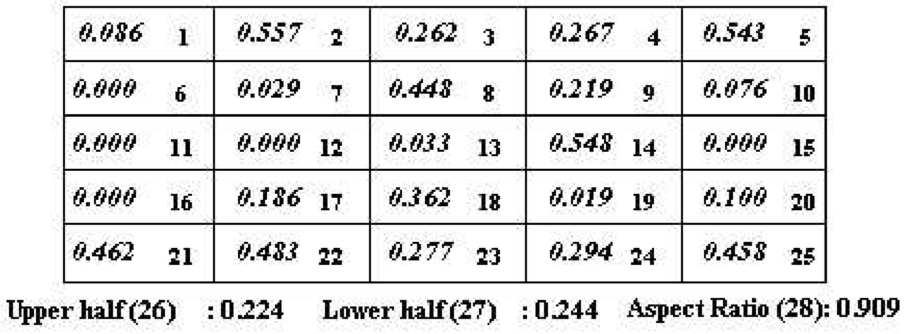
$$J(S_1, S_2) = J_H(S_1, S_2) + J_V(S_1, S_2) \tag{4.5}$$

It can be noted that $J(S_1, S_2)$ is insensitive to bold, expanded and contracted style of symbols. Even it is reasonably stable with respect to binarization error. However, it is observed that inclusion of feature templates for italicized version of symbols in the prototype library improves recognition score.

- **Classifier-3**: Feature vector used by this classifier is computed by dividing a symbols bounding box into 5×5 mesh or grid and then percentage of black pixels in each grid is computed to generate a 25-dimensional vector. Three more data items: first two representing the percentage of black pixels in the top half and the bottom half division and the third one giving the aspect ratio (i.e. width-to-height ratio) are added to obtain 28 dimensional feature vectors. The feature vector used by this classifier is somewhat similar to the one used in [31, 32]. The 28-D feature vector computed for the symbol $\sum$ in figure 4.4a is shown in figure 4.4b.



(a)



0.086 1	0.557 2	0.262 3	0.267 4	0.543 5
0.000 6	0.029 7	0.448 8	0.219 9	0.076 10
0.000 11	0.000 12	0.033 13	0.548 14	0.000 15
0.000 16	0.186 17	0.362 18	0.019 19	0.100 20
0.462 21	0.483 22	0.277 23	0.294 24	0.458 25

Upper half (26) : 0.224 Lower half (27) : 0.244 Aspect Ratio (28): 0.909

(b)

Figure 4.4: Computation of feature vector based on density of black pixels.

An weighted Euclidean distance between two vectors (say, p and q is defined as

$\sum_i w_i \, |p_i - q_i \,|$, where i is the dimension index. In the present system, all w_i are set to 1 but a more judicious choice of w_i will definitely produce better results.

Features used by this classifier is found invariant to font size but sensitive to style variation. Therefore, addition of vectors for italic and bold characters improves recognition performance. However, to improve the speed of the classifiers, characters of identical size are compared. A target vector is compared with a reference vector only if their aspect ratios differ by at most 10% and this reduces the number of comparisons to a large extent.

- **Classifier-4**: This classifier uses wavelet transform for recognition of symbols. Use of wavelet decomposition in symbol recognition task has gained considerable attention in recent past. Since wavelet transform characterizes different physical structures of a character image at different resolution levels, feature extraction based on wavelet decomposition seems to be very effective for symbol recognition task. In our case, the two-dimensional binary image (array) of a symbol is subject to wavelet decomposition by transforming the array on row major order first and then on column major order. Daubechies wavelet [27] is used for this purpose. The simplest member of this family of wavelets is the Daubechies-4 wavelet, which has four coefficients l_0, l_1, l_2, and l_3 that form the low pass (or smoothing) filter (L). The coefficient values used here are as follows:

$$l_0 = \frac{1+\sqrt{3}}{4\sqrt{2}}, l_1 = \frac{3+\sqrt{3}}{4\sqrt{2}}, l_2 = \frac{3-\sqrt{3}}{4\sqrt{2}}, l_3 = \frac{1-\sqrt{3}}{4\sqrt{2}} \tag{4.6}$$

Another set of four coefficients (h_0, h_1, h_2, and h_3) form the high pass filter (H), where $h_0 = l_3$, $h_1 = l_2$, $h_2 = l_1$, and $h_3 = l_0$.

Each symbol image is size normalized to 64×64 maintaining the original aspect ratio. Decomposition is applied twice to obtain 16×16 LL (smooth-smooth) approximation of the original image (actually, decomposition gives 17×17 array for exact reconstruction of the original image). These 17×17 binary images are stored in the prototype library for each reference symbol. For a given symbol ($T = t_{ij}$), the symbols classes ($W^k = \{w_{ij}^k\}$) are ranked according to a distance measure defined as $D(T, W^k) = \sum_i \sum_j XOR\left(w_{ij}^k, t_{ij}\right)$. The class with the lowest D gets the top rank.

It is observed that the wavelet based features are not so much affected in the presence of moderate noise, discontinuity, or small changes in orientation. Figure 4.5a shows the original image of a symbol affected by noise. For this image, the wavelet

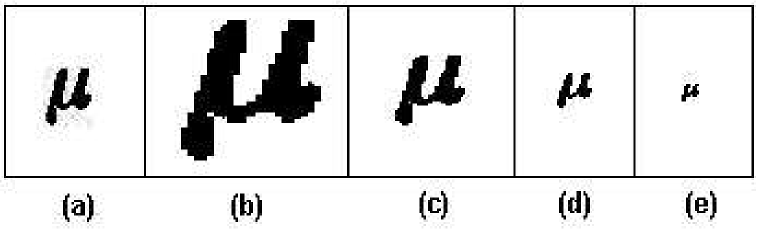


Figure 4.5: Smooth … smooth components of wavelet decomposition of a noisy image at different resolution levels (a) Original noisy image (b) 64 × 64 resolution (c) 32 × 32 resolution (d) 16 × 16 resolution (e) 8 × 8 resolution.

based feature images at different resolution level are shown in figure 4.5(b) - (e). Therefore, the classifier-4 exhibits its insensitivity to different types of degradation including one due to binarization. Moreover, it is invariant to different type styles (e.g. italics, bold, etc.) and font faces.

4.2.3 Fusion of Classifiers

Similarity among the classifiers is, at first, studied before looking for a suitable combination method to integrate the three classifiers (C_2, C_3, and C_4) that constitute the second level of symbol classification. This is done by measuring the agreement among the decisions taken by the classifiers. Let $C = C_i \,|\, i = 1, 2, \ldots, K$ be a set of K classifiers and $P = \{p_t \,|\, t = 1, 2, \ldots, N\}$ be a set of N symbols, each belonging to one of the m possible classes $\{w_1, w_2, \ldots, w_m\}$. For a symbol $p_t \in P$, top choices (highest rank) returned by each classifier C_i are recorded and the similarity measure is given by the index (ρ) discussed in [9]:

$$\rho_C = \frac{\sum_{\substack{i,j=1,\ldots,K \\ i<j}} \rho\{C_i, C_j\}}{{}^{K}C_2} \tag{4.7}$$

where

$$\rho\{C_i, C_j\} = \frac{1}{N}\sum_{1}^{N} Q(C_1(p_t), C_2(p_t)) \tag{4.8}$$

and

$$Q(C_i(p_t), C_j(p_t)) \;=\; 1 \text{ if } C_i(p_t) = C_j(p_t)$$

$$= \quad 0 \text{ if } C_i(p_t) \neq C_j(p_t) \qquad (4.9)$$

It may be noted that the minimum of the similarity index is equal to 0 (when C_i and C_j always disagree to each other) and the maximum is equal to 1 (when C_i and C_j always provide the same response). Similarity measure reveals interdependence among the classifiers. Moreover, such a measure is important to identify whether any classifier is redundant in a group of classifiers.

To combine the classifiers, three approaches based on the highest rank, the Borda count, and Logistic Regression have been tested. Some details of these methods can be found in [49]. Here summary of the methods is given below:

- **Highest Rank Method**: The highest rank method works as follows. Assume there are m classifiers in a system and they rank each class for an input pattern. Therefore, each class receives m-ranks. The minimum (i.e. the highest) of those m ranks is assigned to that class as its score. The classes are then sorted by these scores to derive a combined ranking for that input. Ties are arbitrarily resolved to obtain a strict linear ordering. As the number of classes sharing the same ranks depends on the number of classifiers used, this method is useful when the number of classifiers is small relative to the number of classes. Otherwise, most of the classes are involved in ties and the final ranking becomes less relevant. The problem dealt in this thesis suits well to this framework, since it involves only three classifiers to rank 191 classes.

- **Borda Count Method**: In this method, the *Borda Count* [7]for each class is computed as the sum of the number of classes ranked below it by each classifier. The final ranking is given by arranging the classes so that their Borda counts are in descending order. For each class the Borda count is a measure of the strength of agreement by the classifiers that the input pattern belongs to that class. Though the method assumes additive independence among the individual classifiers, it is simple to implement and requires no training.

- **Logistic Regression**: Unlike Borda count, the Logistic Regression method [26] takes into account the differences in the individual classifier capability. This method works by assuming a binary variable Y (1 for the true class and 0 for other classes) for each class for each input pattern. For a training pattern the true class is known and therefore each class has a known value of Y. For a test pattern let $x = \{x_1, x_2, \ldots, x_m\}$ be the set of rank scores assigned to a class by m classifiers.

Here it is assumed that x_i has the largest value if the class is ranked at the top by the classifier C_i. The regression function $L(x)$ is computed as follows:

$$L(x) = \log \frac{\pi(x)}{1 - \pi(x)} \tag{4.10}$$

and $\pi(x)$ denotes the probability $P(Y = 1|x)$ and is given by

$$\pi(x) = \frac{\exp(\alpha + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_m x_m)}{1 + \exp(\alpha + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_m x_m)} \tag{4.11}$$

where $\alpha, \beta = (\beta_1, \beta_2, \ldots, \beta_m)$ are constant parameters and can be estimated by the techniques based on maximum likelihood or weighted least-squares.

For each test pattern, classes are sorted by their $L(x)$ values and the class with the highest $L(x)$ is then considered as likely to be the true class.

The above combination methods assume that for a given symbol all classifiers return a list of ranks corresponding to the classes. However, in our approach, this is not always assumed. A classifier (C_i) returns a list of ranks only if it finds enough confidence to recognize of a symbol. Therefore, an offset value (τ_i) is used by each classifier (C_i) to judge its confidence for deciding the acceptability of a recognition result. A symbol is *rejected* if two out of the three classifiers do not recognize the symbol.

4.3 Segmentation of Touching Characters

In many occasions, the adjacent characters in expressions touch each other in the scanned image and presence of such touching characters causes recognition errors. This is because the expression is typically segmented by connectivity analysis that considers touching characters as a single unit, which the recognition engine cannot properly tackle.

The main reasons that cause generation of touching characters are mainly attributed to: (i) poor printing technology, (ii) inferior paper quality (iii) photocopied documents, (iv) digitization errors, etc. The statistical analysis conducted on the 400 pages available in our database (discussed in Chapter 2) of mathematical documents shows that out of 82,691 expression symbols 6,144 (7.43%) are touching in nature and generate 2,853 images of touching characters. Expressions contained in documents printed in early years impose a serious problem due to touching characters. Lee and Wang [69, 70] analyzed the reasons behind the error occurring in recognizing expression symbols and found that depending on document's quality 12%-28% errors (out of all types of recognition errors) are due to the presence of touching characters. Suzuki *et. al.* [95] also found about 2% touching character images in their database of 11,194 expressions.

A number of character segmentation techniques have been proposed in the literature. Fujisawa *et. al.* [37], Elliman and Lancaster [29], Casey and Lecolinet [11] have presented elaborate surveys on such methods. Some references contain discussion about segmentation of touching characters as well and from these references we find two categories of approaches where: (i) the touching character segmentation and recognition go hand-in-hand [10, 63, 102], or (ii) the recognition is attempted without segmentation (or segmentation is implicit in nature) [92].

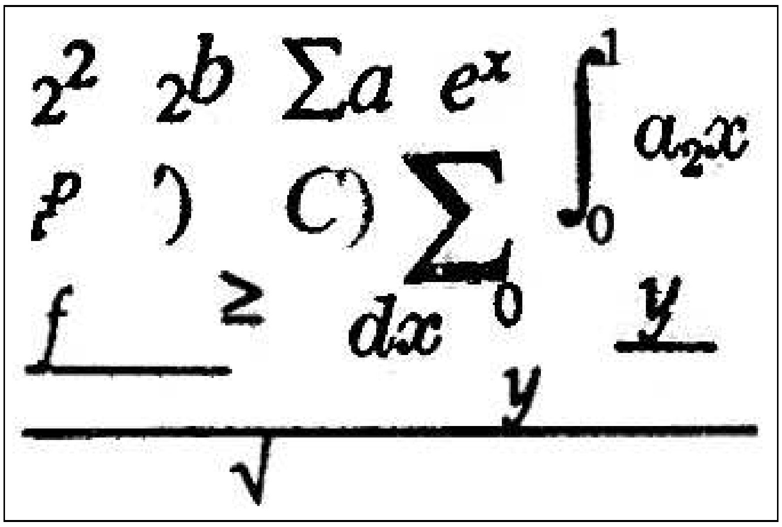


Figure 4.6: Some touching characters found in mathematical expressions.

However, these methods can be applied for character images printed only in one direction; (e.g. horizontal for Roman, vertical for Japanese script). In mathematical expressions, characters are placed in horizontal, vertical or diagonal directions. Some examples of touching characters found in our database of mathematical expressions are shown in figure 4.6. This calls for a new method for segmenting touching characters appearing in expressions.

So far only a few studies have addressed the issue of segmentation and recognition of touching characters appearing in expressions. Lee and Lee [67, 68] proposed a dynamic programming algorithm where the segmentation is performed on a one-dimensional sequence of curve segments representing a connected component. The approach presented by Okamoto et. al. [83] is based on the projection profiles of a given binary image of a pair of touching characters and minimal points of the blurred image obtained by applying the Gaussian kernel to the original image. This segmentation method is restricted for cases where only two characters touch each other.

Very recently, Nomura et. al. [79] proposed an approach for detection as well as segmentation of touching characters in expressions. In their approach, touching characters are detected by looking at the deviation of the feature values (computed on an image of touching characters) from the standard feature values precomputed for isolated characters. The segmentation is achieved by comparing a touching character image with a set of images synthesized from two single character images. Here also touching of only two characters is assumed. Since the number of ways by which two single characters touch each other is quite large, synthesis of all possible touching character images and comparison of a given image with all of these synthesized images is computationally not very much attractive. Moreover, such a scheme may fail to tackle variations in size, style and typefaces used to print expressions.

Considering the limitations of the previous approach we propose a different technique for segmentation of touching characters. Since the number of touching characters is limited in an expression, no separate module for detection of touching characters is implemented in our system. Rather, any pattern rejected by the recognizer is initially suspected as touching character and segmentation is attempted for its recognition. The multifactorial analysis presented in section 3.2 of Chapter 3 forms the core of the proposed segmentation algorithm. The method does not assume anything about the number of characters that may be present in a touching character image.

4.3.1 The Features used for Segmentation

Initially, images of touching characters are investigated and the following observations are noted:

- Observation 1: Though an image of touching characters mostly contains two characters, three or more characters touching each other are not rare. In our database, we found that among 2,853 images of 6,144 touching characters images consisting of 2, 3, and 4 characters are 2501 (87.66%), 266 (9.32%), 86 (3.02%), respectively.
- Observation 2: Adjacent characters may touch each other in horizontal, vertical or diagonal directions.
- Observation 3: If black runs (or crossing counts) are computed along the touching direction, a single run is, in general, encountered at the touching position.

- Observation 4: The thickness of black blob at the touching position is usually small compared to the thickness of other parts.
- Observation 5: The character parts generate uncommon (quite a few in number) stroke patterns above and below the touching points.

Based on these observations, four factors (f_{ic}, f_{mt}, f_{up}, and f_{low}) are defined. Four directions (vertical, horizontal, and two diagonals namely, +45° and −45°) are considered for evaluation of factors. In each direction, factors are evaluated for each object (e.g. each column for vertical direction; similarly, rows are considered as objects for horizontal direction, etc.). Next, the factors are involved in a multifactorial analysis (as explained in section 3.2 of Chapter 3) for finding appropriate cut position in each direction. The four factors used in segmentation are defined as follows:

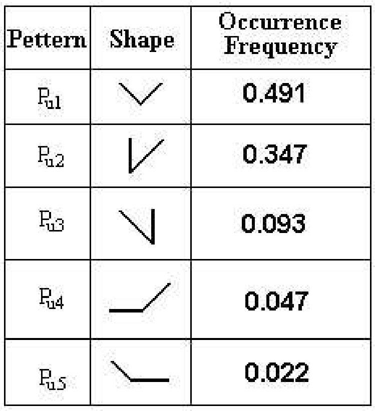
- f_{ic}: inverse crossing-count $= c^{-1}$, where c is the crossing-count (number of white to black transitions) computed for an object (column, row, etc.).

The factor f_{ic} is designed to reflect the property stated in Observation 3. On the other hand, as per our Observation 4 the vertical thickness of the black blob at the touching point is always small compared to the thickness of the other character parts, so we design the second factor (f_{mt}) to take care of this as follows:

- f_{mt}: measure of blob thickness = 1-$e^{-\frac{w_\mu}{t}}$, where t is the number of black pixels encountered in one scan and w_μ is the mean thickness of the character strokes, computed as follows:

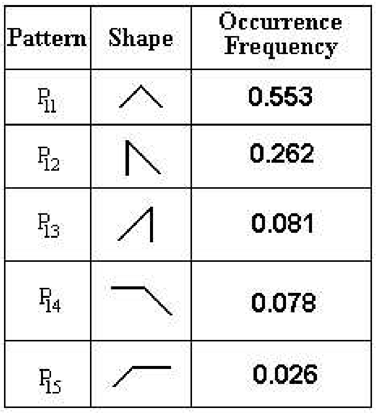
The run-length of black pixels along several directions is recorded at some boundary points. The run-lengths are measured in three directions namely, horizontal, +45° (i.e. upward), and −45° (i.e. downward) direction. Let for a point P the run-lengths in these three directions be $w_h(P)$, $w_u(P)$ and $w_d(P)$, respectively. We take the minimum of these three values as the thickness $w(P)$ at point P i.e. $w(P) = min[w_h(P), w_u(P), w_d(P)]$. At a particular point P, we consider its thickness $w(P)$ provided the thickness in the neighborhood of P along the boundary of the stroke is nearly equals to $w(P)$. In this way, thickness at inconsistent points like corner and serif are avoided. The value of $w(P)$ is noted at several boundary points and a mean is calculated to obtain w_μ.

The other two factors are designed according to the *Observation* 5 stated above. To formulate them a set of touching character images are analyzed and different stroke patterns formed at touching points are noted by using a 5×5 grid whose (3,3) element (i.e.



Pettern	Shape	Occurrence Frequency
P_{u1}		0.491
P_{u2}		0.347
P_{u3}		0.093
P_{u4}		0.047
P_{u5}		0.022

(a)



Pattern	Shape	Occurrence Frequency
P_{l1}		0.553
P_{l2}		0.262
P_{l3}		0.081
P_{l4}		0.078
P_{l5}		0.026

(b)

Figure 4.7: Different shapes formed at touching points, (a) P_{ui}: u-patterns show shapes above a touching point, (b) P_{li}: l-patterns show shapes below a touching point.

the innermost square) is considered as the touching point. Five different stroke patterns are detected above the touching points. We call these patterns as *u*-pattern. Similarly, five different patterns (*l*-pattern) are detected below the touching points. These patterns are shown in figure 4.7.

Let N be the number of samples (touching points). Then the relative frequency of a particular *u*-pattern (P_{ui}) (or *l*-pattern) is computed as number of occurrences of $P_{ui} \div N$. Relative frequency of each u and l-patterns is given in figure 4.7.

In each scan (horizontal, vertical or along diagonals), the presence of any u-pattern and l-pattern are recorded in two boolean variables B_u and B_l, respectively, and based on their values a scan gets two different predefined weights, W_u and W_l for u and l-pattern, respectively. These weights are computed as follows. If B_u is true then $W_u = w_u$ else $W_u = 1 - w_u$ and similarly, if B_l is true then $W_l = w_l$ else $W_l = 1 - w_l$. In our implementation, w_u and w_l both are less than 1 but greater than 0.5.

The third factor, f_{up} is evaluated by the following equation which combines weight W_u and the membership functions $\tilde{A}_u$ (= relative frequency of P_{ui} as given in figure 4.7.

- $f_{up} = W_u + [(1 - w_u) \times \tilde{A}_u] \bigwedge B_u$.

In the above equation, the symbol ($\bigwedge$) represents logical AND operation i.e. the term $[(1 - w_u) \times \tilde{A}_u]$ is added with W_u only if B_u is TURE. Similarly, the fourth factor, f_{low} is evaluated by the following equation:

- $f_{low} = W_l + [(1 - w_l) \times \tilde{A}_l] \bigwedge B_l$, where $\tilde{A}_l$ is the relative frequency of P_{li}, as given in figure 4.7.

4.3.2 Selection of Cut Positions

In a particular direction (vertical, horizontal, and two diagonal directions), let m be the total number of scans in any image of touching characters. For all the m-scans the above four factors (f_{ic}, f_{mt}, f_{up}, and f_{low}) are evaluated and a $4 \times m$ one-factor evaluation matrix is formed. For example, if there are m pixel-columns in the image then the $4 \times m$ evaluation matrix obtained for vertical scans is as follows:

$$V_v = \begin{bmatrix} f_{ic}^1 & f_{ic}^2 & \cdots & f_{ic}^m \\ f_{mt}^1 & f_{mt}^2 & \cdots & f_{mt}^m \\ f_{up}^1 & f_{up}^2 & \cdots & f_{up}^m \\ f_{low}^1 & f_{low}^2 & \cdots & f_{low}^m \end{bmatrix} \tag{4.12}$$

Each column in matrix V_v represents each scan (column wise, row wise, or along diagonals depending on the direction in which the factors are evaluated) in the image and consists of a 4-dimensional vector that reflects four different aspects for that column to be a cut candidate. Next, these four aspects get combined and mapped into a one-dimensional scalar by an ASM-function M_v. The function M_v transforms the $4 \times m$ matrix V_v into a $1 \times m$ matrix V_v' as follows:

$$V_v' = (f_v^1, f_v^2, \cdots, f_v^m) \tag{4.13}$$

where $f_v^i = M_v(f_{ic}^i, f_{mt}^i, f_{up}^i, f_{low}^i) = \frac{1}{4}(f_{ic}^i + f_{mt}^i + f_{up}^i + f_{low}^i)$; i varies from 1 to m.

Since the state spaces of all four factors are theoretically bounded by the interval [0, 1] and M_v retains the property of an ASM-function (discussed in section 3.2), the state space for f_s is also bounded by the interval [0, 1]. Actually, M_v is used to give each column (row for horizontal direction) (i) a degree of membership, f_v^i that reflects the possibility of i-th column to be a cut candidate for separating characters. Higher f_v-value indicates larger possibility for that column to be a cut-column.

Confirmation of Cut Positions

Researchers proposed various approaches [10, 102] for the confirmation of cut positions. In most methods, the character classifier spends a substantial amount of time in attempting to recognize patterns that are not valid. In our approach, we have attempted to reduce the computation by predicting the most favorable cut positions before character classifier is used to confirm it. This concept is borrowed from the predictive parser concept well-known in the field of compiler design for the programming languages [1]. The details of the method for predicting cut positions can be found in [42].

Though the characters may touch in horizontal, vertical, or diagonal directions, a statistical analysis reveals that vertical (i.e. column) cut positions are most common for segmenting touching characters appearing in expressions. This type is followed by cut positions along 45° direction, horizontal (i.e. one or more rows are selected as cut positions) and the +45° direction, respectively.

In our approach, segmentation is also tried in the above sequence. A character image (touching or not) rejected by the recognition engine is subjected to segmentation. Initially, V'_v is computed for the vertical direction i.e. to find appropriate cut columns. If V'_v does not provide any suitable cut column(s), V'_{-45} is computed next for the $-45°$ direction. If V'_{-45} also fails to find appropriate cut position(s), horizontal scans (i.e. on rows) are invoked to generate V'_h. At last, V'_{+45} is constructed to find cut position(s) along $+45°$ direction. If none of these four attempts provides any suitable cut position, the image is considered as non-touching and it remains as a rejected pattern.

Finding of suitable cut positions is explained here. For the vertical direction and the same explanation holds for other directions. In V'_v, the column having the highest f_v value is predicted as the most favorable cut column. The components formed by this cut-column are sent to the classifier and based on its decision, the algorithm goes forward to the next cut-column or tries with the column having the second best f_v value. The algorithm terminates when all segments generated by the selected cut columns pass through the character classifier. A success is achieved if all segments are recognized by the classifier. Otherwise, a failure is reported and segmentation is tried in another direction.

To reduce computation time, for each pattern only top two f_v values that belong to the current pattern are used as two successive predicted cut-columns. Figure 4.8 illustrates the algorithm for a touching triplet found in an expression shown in Figure 4.8a. Figure 4.8b shows the three cut-columns having top three f_v values for the touching character shown in figure 4.8a. Column i has the highest f_v value, so it is predicted as the most favorable cut-column, yielding patterns P1 and P2 in figure 4.8c. The classifier recognizes P1 as a valid character while P2 is rejected (figure 4.8d). P2 is then segmented by the cut-column (column j) having the highest f_v value within P2 (see figure 4.8c). This cut yields P3 and P4 both of which are rejected by the classifier (figure 4.8d). So the algorithm chooses the next cut-column (column k having the second highest f_v value within P2) yielding patterns P5 and P6, both of which are accepted by the character classifier (figure 4.8e). At this stage the algorithm terminates.

In the above example, the segmentation algorithm reports success during its execution in vertical direction. However, there are cases where vertical segmentation fails and

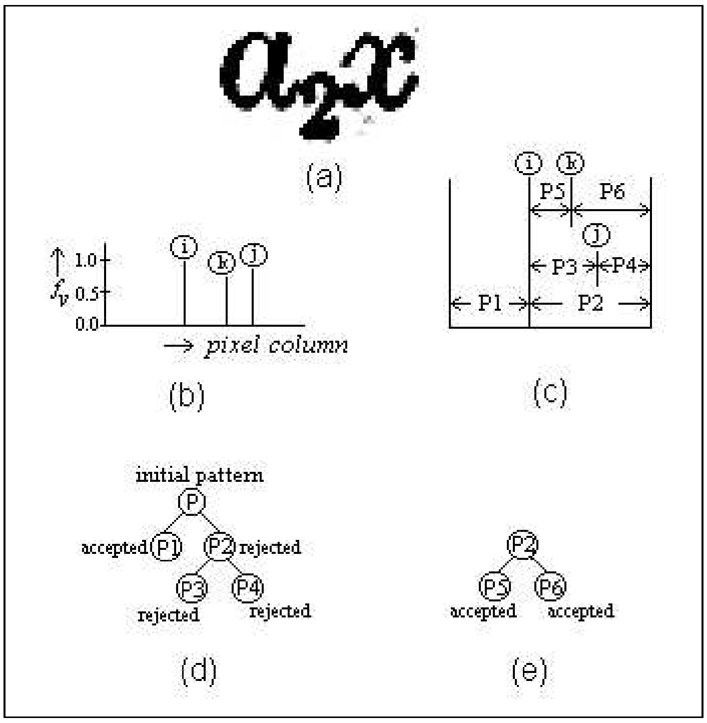


Figure 4.8: Confirmation of cut positions for a triplet.

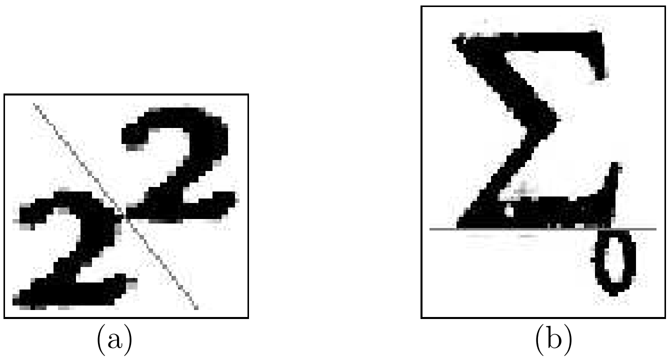


Figure 4.9: Segmentation along (a) diagonal and (b) horizontal direction.

segmentation in other directions are tried subsequently. For example, figure 4.9a does not produce acceptable segmentation results when vertical segmentation is tried. Rather, the segments produced by the cut position along the $-45°$ direction are accepted by the character classifier. Similarly, figure 4.9b shows the accepted cut position along horizontal direction (i.e. row wise).

4.4 Experimental Results

Experiments are carried out on the dataset described in Chapter 3. The database contains 2459 displayed and 3101 embedded expressions. Total number of symbols found in all displayed and embedded expressions is 82,691 (displayed: 59,288 and embedded: 23,403). Actual number of symbols extracted (based on connected component analysis and merging of components based on positional proximity) from these expressions is 79,400 as 6,144 number of characters are touching in nature generating 2,853 character patterns. Among these symbols, 274 distinct symbol classes have been identified among these symbols. Table 2.4 describes the classes and their occurrence statistics.

4.4.1 Performance of Individual Classifiers

As mentioned before, classifier-1 invokes stroke-based recognition scheme to recognize symbols belonging to 83 classes as shown in figure 4.2b. Test results show that out of 79,400 character patterns, 27,846 (almost 35%) symbols pass through classifier-1. Evaluation of recognition results is done using the groundtruthed data available for each expression symbol. Such an evaluation reveals that out of 27,846 symbols classifier-1 correctly recognizes 27,373 symbols giving a recognition accuracy of about 98.3%.

The rest of the symbols (i.e. 79,400 - 27,846 = 51,554) are passed to the second level consisting of a group of three classifiers. Since all of these three classifiers needs training samples for generation of prototype libraries, these 51,554 symbols are divided into two groups (i) training and (ii) test dataset. Training dataset contains 33,511 samples (nearly 65%) that are carefully chosen to represent 191 (274 - 83) symbol classes. Moreover, it is ensured manually that training set does not contain any touching characters. Prototype libraries used by three classifiers are generated in isolation by using the same training set. Individual classifiers are tested on a dataset of 18,043 samples that were not present in the process of prototype library formation. Performance of individual classifiers is reported in Table 4.1. Here, recognition accuracy for classifiers 2, 3, and 4 considers the top choice only.

The patterns rejected by classifier-1 are mainly those symbols that do not belong to the set given in figure 4.2b. The rejection rate provided by the second level classifiers (2, 3, and 4) are also quite high (32%-33%). These rejected patterns mostly represent touching characters which are processed next.

Table 4.1: Evaluation of Individual Classifiers

Classifiers	#Test Samples (T)	#Rejection (R)	Classification Results on (T-R)	
			Correct	Incorrect
Classifier-1	79,400	51,554	27,373 (98.30%)	473
Classifier-2	18,043	6,013	10,599 (88.11%)	1,431
Classifier-3	18,043	5,851	10,459 (85.79%)	1,733
Classifier-4	18,043	6,027	10,878 (90.53%)	1,138

More analysis of the recognition results provided by the second level classifiers reveals that the classifiers are of comparable power but behave differently for different types of symbols. The run-number based classifier (i.e. classifier-2, C_2) shows better performance for complex shaped symbols (e.g. Roman letters, Greek symbols, etc.), but performs poorly for symbols with less structural complexity. The grid-based classifier (i.e. classifier-3, C_3) provides a good overall recognition score but gets confused when one symbol is the mirror image of other. On the other hand, wavelet decomposition based classifier (i.e. classifier-4, C_4) shows consistent confidence for all types of symbols. The classifiers, C_2 and C_4 exhibit their insensitivity to variation in character's style and font. Also, classifier-4 (C_4) is quite robust in the presence of moderate amount of noise.

The similarity among the classifiers (C_2, C_3, and C_4) is estimated following the equation 4.7 and a value 0.731 is obtained for the similarity index, ρ_C. In the present system, the classifiers are trained on the same data set. Therefore, the value of ρ_C may change if classifiers are trained on different training sets.

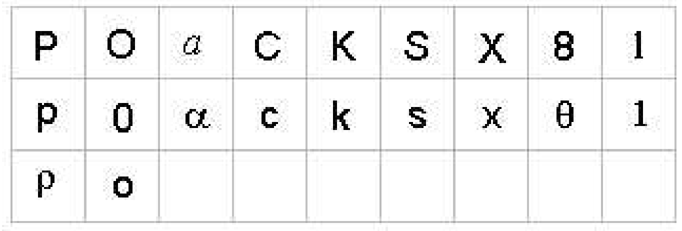
4.4.2 Recognition Results after Combination of Classifiers

Finally, the classifiers are combined following three methods outlined in section 4.2.3. Among these combination techniques, the first two (i.e. the highest rank and the Borda count) do not need any training, whereas the logistic regression needs a training to evaluate its parameters. The details of combination results are shown in Table 4.2. It is to be noted that among the three methods the highest rank method and the logistic regression give comparable performance, the latter being slightly better. On the other hand, the Borda count method does not produce very encouraging results in the present system. This is so because this method does not take into account the differences in the individual classifier capability.

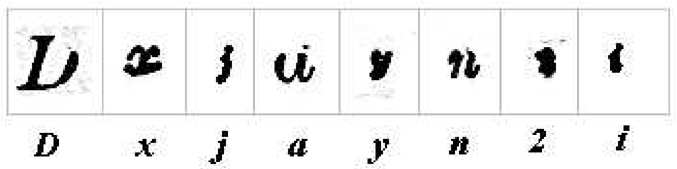
Table 4.2: Recognition Results after Combination of Classifiers

Classifiers and their combination	% Correct in top N choices				
	1	2	3	5	10
Run-number based Classification (C_2)	88.11%	90.31%	93.15%	95.18%	95.65%
Grid based Classification (C_3)	85.79%	88.17%	90.93%	92.16%	92.88%
Wavelet based Classification (C_4)	90.53%	92.65%	94.82%	96.09%	96.87%
Combination by the Highest Rank method	93.26%	95.18%	96.71%	97.83%	98.05%
Combination by the Borda Count	92.03%	94.60%	95.79%	96.45%	97.14%
Combination by Logistic Regression	93.77%	96.73%	97.62%	98.10%	98.38%

Analysis reveals that major sources of classification errors are (i) Substitution errors: Presence of some confusing symbols generates substitution type of recognition error. Figure 4.10(a) shows some expression symbols having similar shapes (symbols in the same column are of confusing nature). (ii) Excessive degradation: Symbol images affected due to aging, digitization error, etc. are often misrecognized. Figure 4.10(b) shows a few of such symbols appearing in test expressions. Actual symbol representing each image is written below the image.



(a)



(b)

Figure 4.10: Classification errors (a) confusing shapes and (b) degraded characters.

4.4.3 Recognition of Touching Characters

As mentioned earlier, characters rejected by at least two of the three classifiers (C_2, C_3, and C_4) are treated as touching characters. Following this, it is observed that 3,156 character patterns are labeled as rejected. Further segmentation is tried on these patterns and it is found that 2,739 patterns (which are truly touching in nature) are recognized by the recognition engine. Analysis shows that these 2,739 patterns generate 5,892 segments which are properly classified by the recognition engine.

Efficiency of the segmentation method proposed for processing touching characters can be computed from the above results. The number of touching images is 2,853. These images consist of 6,144 characters (images containing 2, 3 and 4 characters are 2501, 266 and 86, respectively). Out of 2,853 touching images, 2,739 are recognized, giving an accuracy of about 96% for touching character recognition. On the other hand, out of 6,144 characters generating 2,853 touching patterns, 5,892 characters are recognized through proper segmentation. This improves the overall recognition accuracy by almost 12% (from 86.74% to 98.73%) as explained in Table 4.3.

Table 4.3: Summary of Symbol Recognition Results

Total number of expression symbols	82,691
Number of symbols used to train the 2nd level classifiers	33,511
Number of symbols used in testing	49,180
Number of symbols properly recognized by C_1	27,373
Number of symbols properly recognized by the best combination of C_2, C_3, & C_4	15,288
Number of symbols properly recognized by C_1, C_2, C_3, & C_4	42,661
Accuracy obtained without further processing of touching characters	86.74%
Number of characters recognized after segmentation of touching characters	5,892
Total number of symbols recognized correctly	48,553
Overall recognition accuracy	98.73%

4.4.4 Comparison with the Previous Studies

Comparing performance of the proposed method with those of the previous ones is quite difficult since each uses their own dataset. However, a qualitative comparison is presented in Table 4.4 that shows the distinctiveness of the study presented in this chapter.

Table 4.4: Comparison of the Studies on Recognition of Printed Mathematical Symbols

No.	Authors	Approach	#Symbol classes	#Test Samples	Recognition Accuracy
1.	Okamoto *et. al.* [84]	Template matching and majority voting	NA*	11,565	98.96%
2.	Fateman *et. al.* [31, 32]	Template matching	90	NA*	NA*
3.	Lee and Lee [67, 68]	Feature extraction, nearest neighbor	187	1,033	84.80%
4.	Lee and Wang [69, 70]	Feature extraction, nearest neighbor	190	11,210	96.18%
5.	Ha *et. al.* [47]	Feature extraction and neural network	NA*	NA*	NA*
6.	Suzuki *et. al.* [95]	Feature extraction, three-step coarse-to-fine classification	564	152,143	95.18%
7.	Our Method	Multiple classifier system (4 classifiers)	274	34,676	98.73%

* NA: Not Available

4.5 Summary

This chapter presents a multiple-classifier system for recognition of symbols appearing in printed mathematical expressions. Four classifiers are designed and arranged in a two-level hierarchy. Touching characters are segmented by technique based on the multi-factorial analysis of several factors contributing to identifying appropriate cut positions. Experimental results show high efficiency of the proposed method to recognize a large set of expression symbols. Recognition errors are analyzed to determine their sources. Chapter 7 presents a set of error correcting rules that consider several contextual information to correct some of the errors encountered here. A comparative study is also presented to compare our approach with the previous ones.

Special Acknowledgement: A paper[1] based on the study described in this Chapter was presented in *the 17th International Conf. on Pattern Recognition* (ICPR) and we sincerely thank the anonymous reviewers for their valuable comments and suggestions based on which several modifications were incorporated in the work presented in this Chapter.

[1]U. Garain, B.B. Chaudhuri, and R.P. Ghosh, "A Multiple Classifier System for Recognition of Printed Mathematical Symbols," *17th Int'l Conf. on Pattern Recognition* (ICPR), pp. 380-383, Cambridge, UK, 2004.

CHAPTER 5

INTERPRETATION OF EXPRESSION STRUCTURE

5.1 Introduction

The geometric structure of an expression can be significantly more complex than that of normal text lines. For example, plain text is written linearly from left to right, but mathematical symbols are written above, below, and one inside another. The spatial relationships among symbols are crucial to the interpretation of the expression. This means that even if all the characters are correctly recognized, there still remains the non-trivial problem of interpreting the two-dimensional structure of an expression. Ambiguities arise in areas like (i) *The semantic role of symbols*: Several symbols (e.g. *horizontal line*, *dot*, etc.) have multiple meaning depending on the context; (ii) *Relative symbol proximity and position*: Expression symbols use spatial relationship to indicate the logical relationship among them. For instance, structures like *superscript*, *subscript*, *implied multiplication*, *matrix*, etc. are indicated implicitly by the geometric layout of operands.

This problem has been studied by researchers and various approaches as reviewed in section 1.1 of Chapter 1 are documented. However, these studies reveal that additional research is needed for automatic understanding of mathematical expressions to achieve acceptable accuracy.

In this chapter, a simple grammar-based approach has been presented to recognize complex two-dimensional structure of printed expressions with high accuracy. The proposed technique is based on symbol identities, L-values (`level`[1]), and their positional information. The entire expression image initially is partitioned into different vertical and horizontal stripes based on pixel projection. This partition is done recursively until an atomic stripe is obtained. Each stripe represents a token or lexical group. Next, two or more tokens are combined together to form a sub-expression. Finally, the sub-expressions are merged to form the final expression string. The approach is a bottom-up one and merging of tokens or sub-expressions are guided by (i) the geometric properties derived from the statistical understanding of the corpus and (ii) the productions generated by

[1]The term `level`-value has been defined in section 2.2.2 of Chapter 2.

a formal context free grammar. Space and time complexity analysis for the proposed approach is also presented. The present study differs from the others in a number of ways given below.

- It defines an easily implementable technique for parsing expressions;
- Time and space complexity of the proposed parsing techniques are presented;
- The simple grammar used here is able to successfully process expressions found in various branches of science;
- A new way of defining structural complexity of an expression is proposed.
- The test data set used in the present experiment contains about 5,560 expressions taken from various branches of science. Wide structural variability of these expressions makes the data set a representative one.
- An in-depth evaluation strategy is presented to judge the performance of the proposed technique for structural analysis.

The rest of this chapter is organized as follows. Section 5.2 presents some structural properties of printed expressions. Our technique along with the proposed grammar is described in section 5.3 which also contains a complexity measure of our algorithm. Section 5.4 presents the experimental results and analysis. Section 5.5 summarizes the chapter.

5.2 Some Structural Properties of Expressions

We start with the corpus consisting of 2459 displayed and 3101 embedded expressions taken from several branches of science. Chapter 2 provides details about this dataset. By studying the expressions in corpus, we noted a number of structural properties inherent in these expressions. Some of them, used in designing our reconstruction algorithm, are mentioned below.

- Property 0. *Bounding box of a symbol* : For a symbol s, we define its bounding box $B(s)$ to be the smallest upright rectangle enclosing the symbol. We denote it by a 4–tuple (xl, xr, yt, yb), where (xl,yt) and (xr,yb) represent the top-most left corner and the bottom-most right corners of B, respectively. Figure 5.1(c) and (d) show the bounding boxes for symbols of expressions in (a) and (b), respectively. We define the height(yh) and width(xw) of a symbol by

$$\begin{aligned} yh &= yb - yt + 1 \\ xw &= xr - xl + 1 \end{aligned} \tag{5.1}$$

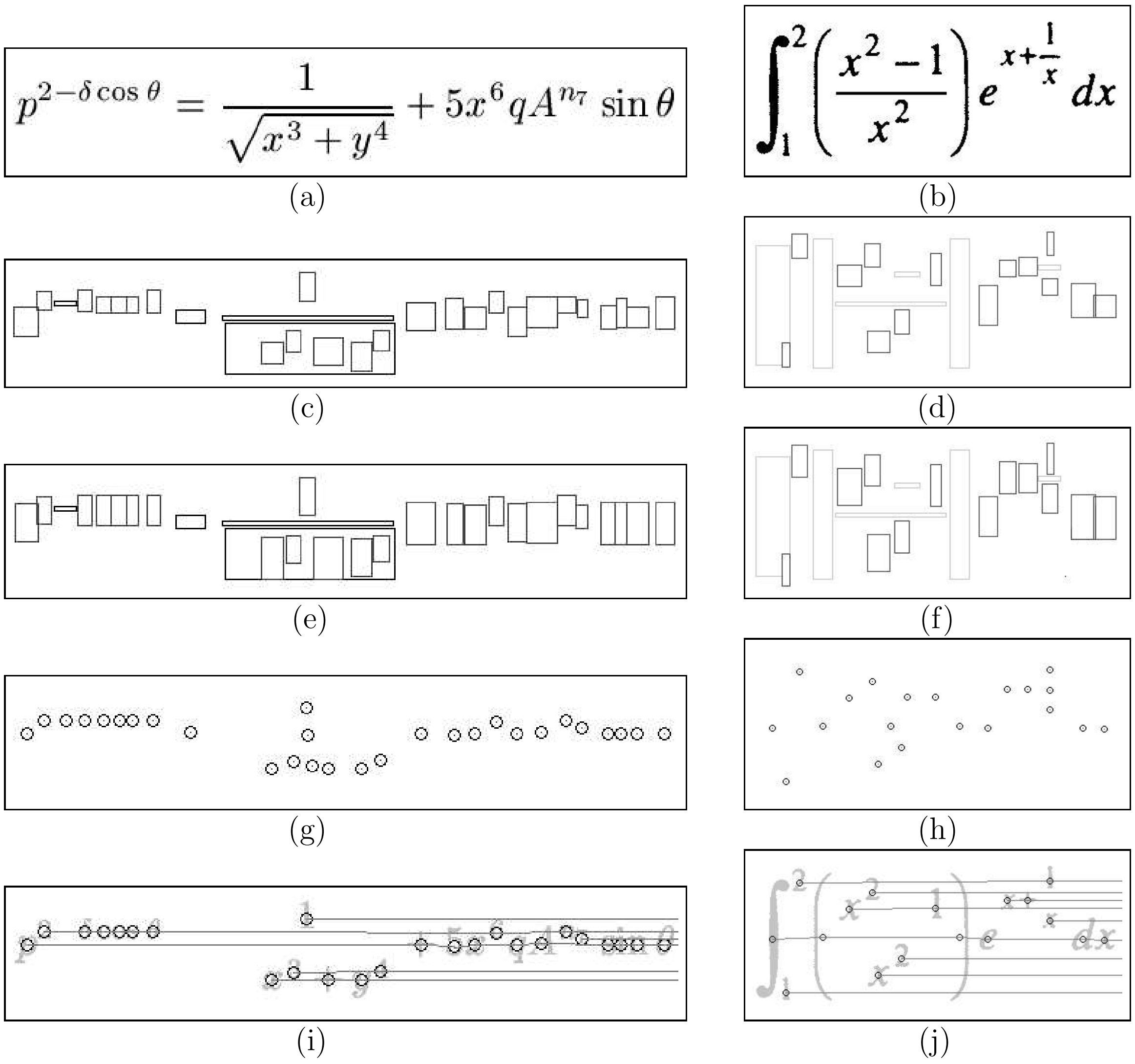


Figure 5.1: Detection of horizontal lines and symbols' L-values: (a) & (b) expression images, (c) & (d) bounding boxes for expression symbols, (e) & (f) extended bounding boxes for symbols, (g) & (h) symbol centres, (i) & (j) horizontal lines (excluding the *HEES* symbols).

- Property 1. *Enclosing Zone (EZ), Extended Bounding box (EB), and Center (C) of a Symbol* : Characters like Roman letters, Greek symbols, etc. exhibit three zones in a text line. Let *Enclosing Zone* (EZ) refer to all the three zones and is represented by a pair of y-coordinates (eyt, eyb), eyt and eyb are being the y-coordinates of the top and bottom rows of the zone, respectively. If the symbols A, a and p occur within same Enclosing Zone EZ, usually A and a will have yb-values very close to each other, but the yt-values will vary. Similarly, for symbols a and p, yt-values are close but not the yb-values.

We normalize EZ by assuming that an EZ starts at $yt = 0$ and ends at $yb = 1$. With respect to a normalized EZ, values of (yt,yb)-pair for different class of symbols are as given in Table 5.1. Each row in this table corresponds to one class. Now using Table 5.1, EZ of any symbol s with absolute values of (yt,yb) can be estimated by checking the class to which s belongs to. However, there are some symbols (elastic symbols) which are treated as exceptions for computing their EZ. This is discussed below under Property 1a.

Once $EZ(s) = (eyt, eyb)$ is obtained for a symbol (s), its *Extended Bounding Box* (EB) is computed as $EB = (xl, xr, eyt, eyb)$. For different classes of symbols, the rules for estimating EB from B (*bounding box*) given in Table 5.2. Figure 5.1(e) and (f) show the extended bounding boxes for symbols of expressions in (a) and (b), respectively.

Table 5.1: Symbol Positions with respect to the Normalized Enclosing Zone

Symbol	Normalized	
	top	bottom
$A - Z$, $0 - 9$, $\Gamma, \Delta, \Theta, \Lambda, \Xi, \Pi, \Sigma, \Upsilon, \Phi, \Psi, \Omega, b, d, h, k, l, \delta, \theta, \vartheta, \lambda$	0.00	0.77
$g, j, p, q, y, \gamma, \rho, \varrho, \varsigma, \varphi, \chi, \mu$	0.23	1.00
$a, c, e, i, m, n, o, r, s, u, v, w, x, z, \alpha, \epsilon, \varepsilon, \iota, \kappa, \nu, o, \pi, \varpi, \sigma, \tau, \upsilon, \omega$	0.24	0.77
t	0.13	0.77
$f, \beta, \xi, \phi, \psi$	0.00	1.00

The center of a symbol s is given by the y-center of its $EB(s)$ and is computed as the arithmetic mean of the eyt and eyb values. Let $EB(xl, xr, eyt, eyb)$ denote the extended bounding box for the symbol, s. Therefore, center of s, $C(s)$ is given by $C(s) = (\frac{eyt+eyb}{2})$. Figure 5.1(g) and (h) show the center positions for expression symbols in (a) and (b), respectively.

Table 5.2: Generation of Extended Bounding Box (EB) of Symbols

Symbol	eyt	eyb
$A-Z$, $0-9$, $\Gamma, \Delta, \Theta, \Lambda, \Xi, \Pi, \Sigma, \Upsilon, \Phi, \Psi, \Omega, b, d, h, k, l, \delta, \theta, \vartheta, \lambda$	0	$yb + yh * 0.3077$
$g, j, p, q, y, \gamma, \rho, \varrho, \varsigma, \varphi, \chi, \mu$	$yt - yh * 0.3077$	0
$a, c, e, i, m, n, o, r, s, u, v, w, x, z, \alpha, \epsilon, \varepsilon, \iota, \kappa, \nu, o, \pi, \varpi, \sigma, \tau, \upsilon, \omega$	$yt - yh * 0.4444$	$yb + yh * 0.4444$
t	$yt - yh * 0.1818$	$yb + yh * 0.3636$
$f, \beta, \xi, \phi, \psi$	0	0

- Property 1a. *Elastic Symbols* : There are some symbols that vary in size in different context. For example, there are horizontal lines ('\hline') of different size in the following expression:

$$\frac{a}{b} = \frac{c}{d+e} - \frac{f+g+h}{i}$$

Some other elastic symbols are $\sum$, $\int$, *arrow*, *bracket* symbols, etc. For all these symbols, $eyt = yt$, $eyb = yb$ and $C = (yt + yb)/2$. Among these symbols, certain symbols are horizontally elongated and denoted as $HEES$ (horizontally elongated elastic symbol) symbols. An elastic symbol is identified as $HEES$ if its aspect ratio (i.e. $\frac{width}{height} = \frac{xw}{yh}$) is more than a pre-defined threshold ($\alpha > 1$)[2].

- **Property 2.** *Level* (L) *of a Symbol*: Computation of symbol L-values initially require determination of horizontal lines on which symbols are arranged in an expression and identification of one of the lines as the dominant baseline [113] of the expression. This is done as outlined in Algorithm 1.

The Algorithm 1 defines `level` (L) values for each symbols (excluding those labelled as $HEES$). Symbols of same L-values define a *horizontal line* (HL) in the expression. Let ℓ be the number of such lines, $HL_1, HL_2, \ldots, HL_\ell$, where HL_i is defined by its member symbols having L = i. This ℓ basically determines the geometric complexity (GC) of an expression as discussed in section 2.2.2 of Chapter 2. Center of an HL ($C(HL)$) is computed as an average of its members' center values. Figure 5.1(i) and (j) show the HLs found for expressions in (a) and (b), respectively. Next, symbols detected as HEES are added to different

[2]In our experiment, $\alpha = 2.5$ has been used to detect HEESs

HLs. An HEES (s_{HEES}) is added to HL_i (i.e. L(s_{HEES}) is assigned to i) if $|\, C(s_{\text{HEES}}) - C(HL_i)| \leq |\ C(s_{\text{HEES}}) - C(HL_j)|\, \forall j$.

Algorithm 1: Computation of L-values.

```
For all symbols (excluding those identified as HEES):
Step 1. Sort the symbols in ascending order of their centers ($C$-values).
        Let $s_1, s_2, \ldots, s_k$ be this sorted list of symbols.
Step 2. level = 0; i = 1;
        L($s_i$) = level; i = i + 1;
        while($i \leq k$)
           if (( $|\, C(s_i) - C(s_{i-1})| + 1$) $< \beta[s_{i-1}.yh]$) /* see comment[3]*/
                 L($s_i$) = level;
           else
                 level = level + 1;
                 L($s_i$) = level;
           end if
           i = i + 1;
        end while
```

Once all symbols are arranged in different HLs, dominant baseline is selected. The HL contains the left-most expression symbol (say, s_l where $s_l.xl \leq s_i.xl$, $\forall\ i$). L-values of the symbols belong to this HL line is re-assigned to *zero* (0) and L-values of other symbols are changed with respect to this new HL_0. L-values of the symbols belonging to the HL just above this line (i.e. the new Hl_0) are assigned to 1 and similarly, L-values of the symbols belonging to the HL just below the dominant baseline are assigned to -1. In this way, L-values increases above and decreases below the dominant baseline. Lines (along with symbols) with re-assigned values have been shown in figure 2.4(b) for the expression in figure 5.1(b).

The above algorithm for determining symbols' L-values (or in other sense, the concept of symbols' `level` itself), is not dependent on parameters like font faces, type style, character size, etc. This is because symbols within a structure convey their meaning based on their spatial arrangement or layout. The horizontal lines (i.e. HLs that essentially provide information on geometric complexity (`GC`) of an

[3]The value of β is determined empirically. In our experiment, $\beta = 0.25$ has been used.

expression) on which symbols are arranged in expressions are important part of this layout. Therefore, if this layout were to depend on font faces, styles, etc., the meaning of the same expression would have been changed with the variations in these parameters. As this does not happen, the proposed method for finding HLs (or symbols' L-values) is quite robust against the change in certain parameters like font faces, type style, character size, etc.

However, the method is somewhat sensitive to the problem attributed to skew. Uniform document skew can be tackled at the pre-processing phase dealing with binarization, page segmentation, etc. (discussion on these topics is beyond the scope of this thesis), but if expression symbols exhibit uneven skew among themselves, the proposed method for determining symbols' L-value may fail to provide exact results. In such cases, a pre-defined static value of β (the user-defined parameter in Algorithm 1) may not assign the correct `level`s to all expression symbols. This problem has been illustrated later in scetion 5.4 (refer figure 5.5) of this Chapter.

- Property 3. *Reduction Ratio* (RR): Consider an expression A^A. The size (characterized by the height) of symbol reduces from the base level to the script (superscript or subscript) level. The ratio by which the height of a symbol (say, s) decreases from the base level to script level will be called the *reduction ratio* of that symbol s, and denoted by $RR(s)$. In general, the *reduction ratio* of a symbol s is defined as

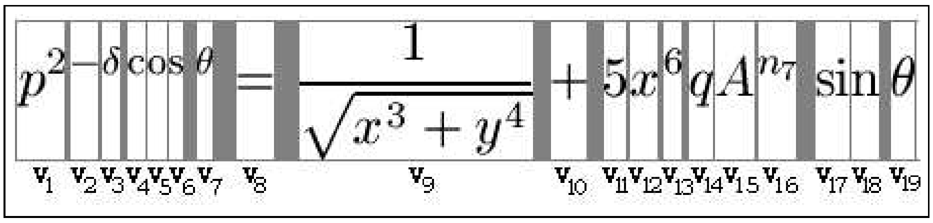
$$RR(s) = \frac{\text{height of } s \text{ as super/subscript}}{\text{height of } s \text{ as base}} \tag{5.2}$$

  The RR-values of different symbols have been studied and the mean and standard deviation were found to be 0.60222 and 0.02371, respectively. Thus, the 3σ-limit for RR–values is given by the interval $(0.53109, 0.67335)$ and all the observed RR-values to lie in this interval. In other words, there is a reduction in height of a symbol by at least 32%(approx.) and at most 47%(approx.) at the script (super/sub) level with respect to its height at base level. This feature helps us to determine a *script* relation between a pair of symbols (or symbol groups) deferring in their L-values.

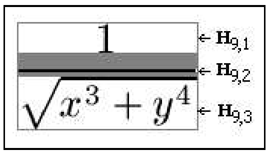
- Property 4. *Space between Symbols*: The average space between two symbols, occurring in the same line, side by side, was measured as percentage of the height of their (normalized) Enclosing Zone (EZ). This measure is denoted by sp and shows a mean and variance of 0.05392 and 0.00050, respectively. This measure is used to identify function words (e.g. 'sin', 'log', 'lim', 'max', etc.) in expressions.

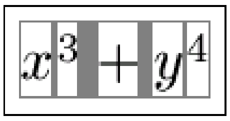
5.3 Symbol Arrangement Analysis

Let S be the set of pre-processed symbols (i.e. each symbol is tagged with its identity, bounding box info (B), enclosing zone (EZ), extended bounding box (EB), center (C), level (L-value), etc.). The symbols S are sorted in ascending order of their xl values. Next, expression structure is interpreted and coded into a TEX string as described below.


$p^2 - \delta \cos\theta = \frac{1}{\sqrt{x^3+y^4}} + 5x^6 q A^{n_7} \sin\theta$
V_1 V_2 V_3 V_4 V_5 V_6 V_7 V_8 V_9 V_{10} V_{11} V_{12} V_{13} V_{14} V_{15} V_{16} V_{17} V_{18} V_{19}

(a)





(b) (c)

Figure 5.2: Structure Analysis of an expression image: vertical and horizontal segmentation.

5.3.1 Expression Reconstruction

The final TEX string for the input expression is generated according to the steps outlined in Algorithm 2. Initially, at Step 2 (see Algorithm 2) the expression image is divided into n vertical stripes (called *vStripe*) based on the white space between horizontally adjacent symbols. Figure 5.2(a) shows the *vStripe*s for the image in figure 5.1(a), where $n = 19$. Next, symbols under each *vStripe* are further segmented into horizontal stripes (called *hStripe*) based on the white space between vertically adjacent symbols.

This vertical and horizontal segmentation continue until each stripe contains a single symbol (Step 1 of the algorithm) or no further segmentation is possible. For the former case TEX equivalent of the symbol is returned (Step 1) and in the later case, further processing is invoked. For example, the *vStripe*, V_9 of figure 5.2(a) is partitioned into three *hStripes* (see figure 5.2(b)), of which both $H_{9,1}$ and $H_{9,2}$ contain a single symbol. In case of $H_{9,3}$, no further segmentation is possible and Step 4.1 is invoked. Here, the largest (based on bounding box area) symbol (i.e. $\sqrt{}$) is separated and *procExp()* is invoked recursively on the rest of the symbols (see figure 5.2(c)). Similarly, the vStripe

V_1 (likewise V_{16} and V_{17}) of figure 5.2(a) does not show any further segmentation and at Step 4.1, the largest symbol 'p' is separated leaving only a single symbol i.e. '2'. At this stage, the grammar, G (given in the next subsection) is applied to search the relation between p and 2 to produce the TEX string '`p^{2}`'. Application of the rule, $E \rightarrow S\hat{}\{S\}$ (discussed later in section 5.3.2) uses information like extended bounding box (EB), reduction ratio (RR), and L-values of symbols.

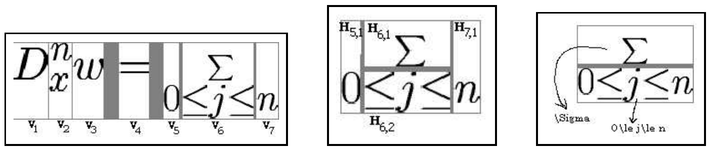
Once each *hStripe* of a *vStripe* are converted into their equivalent TEX strings (at Step 5), *vStripe*s are checked and processed in a pairwise left to right manner (see Step 5.1). At this step, a symbol of V_i gets merged with another symbol of V_{i+1} if their L-levels are the same. For example, symbol in $H_{5,1}$ of V_5 gets connected with the symbol in $H_{6,2}$ of V_6 as shown in figure 5.3(a) and (b). This merging gets propagated from left to right till no further merging is possible. The *vStripe*s that are involved in such a merging process get fused to form a new *vStripe*, e.g., $H_{5,1}$, $H_{6,2}$, and $H_{7,1}$ of figure 5.3(b) get connected to one another and subsequently, V_5, V_6, and V_7 of figure 5.3(a) get fused to form a new *vStripe* as shown in figure 5.3(c). The function words like *sin*, *cos*, *log*, etc. are formed at this stage.

After execution of Step 5, each *vStripe* is represented by one or more TEX strings corresponding to its *hStripe*s. For more than one TEX strings, Step 6 finds a single TEX string for each *vStripe* using the grammar, G. At this stage if G fails to combine the strings, the algorithm simply passes them to the next step. For example, two *hStripe*s in figure 5.3(c) are merged to generate a single string (likewise, three *hStripe*s in figure 5.2(b)). However, the *hStripe*s n and x in *vStripe* V_2 of figure 5.3(a) cannot be merged to produce a single string and they are simply passed to the next step.

At Step 7, pairwise merging of *vStripe*s is done in a left to right manner. If each of the *vStripe*s in a pair (V_i, V_{i+1}) is represented by a single TEX string then the relation between V_i and V_{i+1} is searched using grammar G. For example, V_{12} and V_{13} of figure 5.2(a) together form 'x^6' using the suitable productions of G. However, when a *vStripe* is not represented by a single TEX string, instead of one relation, several relations are searched to merge the pair. For instance, V_2 of figure 5.3(a) is represented by two TEX strings, namely, 'n' and 'x' and hence, when the pair (V_1, V_2) is considered for merging, relation between 'D' and 'n', and between 'D' and 'x' are searched and a single TEX string is returned using several productions.

Algorithm 2: Expression Reconstruction.

```
TeX String procExp($S$)
Step1: If $S$ contains only one symbol then return(TeX equivalent for the symbol);
Step2: Segment $S$ into $n$ non-overlapping vStripes, say, $V_1$, $V_2$, ..., $V_n$.
Step3: For $i = 1$ to $n$ Segment $V_i$ into $m_i$ non-overlapping hStripes, say,
        $H_{i,1}$, $H_{i,2}$, ..., $H_{i,m_i}$.
Step4: If $n = 1$ (i.e. only one vStripe) and for that stripe if $m_i = 1$ then separate the largest
        (based on bounding box area) symbol. Let TeX equivalent of this symbol be $s_l$.
        Return getRelation($s_l$, procExp($S$ - $s_l$)), where getRelation(a,b) uses the grammar,
        $G$ and returns a single TeX string involving '$a$' and '$b$'.
Step5: For all $H_{i,j} \Big|_{i=1,2,...n}^{j=1,2,...,m_i}$, $s_{i,j}$ = procExp($H_{i,j}$); where $s_{i,j}$ is the TeX string for $H_{i,j}$.
        Step5.1: For $i$ (1 to $n$-1), $j$ (1 to $m_i$) and $k$ (1 to $m_{i+1}$) if $L$-values of $H_{i,j}$
                  and $H_{i+1,k}$ are equal then $s'_{i,j}$ = getRelation ($s_{i,j}$, $s_{i+1,k}$).
        Step5.2: If $s'_{i,j}$ is not NULL then combine $V_{i+1}$ with $V_i$ to form a new vStripe.
                  For $j$ = i+1 to $n - 1$ set $V_i = V_{i+1}$;
                  Decrement n by 1.
Step6: For $i = 1$ to $n$ call procVstripe($V_i$); The procedure procVstripe(V) tries to
        find whether all or some of the hStripes within $V$ can be combined following the
        grammar, $G$ and either returns a single TeX string for $V$ or a set of strings
        corresponding to hStripes which were not merged.
Step7: For $i = 1$ to $n$-1 the vStripes $V_i$ and $V_{i+1}$ are merged.
        The process of merging is guided by the grammar, $G$. For any $j$, if $V_j$ cannot be
        merged with $V_{j-1}$ or $V_{j+1}$ then $V_j$ is ignored and the merging proceeds with the
        rest of the vStripes and returns a single TeX string.
```



(a) (b) (c)

Figure 5.3: Structure analysis of an expression image: merging of vertical stripes.

5.3.2 The Grammar (G)

A grammar is formally defined as a 4-tuple:

$$G = (V_N, V_T, P, S) \tag{5.3}$$

where V_N is a set of non-terminals (variables), V_T is a set of terminals (constants), P is the set of productions or rewriting rules, S is the start or root symbol. It is assumed that S belongs to the set V_N and that V_N and V_T are disjoint sets. The alphabet V is the union of sets V_N and V_T. Depending on the nature of production rules there exist different types of grammar. Here, the grammar used in the above reconstruction algorithm is a context-free one, which has productions of the form $A \to \beta$, where A is in V_N and β is in V^+. The name context-free arises from the fact that the variable A may be replaced by a string β regardless of the context in which A appears.

In our application, symbols that occur in the expressions are considered as terminals. These terminals are grouped into different categories. The number of symbols in each category and name of the *non-terminal* (written inside braces) representing the respective category are (i) Arabic Numerals (*AN*): 10; (ii) Roman uppercase letters (*RU*): 26; (iii) Roman lowercase letters (*RL*): 26; (iv) Greek Symbols (*GS*): 41; (v) Mathematical Operators (*MO*): 25; (vi) Relational Operators (*RO*): 39; (vii) Arrow Symbols (*AS*): 32; (viii) Miscellaneous Symbols (*MS*): 33 (e.g. prime('$\prime$'), for all ('$\forall$'), there exists ('$\exists$'), etc.); (ix) Elastic Symbols (*ES*): 14 (e.g. '$\int$','Σ','\hline', etc.); (x) Brackets (*BR*): 6; (xi) Function Words (*FW*): 32 (e.g. 'sin','cos', 'arg','lim', 'ln',etc.).

Apart from these symbols, V_T contains three more symbols namely, '\\', '$\wedge$' and underscore ('_'), where the last two are used to produce TeX strings corresponding to super- and sub-scripts, respectively. In addition, twenty seven (27) TeX keywords (e.g. 'frac', 'sqrt', 'mathcal', 'ldots', 'hat', 'bar', etc.) are included in the V_T for convenience of generating the TeX strings. Hence, V_T, in total, contains 315 (including *NULL*) terminal symbols. On the other hand, V_N contains 18 non-terminals and P is a collection of 345 productions. The major productions are explained below.

Initially, the input expression, as outlined in Algorithm 2, is recursively segmented into n vertical stripes. This is implemented by the production rule originated at the start symbol, S. This is given in Equation 5.4, where E is a non-terminal that is used to produce a syntactically valid TeX string for a *vStripe*.

$$S \to ES|E \tag{5.4}$$

The productions originated by E are given in equation 5.5, where, *AN*, *RU*, *RL*, *GS*, etc. are non-terminals that finally generate terminal symbols of respective categories mentioned above.

$$\begin{aligned} E \;\rightarrow\; & S\hat{}\{S\} \mid S\_\{S\} \mid \backslash frac\{S\}\{S\} \mid \\ & \backslash stackrel\{S\}\{S\} \mid \backslash sqrt\{S\} \mid \\ & \backslash overline\{S\} \mid \backslash underline\{S\} \mid \\ & \backslash overbrace\{S\} \mid \backslash underbrace\{S\} \mid \\ & \backslash mathcal\{RU\} \mid \backslash ELLIP \mid \backslash ACCENT \mid \\ & \backslash begin\{array\}\ MAT\ \backslash end\{array\} \mid \\ & AN \mid RU \mid RL \mid GS \mid MO \mid RO \mid \\ & AS \mid MS \mid ES \mid BR \mid FW \mid \epsilon \end{aligned} \tag{5.5}$$

The non-terminal *ELLIP* is used to generate four types of ellipses as shown in equation 5.6. Likewise, different accents are generated by the non-terminal *ACCENT* as given in equation 5.7.

$$ELLIP \rightarrow ldots \mid cdots \mid vdots \mid ddots \tag{5.6}$$

$$ACCENT \rightarrow hat\{S\} \mid bar\{S\} \mid \ldots \mid vec\{S\} \tag{5.7}$$

The non-terminal *MAT* used in equation 5.5 takes care of matrix structures. Further expansion of *MAT* is given in equation 5.8. For the sake of simplicity, the grammar, *G* does not retain the alignment information (e.g. left, right or center) for matrix columns.

$$\begin{aligned} MAT &\rightarrow ROW\ \backslash\backslash\ MAT \mid ROW \\ ROW &\rightarrow S\ \&\ ROW \mid S \end{aligned} \tag{5.8}$$

To illustrate how the grammar works, consider the expression shown of figure 5.4. Initially, three *vStripe*s are formed by vertical segmentation (see Algorithm 2) of the image. In the grammar, each *vStripe* is represented by the non-terminal *E* that generates a TEX string for the respective *vStripe*. Concatenation of these TEX strings generate the final expression string. Equation 4.11 demonstrates how the final TEX string for the expression in figure 5.4 is generated by the grammar. Starting with the start symbol *S*,

$$\boxed{\frac{1+\sqrt{5}}{2}=\phi}$$

Figure 5.4: An expression for which the generated TEX string is shown in equation 5.9.

each line of equation 5.9 shows the application of a production to expand the right-most non-terminal.

$$\begin{aligned}
S &\rightarrow E\ S \\
&\rightarrow E\ E\ S \\
&\rightarrow E\ E\ E \\
&\rightarrow E\ E\ GS \\
&\rightarrow E\ E\ \backslash phi \\
&\rightarrow E\ RO\ \backslash phi \\
&\rightarrow E\ =\ \backslash phi \\
&\rightarrow \backslash frac\{S\}\{S\}\ =\ \backslash phi \\
&\rightarrow \backslash frac\{S\}\{E\}\ =\ \backslash phi \\
&\rightarrow \backslash frac\{S\}\{AN\}\ =\ \backslash phi \\
&\rightarrow \backslash frac\{S\}\{2\}\ =\ \backslash phi \\
&\rightarrow \backslash frac\{E\ S\}\{2\}\ =\ \backslash phi \\
&\rightarrow \backslash frac\{E\ E\ S\}\{2\}\ =\ \backslash phi \\
&\rightarrow \backslash frac\{E\ E\ E\}\{2\}\ =\ \backslash phi \\
&\rightarrow \backslash frac\{E\ E\ \backslash sqrt\{S\}\}\{2\}\ =\ \backslash phi \\
&\rightarrow \backslash frac\{E\ E\ \backslash sqrt\{E\}\}\{2\}\ =\ \backslash phi \\
&\rightarrow \backslash frac\{E\ E\ \backslash sqrt\{AN\}\}\{2\}\ =\ \backslash phi \\
&\rightarrow \backslash frac\{E\ E\ \backslash sqrt\{5\}\}\{2\}\ =\ \backslash phi \\
&\rightarrow \backslash frac\{E\ MO\ \backslash sqrt\{5\}\}\{2\}\ =\ \backslash phi \\
&\rightarrow \backslash frac\{E\ +\ \backslash sqrt\{5\}\}\{2\}\ =\ \backslash phi
\end{aligned}$$

$$\begin{aligned}&\rightarrow \ \backslash frac\{AN\ +\ \backslash sqrt\{5\}\}\{2\}\ =\ \backslash phi\\&\rightarrow \ \backslash frac\{1\ +\ \backslash sqrt\{5\}\}\{2\}\ =\ \backslash phi \qquad (5.9)\end{aligned}$$

5.3.3 Complexity of the Proposed Method

Under Pre-processing phase, the calculation of the center C for each symbol is done in $O(1)$ and hence in $O(n)$ for all n symbols. Symbols are sorted twice, once in increasing C–values and at a later stage in increasing xl-values, involving complexity of $O(n \log n)$. Grouping of symbols into horizontal levels is done in $O(n)$. Similarly, horizontal sets of symbols are found in $O(n)$. Thus, the Pre-processing is done in $O(n \log n)$ time.

Under reconstruction, division of symbols in *vStripe*s and then every *vStripe* into *hStripe*s is done in $O(n)$ time since the total number of *hStripe*s will never exceed the total number of symbols, i.e. n. If *vStripe*s V_i and V_{i+1} which are to be merged contain m_i and m_{i+1} *hStripe*s, respectively, then for every *hStripe* $H_{i+1,t}$ in V_{i+1}, the *hStripe* $H_{i,s}$ is searched so that ($H_{i,s}$ and $H_{i+1,t}$) can be merged. This searching can be done in $O(\log m_i)$ time and hence this merging takes $O(m_{i+1} \log m_i)$ time. Therefore, the merging of *hStripe*s is done in $O(n \log n)$ time. Finally, the pairwise merging (Step 7 of the Algorithm 2) of *vStripes* is done in $O(n)$ time. Thus, the overall time complexity of our proposed approach is $O(n \log n)$ and the space complexity is $O(n)$.

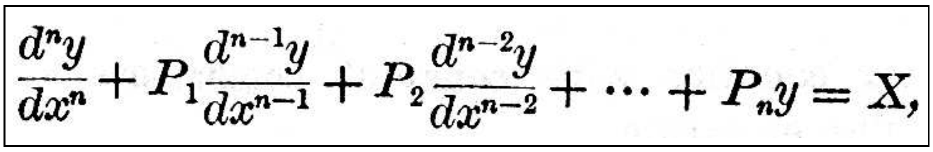
Table 5.3: Structure Recognition Accuracy at Symbol Level

Number of Symbols	Accuracy	
	Detection of Level	Placement
82,691	80,723 (97.62%)	77,515 (93.74%)

5.4 Test Results and Discussion

All the 5,560 expressions (displayed: 2,459 and embedded: 3,101) present in the corpus are used to test our proposed technique. The total number of symbols that appear in all expressions is 82,691. Since *level*s (L) of symbols play important role for their placement, the test procedure initially finds the accuracy for detection of L-value for each symbol. Next, placement of a symbol is checked by looking at the interpretation result for its immediate parent structure that contains the symbol. For example, placement of '7' in $c^{2^{n_7}}$ is assessed by checking the recognition result of the subscript structure 'n_7'. Results obtained for detection of the level and placement for each symbol are shown in Table 5.3. Errors in these two operations are mainly attributed to:

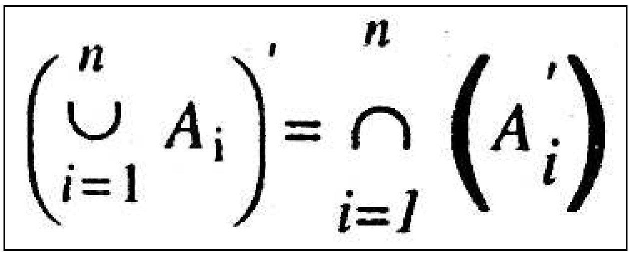
(i) *Uneven skew*: Some of the expressions taken from very old documents show non-uniform skew which is difficult to correct. One such example is shown in figure 5.5. Because of the skew, the symbols '*n*' and '*y*' of the last term of the summation are assigned incorrect L values.



$$\frac{d^n y}{dx^n} + P_1 \frac{d^{n-1} y}{dx^{n-1}} + P_2 \frac{d^{n-2} y}{dx^{n-2}} + \cdots + P_n y = X,$$

Figure 5.5: An expression with uneven skew.

(ii) *Non-uniformity of type-setting*: Documents generated with very old printing technology lack proper layout for typing expressions. Figure 5.6 shows one such example, where the '$\bigcup$' and '$\bigcap$' are placed on the same level but their upper (and similarly lower) limits are typed on different levels. Such improper layout in the original expression leads to error in the detection of *level* for some symbols and eventually, their placements in the final expression become incorrect.



$$\left(\bigcup_{i=1}^{n} A_i \right)' = \bigcap_{i=1}^{n} \left(A_i' \right)$$

Figure 5.6: An expression showing unusual typography.

(iii) *Ambiguous role of a symbol*: Several symbols like *dot*, *horizontal line segment*, etc. represent different meaning in different context. This leads to ambiguous parsing and selection of a particular parse result is difficult. For example, parsing of the expression in figure 5.7 gives different results due to incorrect placement of the *underline* of '*a*', *overline* of '*b*' and the *fraction* lines. Use of some contextual information helps to remove such ambiguities to some extent. However, we view that implementation of a *probabilistic* version of the proposed CFG may help us a lot to resolve many of these ambiguities.

(iv) *Effect of geometric complexity*: The geometric complexity plays a crucial role on the placement of symbols. Test results show that the number of errors in symbol placement increases with higher complexity of expressions. Figures in Table 5.5 also attest

$$\frac{a}{b} = \frac{a}{b}$$

Figure 5.7: An expression with symbols having ambiguous role.

to this observation. For example, the expression in figure 5.8 shows a high geometric complexity and its recognition is incorrect due to wrong placement of symbols having high L values.

(v) *Effect of symbol recognition error*: In a few cases, errors encountered during recognition of symbols also generate errors during determination of symbol levels and eventually for placement of symbols. Some of the touching characters, for which no proper cut positions are found, cause errors in level detection.

$$D_x^n w = \sum_{0 \leq j \leq n} \sum_{\substack{k_1+k_2+\cdots+k_n=j \\ k_1+2k_2+\cdots+nk_n=n \\ k_1,k_2,\ldots,k_n \geq 0}} D_u^j w \frac{n!(D_x^1 u)^{k_1} \cdots (D_x^n u)^{k_n}}{k_1!(1!)^{k_1} \cdots k_n!(n!)^{k_n}}$$

Figure 5.8: Faa-de-bruno's Formula: an expression with high geometric complexity.

Next, the recognition accuracy for individual structures is measured. As explained in Chapter 2, expressions in the corpus are groundtruthed using MathML presentation tags. Each expression is tagged with its geometric complexity GC representing the number of horizontal lines on which expression symbols are arranged in an expression. Using the corpus expressions, it is studied that arrangement of symbols around operators form different geometric layouts, some of which are one-dimensional (1-D) in nature while others are two-dimensional (2-D). For example, operators like '+', '-', '=', function words like 'sin', 'cos', 'log', etc. include 1-D structures whiles superscripts, subscripts, operators (e.g. '$\sum$', '$\prod$', '$\int$', etc.) with limit expressions, matrix, etc. form the 2-D layout. Twelve elementary structures are detected which are two-dimensional in nature. These structures are: (i) *Superscript*, (ii) *Subscript*, (iii) *Fraction*, (iv) *Root*, (v) *Overline*, (vi) *Underline*, (vii) *Overbrace*, (viii) *Underbrace*, (ix) *Ellipses*, (x) *Accent*, (xi) *Matrix* and (xii) *Stacking of symbols*.

A structure is called *nested* if it contains another structure within its scope. For each

2-D structure, its Degree of Nestedness (DoN) is measured, as explained in section 2.2. While GC gives an impression about the geometric complexity of an expression, the value of DoN does the same for an individual structure of an expression. Results for recognition of structures are shown in Table 5.4.

Table 5.4: Structure Level Accuracy

No.	Structure	#Occurrences	DoN Values	Correct Recognition
1.	Superscript	4,267	1, 2, 3, 4	4,033 (94.52%)
2.	Subscript	3,986	1, 2, 3	3,732 (93.63%)
3.	Fraction	2,063	1, 2, 3, 4, 6	1,943 (94.18%)
4.	Root	227	1, 2, 3, 5	207 (91.19%)
5.	Overline	60	1, 2, 3	47 (78.33%)
6.	Underline	13	1, 2	7 (53.85%)
7.	Overbrace	47	1, 2, 4	40 (85.11%)
8.	Underbrace	19	1, 3	16 (84.21%)
9.	Ellipses	828	1	795 (96.01%)
10.	Accent	341	1, 2, 3	321 (94.13%)
11.	Matrix	73	1, 2, 3	64 (87.67%)
12.	Stacking of symbols	154	1, 2	127 (82.47%)
	Summary	12,078	–	11,332 (93.82%)

Finally, the recognition accuracy at the whole expression level is assessed and results are given in Table 5.5. Figures in this table show that the recognition accuracy at the expression level is quite low because placement error for a single symbol makes recognition of an expression incorrect. Therefore, a more reasonable performance measure is presented in Chapter 7 where a performance index is computed to provide a judicious evaluation of accuracy for interpretation of expression structures.

A number of studies (as mentioned in section 1.1 of Chapter 1) deal with structure recognition but it seems difficult to compare the results as the test data used by the authors (except one reported in [113]) are not publicly available. Moreover, the authors define their own way of computing performance and in most cases, this computation is specific to their own techniques. Unlike others, the study described in [113] presents an in-depth analysis of their structure analysis results using expressions from University of Washington Database [87], The best expression-level recognition accuracy reported in [113] is 38%, whereas, in our case it is about 78% (note that the datasets used in the experiments are different). However, since this expression-level accuracy does not reflect the true performance of an expression recognition system, our proposed performance index (presented in Chapter 7) could be viewed as an important evaluator.

Table 5.5: Expression Level Accuracy

Complexity (GC)	Number of Exp.	Correct Recognition
1	1,042	977 (93.72%)
2	1,987	1,640 (82.54%)
3	1,109	835 (75.31%)
4	801	568 (70.91%)
5	162	93 (57.14%)
6	202	117 (57.69%)
7	93	62 (66.67%)
8	70	31 (44.44%)
9	31	8 (25.81%)
10	16	6 (37.50%)
11	23	7 (30.43%)
12	12	3 (25.00%)
13	7	1 (14.29%)
14	3	0 (00.00%)
15	2	0 (00.00%)
Summary	5,560	4,348 (78.20%)

5.5 Summary

A method for structural analysis of printed mathematical expressions is presented in this chapter. Certain geometric properties important for symbol arrange analysis have been outlined and method for determination geometric complexity (`GC`)of expressions and eventually, `level`-values of expression symbols has been discussed. The method for understanding an expression's structure depends of these geometric properties.

The parsing approach uses a context–free grammar. Initially, an expression structure is understood by recursively dividing it into vertical and horizontal stripes until an atomic level is reached. Each stripe represents a token or lexical group. Finally, the context–free grammar is used to merge the tokens one after another to form an equivalent $\TeX$ string for the input expression. The generality of our technique is tested using a large number of expressions taken from various branches of science. Ease of machine implementation is another elegant feature of the proposed approach. The time and space complexity of the parsing technique is also presented.

Experimental results are presented in details. Efficiency of the proposed approach for determination of (`GC`) and symbols L-values has been tested separately as any failure at this stage heavily affects the subsequent stages of expression reconstruction process. Performance of the method for interpretation of 2-D structures has been tested initially at the individual structure level and then at the level of understanding a whole expression.

Number of cases where the proposed method fails have been illustrated and the major reasons for generating errors in interpreting expressions' structures have been outlined.

Special Acknowledgement: A paper [77] based on the approach described in this Chapter was presented in the *7th Int. Conf. on Document Analysis and Recognition* (ICDAR), Edinburgh, Scotland, 2003. We sincerely thank the anonymous reviewers for their valuable comments and suggestions based on which a few modifications were made to the work presented in this Chapter.

CHAPTER 6

RECOGNITION OF ONLINE HANDWRITTEN EXPRESSIONS

6.1 Introduction

There are several ways to input mathematical expressions into digital documents. One of the popular ways is the use of mouse or keyboard where the expressions are entered either in a linear format (e.g. TEX) or by using structured editor (e.g. equation editor available with MS-Word). These approaches are neither convenient nor easy to use because the users require training and practice to work on a language like TEX or any equation editor. Moreover, editors like the one available with MS-Word employ a non-standard storage format which makes further processing difficult. An alternative way is to write expressions by hand and employ a smart system that automatically interprets and enters them into the document under preparation.

The handwritten expressions can be processed under offline or online environments. In the offline environment, expressions already written on a piece of paper is scanned (or digitized) into image files and a OCR system is used to understand the expressions and convert them into some suitable format (e.g. TEX, XML, etc.). The process is unattractive, because it involves a number of time-consuming steps. Moreover, recognition of offline handwriting is more difficult than the online one since directional information, pressure, azimuth, etc. of pen movement is not available in this case. On the other hand, in the online environment, expressions are written on an electronic device (tablet) connected to a computer and the temporal features like pen up and pen down positions, the sequence of strokes and the direction of writing for each stroke are available to the recognizer. It is likely to yield better recognition accuracy and hence more suitable for entering expressions in computer.

This chapter is motivated towards recognition of online handwritten expressions. Like printed expressions, recognition of handwritten ones also involves two major aspects: (i) symbol recognition and (ii) interpretation of two-dimensional (2-D) structures. The problems that were encountered during recognition of printed expressions are also to be tackled under online environment. However, the handwritten data impose some additional difficulties due to variations in writing styles of different people. Each writer has his/her own writing style giving different shapes for the same character. Many symbols are composed of more than one stroke and many writers tend to connect and abbreviate the strokes of a character. Significant stroke number and order variation is observed with writer variation.

On the other hand, expression symbols, as explained earlier, use spatial relationships to indicate logical relationships among them. For example, structures like superscripts, subscripts, implied multiplication, matrix, etc. are indicated implicitly by the geometric layout of operands. Given a spatial relationship between two symbols, it is difficult to determine the logical relationship between them. The ambiguity in spatial relationships is substantially increased for handwritten expressions.

There exist several research efforts towards recognition of handwritten expressions. Survey of the existing approaches is presented in section 1.1.2 of Chapter 1. These surveys reveal that the studies dealing with processing of online handwritten expressions are very few in number. Moreover, the reported techniques have mostly used a single classifier where contextual information is rarely used. Structure analysis has been done more or less in an offline manner and the online spatio-temporal information has not been exploited efficiently.

The purpose of this chapter is to describe an improved system for understanding online handwritten expressions. The present study differs from the previous ones in a number of ways; namely, (i) it involves two different classifiers to capture wide variations in shape and size of the large number of expression symbols, (ii) different combination methods have been attempted to arrive at an efficient fusion of the classifiers, (iii) online features are used along with several offline interpretations are integrated using a Context-free Grammar to understand 2-D expression structures, (iv) the experiment is done using a large representative dataset containing expressions of various nature, (v) an in-depth study is presented to analyze the experimental results, (vi) in addition, the proposed system extends support for use of numerals and several characters of an Indian Language (IL) script (Devnagari (Hindi)) to enter expressions containing IL digits and letters.

The rest of the chapter is organized as follows. Section 6.2 describes the architecture of the proposed system, format of the raw data provided by the hardware and some of the data pre-processing techniques. Section 6.3 presents symbol recognition method that tries to exploit the neuromotor characteristics of handwriting using ideas behind human learning, especially the way by which the children learns to write under instructions from parents or teacher. The algorithm uses multiple-classifier to achieve high recognition accuracy. Section 6.4 presents our approach for interpretation of expression structures. The technique at first exploits online spatio-temporal information to form smaller symbol groups, though the final expression is constructed offline. Experimental results and related discussions are presented in section 6.5. Also, this section presents a qualitative study to compare our proposed approach with the existing ones. Section 6.6 summarizes the chapter.

6.2 Architecture of the Proposed System

Figure 6.1 shows the architecture of our proposed system. The input to the system is handwritten strokes drawn on an electronic data tablet. Each stroke goes through some pre-processing steps discussed next. The recognition method works at the stroke level. Individual strokes are then grouped into a meaningful symbol. A stroke is combined with the next or neighboring one by looking at some spatio-temporal information like time-gap between these two strokes, positional proximity, etc. A symbol written with multiple strokes is recognized by looking at the sequence of its strokes. This sequence is checked using a rule base maintained against each symbol consisting of multiple strokes. To tackle the stroke-order variations, there may be multiple definitions of the stroke sequences for a single symbol. Several structural relations between a pair of symbols are identified online. However, the existence of such relations gets confirmed under an offline processing where the entire expression is reconstructed.

6.2.1 Data acquisition and pre-processing

The raw data recorded by the hardware goes through several pre-processing steps. The main objective of this step is to remove variations (due to noise and uncontrolled pen movement) that would otherwise complicate the recognition process. Several pre-processing steps like 4-connected to 8-connected region generation, interpolation of missing points, smoothing, etc are used. The first level of data compression is achieved as follows.

Let the digitizer output be represented in the format of $(pt[l])_{l=1}^{N} \in \mathcal{R}^2 \times \{0,1\}$, where $pt[l]$ is the pen position having x-coordinate $(pt[l].x)$ and y-coordinate $(pt[i].y)$. For writing expression symbols, N varies from 5 to 50 for a continuous stroke. Let, $s = pt[i] - pt[j]$, where $pt[i]$ and $pt[j]$ are two consecutive pen points. Now the i-th point, $(pt[i])$ is retained with respect to j-th point, $(pt[j])$ if the following condition is satisfied:

$$(s.cx)^2 + (s.cy)^2 > m^2 \tag{6.1}$$

where $s.cx = pt[i].x - pt[j].x$ and $s.cy = pt[i].y - pt[j].y$. The parameter, m is empirically chosen. If m is set to 0, equation 6.1 removes all repeated points and for $m = 1$,

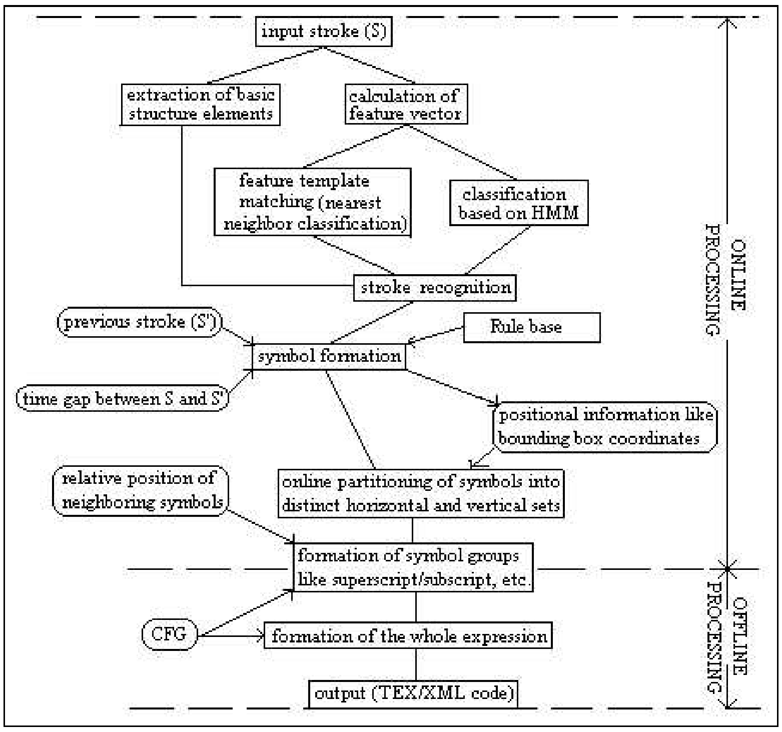

input stroke (S)
extraction of basic structure elements
calculation of feature vector
feature template matching (nearest neighbor classification)
classification based on HMM
stroke recognition
previous stroke (S')
Rule base
symbol formation
time gap between S and S'
positional information like bounding box coordinates
relative position of neighboring symbols
online partitioning of symbols into distinct horizontal and vertical sets
formation of symbol groups like superscript/subscript, etc.
CFG
formation of the whole expression
output (TEX/XML code)
ONLINE PROCESSING
OFFLINE PROCESSING

Figure 6.1: Architecture of the proposed system.

equation 6.1 implements 4-to 8-connectivity conversion.

6.3 Symbol Recognition

For recognition of symbols, our algorithm tries to exploit the neuromotor characteristics of handwriting. Consider the way in which a child learns to write. S/he is advised to down the pen at some position, make straight/curved pen movement in a particular direction, create loops when needed and lift the pen at some other position. Pen down position for the next stroke is also mentioned and s/he follows such instructions until the character is complete. Apart from pen up/down positions and the direction of pen movement, the children are also taught the relative lengths of different parts of a stroke. These aspects are captured and used as features.

6.3.1 Feature Extraction

To elaborate our feature extraction process we use the same notation $\{pt[i]_{i=0}^{N-1}\}$ discussed earlier. Additionally, pen up and pen down information is captured to distinguish a stroke. At first, we extract angle variation information as follows. Let,

$$\begin{aligned} r[i] &= pt[i] - pt[i-1],\ i = 1, 2, \cdots, N-1 \\ \phi_i &= \pi + \text{sign} \times \cos^{-1}(\frac{r[i].cx}{\delta l_i}) \end{aligned} \tag{6.2}$$

where

$$\begin{aligned} r[i].cx &= pt[i].x - pt[i-1].x \\ r[i].cy &= pt[i].y - pt[i-1].y \end{aligned}$$

and

$$\begin{aligned} \text{sign} &= -1,\ \text{if } r[i].cy < 0 \\ &= 1, \text{otherwise.} \\ \delta l_i &= \sqrt{(r[i].cx)^2 + (r[i].cy)^2} \end{aligned}$$

It is to be noted that ϕ_i and δl_i are the angle and the Euclidean distance, respectively between one point and the next one. In reality, instead of angle variation information, direction change information is taught to a child. That is why we convert each into a direction code (an integer) following a 8-direction Freeman coding [36] as shown in figure 6.2. The following expression converts ϕ_i into an 8-direction code, d_i.

$$d_i = \left(\left((\text{int})\left(\frac{8\phi_i}{\pi}\right) + 1\right) \text{mod } 16\right) \div 2 \tag{6.3}$$

where mod is the modulus operator and int returns integer part of a real number. Next, we normalize δl_i which represents local trajectory length. This is done in a straightforward manner:

$$\delta l_i = \frac{\delta l_i}{\sum_{j=1}^{N-1} \delta l_j} \tag{6.4}$$

6.3.2 Description of the Classifiers

Considering the wide variations of handwritten symbols in respect of their shape, size, etc., two different classifiers are used in our system. *Classifier* 1 involves feature template matching approach and employs a nearest neighbor classification scheme, whereas, *classifier* 2 uses Hidden Markov Model (HMM) [90] for classification. The details of the classifiers are described below.

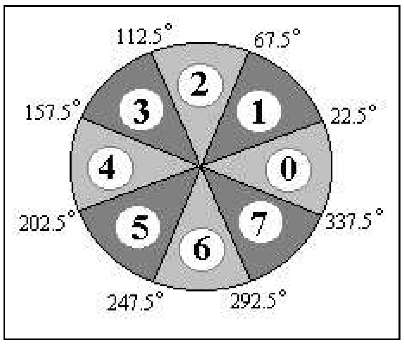


Figure 6.2: An 8-directional Freeman-Chain coding.

- **Classifier 1**: Feature vector used by this classifier is defined by the tuple

$$(d_i, \delta l_i),\ \text{i} = 1, 2, \ldots, N-1 \tag{6.5}$$

Nearest neighbour classification is implemented by a distance measure as follows.

Assume the feature vectors for T (stroke to be recognized) and S (stored prototype) are given by $f_T = \left(d_k^T, \delta l_k^T\right)_{k=1}^{K}$ and $f_S = \left(d_j^S, \delta l_j^S\right)_{j=1}^{J}$, where $\sum_k \delta l_k^T = \sum_j \delta l_j^S = 1$. For $K = J (= N)$ T and S can be compared using

$$J\left(T, S\right) = \sum_{i=1}^{N} 4 - \left|4 - \left|d_i^T - d_i^S\right|\right| \tag{6.6}$$

Note that in 8-directional coding (see figure 6.2) the maximum difference between two successive direction codes can be 4.

In reality, K is rarely equal to J, hence, the condition $K = J = N$ is hardly satisfied. So, the equation 6.6 is modified by defining two more measures given below.

$$\Delta L_k^T = \sum_{i=1}^{k} \delta l_i^T \text{ and } \Delta L_j^S = \sum_{i=1}^{j} \delta l_i^S \tag{6.7}$$

Next, we sort the union of $\left(\Delta L_k^T\right)_{k=1}^{K}$ and $\left(\Delta L_j^S\right)_{j=1}^{J}$ together in an increasing sequence. Let this sequence of numbers be $(\Delta L_r)_{r=1}^{R}$ where $\Delta L_R = 1$. Now, it is clear that the direction codes $\left(d_k^T\right)$ of T and $\left(d_j^S\right)$ of S are constant over $\Delta L_r - \Delta L_{r-1}$.

We can re-define f_T and f_S as $\left(d_r^T, \delta l_r\right)$ and $\left(d_r^S, \delta l_r\right)$, respectively and the equation 6.6 can be re-stated as

$$J(T,S) = \sum_{r=1}^{R} \delta l_r \times \left(4 - \left|4 - \left|d_i^T - d_i^S\right|\right|\right) \tag{6.8}$$

It may be noted that the metric property of J is retained in the equation 6.8 since $\sum \delta l_r = 1$. A formal proof can be found in [39]. For any input stroke T, $J(T, S_i)$ is measured against all stored prototypes S_i and T is classified as S_i if $J(T, S_i) < J(T, S_j)\ \forall j \neq i$. However, in our implementation, classes are ranked based on the J values and the class with lowest J gets the highest rank.

- **Classifier 2**: This classifier uses a left-to-right Hidden Markov Model (HMM) for recognition of symbols. HMM is a general probabilistic structure, which is applicable to a broad class of problems where time evolution is important. HMM is characterized by a Markov chain (discrete time and finite states Markov process) $\{S_t\}_{t=1}^{\infty}$ and an observation process $\{O_t\}_{t=1}^{\infty}$. Let $\{s_i\}_{i=1}^{N}$ be the set of all states of the Markov process while $\{o_i\}_{i=1}^{M}$ be the observation set.

  Notationally, a HMM is denoted by $\lambda = (\pi, A, B)$, where π is the *initial probability vector* defined as $\pi = \{\pi_i\}$, where π_i is the probability of being in state i at the beginning of the experiment i.e. at $t = 1$. A and B are two matrices called the *transition* and *observation* matrix, respectively. Here, $A = \{a_{i,j}\}_{i,j\leq N}$, where $a_{i,j}$ denotes the probability $P_r(S_{t+1} = s_j | S_t = s_i)$ of being in state j at time *t+1* given that the model was in state i at time t and $B = \{b_{i,l}\}_{i\leq N, l\leq M}$, where $b_{i,l}$ denotes the probability $P_r(O_t = o_t | S_t = s_i)$ of observing the symbol o_l at time t when the model is in state i. A HMM (λ) is called stable when its parameters i.e. π, A, and B do not depend on time.

  In this chapter we only consider stable HMMs and estimate their parameters as follows:

  - The states of HMM: For each stroke, we consider its own HMM. Since different writers write a particular stroke in different manners, a number of observation sequences are required to train the model. Let there be η number of such sequences available. Let $O = O_1, O_2, \ldots, O_M$ be a sequence consisting of M observation symbols. Each observation symbol (O_i) is assumed to be a vector of dimension D.

    In our implementation, (O_i) is presented in a two-dimensional vector, $(\phi_i, \delta l_i)_{i=1}^{M}$ where ϕ_i and δl_i and are given by the equations 6.2 and 6.4, respectively. Each of the ηM observation vectors is mapped into one of the N clusters. Each cluster is defined by giving a direction code d_i to ϕ_i based on the Freeman-chain coding formulated in the equation 6.3 and by labeling each segment length δl_i as short, medium or long, based on two predefined thresholds. As d_i can take one of the eight values (zero to seven) and ϕ_i can have one of three descriptions (short, medium, or long), any observation O_i will be mapped into one of the 24 ($N = 8 \times 3$) clusters. Each cluster forms a state (1 to N).

– Training and recognition of strokes: To train the model, we follow the Segmental K-means Algorithm [56]. The initial ($\hat{\pi}_i$) and transition probabilities ($\hat{a}_{ij}$) are calculated as follows:

For $1 \leq i \leq N$

$$\hat{\pi}_i = \frac{\text{Number of occurrences of } \{O_1 \in i\}}{\text{Total Number of occurrences of } O_1 \text{ i.e. } \eta} \tag{6.9}$$

For $1 \leq i \leq N$ and $1 \leq j \leq N$

$$\hat{a}_{ij} = \forall t \frac{\text{Number of occurrences of } \{O_t \in i \text{ and } O_{t+1} \in j\}}{\text{Total Number of occurrences of } \{O_t \in i\}} \tag{6.10}$$

The mean vector and the covariance matrix for each state are defined as follows:

For $1 \leq i \leq N$

$$\hat{\mu}_i = \frac{1}{N_i} \sum_{O_t \in i} O_t \text{ and } \hat{C}_i = \frac{1}{N_i} \sum_{O_t \in i} (O_t - \hat{\mu}_i)^T (O_t - \hat{\mu}_i) \tag{6.11}$$

The observation matrix is estimated by calculating the symbol probability distributions (assumed Gaussian) of training vector for each state as follows:

For $1 \leq i \leq N$ and $1 \leq j \leq N$

$$\hat{b}_i(O_t) = \frac{1}{(2\pi)^{\frac{D}{2}} \left|\hat{C}_i\right|^{\frac{1}{2}}} \exp\left[-\frac{1}{2}(O_t - \hat{\mu}_i)\,\hat{C}_i^{-1}\,(O_t - \hat{\mu}_i)^T\right] \tag{6.12}$$

To improve the above estimation, the optimal state sequence $S^* = (s_1, s_2, \ldots, s_M)$ is found for each training sequence using $\hat{\lambda}_i = \left(\hat{\pi}_i, \hat{A}_i, \hat{B}_i\right)$ computed in the previous step. This is done by using Viterbi Algorithm [35], which is essentially a dynamic programming approach for maximizing the probability $P(O, s_1, s_2, \ldots, s_M|\lambda)$. A vector is reassigned a state if its original assignment is different from the corresponding estimated optimum state. If any vector is reassigned a new state, the new assignment is used to compute new $\left(\hat{\pi}_i, \hat{A}_i, \hat{B}_i\right)$ following the above steps; otherwise, the model formation is complete.

Once the training is over, we obtain well-estimated HMMs for every stroke (for some strokes more than one corresponding HMMs are maintained). Let us consider a given observation sequence which will configure one stroke. Then we calculate the probabilities of occurrence of the observation sequence against each HMM, λ_i. In actual implementation, the classes (w_i) are ranked based on the $P\left(O|\lambda_i\right)$ values and the class with the maximum $P\left(O|\lambda_i\right)$ gets the highest rank.

- Fusion of the Classifiers: Classifiers are combined following the approach presented earlier in section 4.2.3. Initially, similarity of the classifiers is studied by measuring the agreement between their decisions following the equation 4.7 of Chapter 4. Next, classifiers are combined based on the highest rank, the Borda count, and logistic regression. These combination methods have been briefed in section 4.2.3. Relative merits and demerits of each combination method are studied. Details of experimental results are presented in section 6.5.

6.4 Interpretation of Structures

The geometric structure of the input expression is analyzed and interpreted into a corresponding TEX string, which is finally converted into MathML format. The approach consists of three stages, namely, (i) online interpretations, (ii) offline processing, and (iii) compilation of TEX string. Each of these stages works as described below:

6.4.1 Online Interpretations

For each symbol several information like (i) symbol bounding box (B), (ii) symbol center (C), etc. are noted. The y-center of B is recorded as the symbol's center (C). Each symbol is tagged with a `level` (L-value). The method for assigning symbols' L-values is similar to the one described in section 5.2 (refer Property 2) of Chapter 5. Only modification involved here is that the properties like enclosing zone (EZ), extended bounding box (EB), etc. are not computed for expression symbols.

As the performance of the method for interpretation of expression structure tightly depends on how accurately the symbols' L-values are computed, several restrictions are imposed on writing style. For example, for easy detection of the dominant baseline of the expression, writing of an expression starts with its left-most baseline symbol. In

Algorithm 1 of Chapter 5, it was assumed that the left-most expression symbol belong to the dominant baseline (HL_0) and here, the said restriction provides further information to detect the HL_0 correctly.

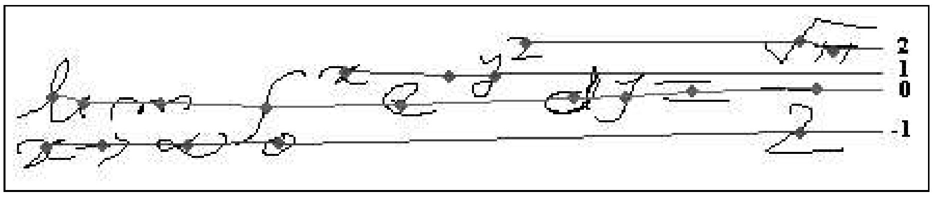
Moreover, the accuracy of results (i.e. assignment of L-values) depends on the value of an user-defined parameter, β used in the Algorithm 1 of Chapter 5. For printed environment, selection of a right value of β is not difficult as symbols are typed following some typographical rules. In case of handwriting environment, choice of a right value for β becomes difficult if writer use arbitrary scaling (i.e. symbols height or width) or place symbols at their own will while drawing the symbols. On the other hand, it is not expected that writers will exactly follow the typographical rules followed in case of typeset characters but at the same time, their writing style should not be far from the idle one (i.e. printed environment). Writers involved in this experiment were made aware of this issue. In spite of that, certain symbols get incorrect L-values because of this problem (mainly due to casual placement of symbols). This has been further discussed in section 6.5.3 analysing the experimental results.

Once symbols are tagged with required information as described above, subsequent processing steps are invoked. For a pair of symbols, their bounding box information and the L-values help to determine any spatial relationship that exists within the pair. In fact, several structures like *square root*, first level *scripts* (superscripts and subscripts), limit, etc. are identified online and corresponding TEX strings are generated for them.

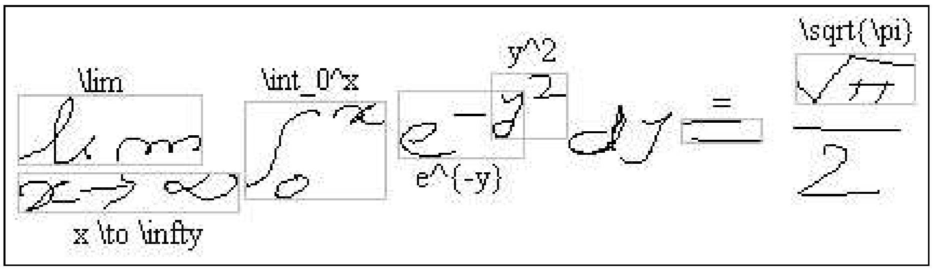
Meaning of certain ambiguous symbols is also understood online. For example, a *dot* ('.') symbol appear in different context like (i) as a decimal point, (ii) as an accent marker ($\dot{a}$), (iii) a part of another symbols like 'i', 'j', ':', ';', or *ellipses* (e.g. '$\ldots$', '$\cdots$'), etc. Therefore, meaning of a *dot* is interpreted by looking at its neighboring symbols (i.e. symbols left, right and below to *dot* and symbol drawn just before the *dot* are examined). Similarly, ambiguity in the role of a horizontal line segment (e.g. *bar*, *fraction line*, *minus sign*, etc.) is also resolved during online processing.

Function words (e.g. *sin*, *log*, *exp*, etc.) are recognized online. Since these words maintain linear 1-D structures, detection of such structures is achieved without much effort. A finite automata is maintained to spot occurrence of any such function word in an input expression.

Figure 6.3 demonstrates the steps described above. Figure 6.3(a) shows how symbols are given their L-values based on their center C(y) values. The structures identified online are shown in Figure 6.3(b). In the present system, writing of a *root* sign imposes a restriction that *root* ($\sqrt{\ }$) symbol is drawn first, then symbols under *root* are written. This simple assumption makes online detection of *root* structure easier. The expression in Figure 6.3(a) contains four horizontal lines whose meanings are also interpreted online. Occurrence of the function word *lim* is also identified at this stage.



(a)



(b)

Figure 6.3: Online processing: (a) assignment of symbols' L-values and (b) recognition of several structures.

6.4.2 Offline Processing

Structures unidentified during online processing are recognized at this stage. Moreover, the relations identified at the first stage are also checked for final acceptance. Initially, the entire expression image is recursively segmented into several vertical and horizontal stripes. TEX strings are generated for each of these stripes. Next, a bottom-up approach is followed to merge two stripes to generate a new TEX string. This merging process continues until the final expression is constructed. Both process, namely, generation of TEX string for each stripe as well as the merging of stripes are guided by a context-free grammar, G, described in section 5.3.2 of Chapter 5.

Figures 6.4 (a) through (e) demonstrate the major processing steps. Steps are almost similar to the ones outlined in the Algorithm 2 of Chapter 5. At first, the expression is divided into n (= 9) vertical stripes (called *vStripe*) as shown in figure 6.4(a). Next, symbols under each vStripe are further segmented into horizontal stripes (called *hStripe*) based on white space between vertically adjacent symbols. Each vStripe and hStripe are tagged to keep track of their order of generation. For example, a tag $H_{9,1}$ for a stripe indicates that it is one of the stripes generated while the vStripe V_9 is segmented horizontally (i.e. projection of black pixels on the vertical axis).

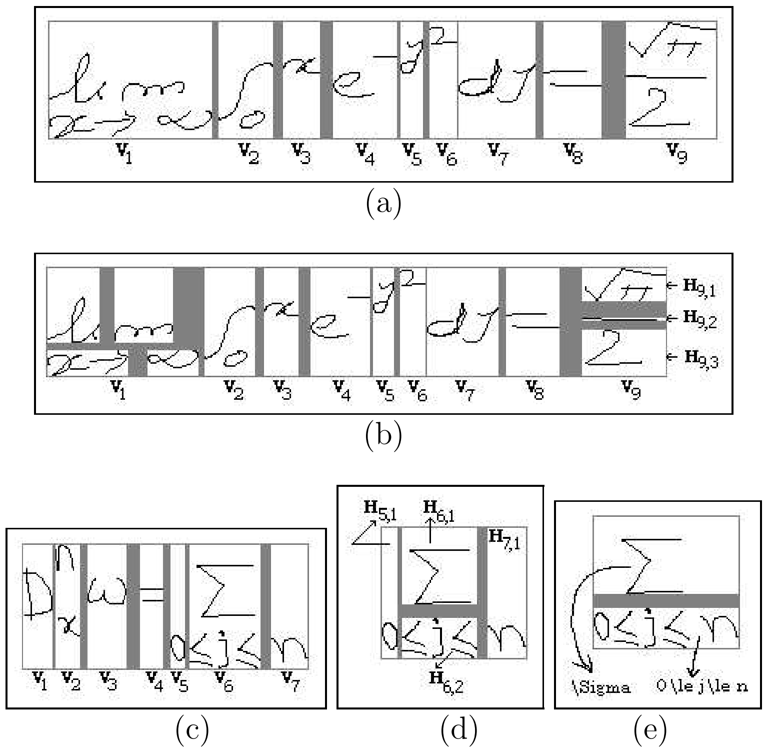


Figure 6.4: Offline processing: horizontal and vertical segmentation.

This vertical and horizontal segmentation go hand-in-hand until each stripe (called atomic box) contains a single symbol, or no further segmentation is possible (as shown in Figure 6.4(b)). Note that the recognizer returns a single symbol (‘=’) for V_8 though it contains two strokes. Combination of two such horizontal lines of almost equal width and lying one above the other into ‘=’ sign is done online. Further processing is invoked for an atomic box containing more than one symbol. For instance, the vStripe V_9 of Figure 5(a) is divided into three hStripes (see Figure 6.4(b)), of which both $H_{9,2}$ and $H_{9,3}$ contain a single symbol and in case of $H_{9,1}$, further processing is done as follows. The *largest* symbol (i.e. the root symbol) is separated and the rest of the symbols are subject to further segmentation. The *largest* symbol is determined by the bounding box area. However, in case of $H_{9,1}$, only one symbol (i.e. the symbol, π) is left and corresponding $\TeX$ strings (i.e. \pi) is returned. Similarly, the vStripe V_2 of Figure 6.4(b) does not show any further segmentation. Therefore, the largest symbol (i.e. the symbol, $\int$) is separated leaving only a single symbol. At this stage, a suitable production rule of G is searched to get the relation between “\int” and “0” is produced.

Once each hStripe of a vStripe is converted into their equivalent $\TeX$, vStripes are processed in a pairwise left to right manner. In this step, a hStripe of V_i gets merged with another hStripe in V_{i+1} if levels $(L)$of both hStripes are the same. Level of a hStripe is determined based on its center, which is given by the midpoint of its enclosing bounding box. For example, $H_{5,1}$ of V_5 (Figure 6.4(c) and 6.4(d)) gets connected with $H_{6,2}$ of V_6. This merging gets propagated from left to right till no further merging is possible. The vStripes that are involved in such a merging process get fused to form a new vStripe, e.g., $H_{5,1}$, $H_{6,2}$, and $H_{7,1}$ of figure 6.4(d) get connected with one another and subsequently, V_5, V_6, and V_7 of figure 6.4(c) get fused to form a new vStripe as shown in figure 6.4(e). The function words like *sin*, *cos*, *log*, etc. are formed following this approach.

When each of the vStripes is represented by one or more TEX strings corresponding to its constituents hStripes, a single TEX string for each vStripe is searched using G. If such a single string is not found, it passes the strings to the next step. For example, three hStripes in figure 6.4(b) are merged to generate a single string following production rule corresponding to *fraction* stated in equation 5.5. However, the hStripes of V_2 i.e. n and x of Figure 6.4(c) cannot be merged to produce a single string and they are simply passed to the next step.

Next, pairwise lumping of vStripes is done in a left to right manner. If each vStripe in a pair adjacent vStripes (say, V_i and V_{i+1}) is represented by a single TEX string then a relation between V_i and V_{i+1} is searched using G. For instance, V_5 and V_6 together form the TEX string, $y\hat{}\{2\}$ using production rule given in equation 5.5. However, when a vStripe is represented by more than one TEX string, several production rules are used to merge the pair. As an example, V_2 of Figure 6.4(c) is represented by two TEX strings, namely, n and x and hence, when (V_1, V_2) pair is considered for lumping, relation between D and n, and between D and x are searched and a single TEX string is returned by using two different production rules corresponding to *superscript* and *subscript* formation. It is to be noted that processing of matrices does not require any extra effort. The algorithm understands matrix structures using production rules given in equation 5.8.

6.4.3 Compilation of TEX strings and use of Contextual Information

Whenever a TEX string is generated at any stage (during online or offline processing) it is checked for its syntactic validity. Any syntactic failure calls for immediate processing with available alternatives. For example, in Figure 6.3(a), *lim* has initially been recognized as *um* which gives two possible interpretations, (i) '*u*' and '*m*' are two variables and multiplication is implied or (ii) the combination *um* is a function word. First interpretation is rejected looking at the *limit* expression $(x \rightarrow \infty)$ below it. In the second case, the generated TEX string "\um" does not pass through the validation check. Hence, the system uses other alternatives provided by the symbol recognition module. For *li* of *lim*, the recognition engine gives *u* as the best choice and *li* as the second best choice, which forms a valid TEX string "lim". If none of the alternatives generates a valid string, the system generates a string with the best choices for its constituent symbols and leaves it for manual correction at a later stage.

Several contextual information are used while generating a new TEX string by merging the boxes. For instance, vStripes, V_2 through V_7 (see Figure 6.4(b)) are merged and a TEX string i.e. "\int_0^x e^{$-y$^2} dy " is formed. Here, "*dy*" seems to a multiplication of '*d*' and '*y*'. However, presence of "\int" looks for its differential and converts the string into "\int_0^x e^{$-y$^2}\, dy ", where *dy* conveys its actual meaning.

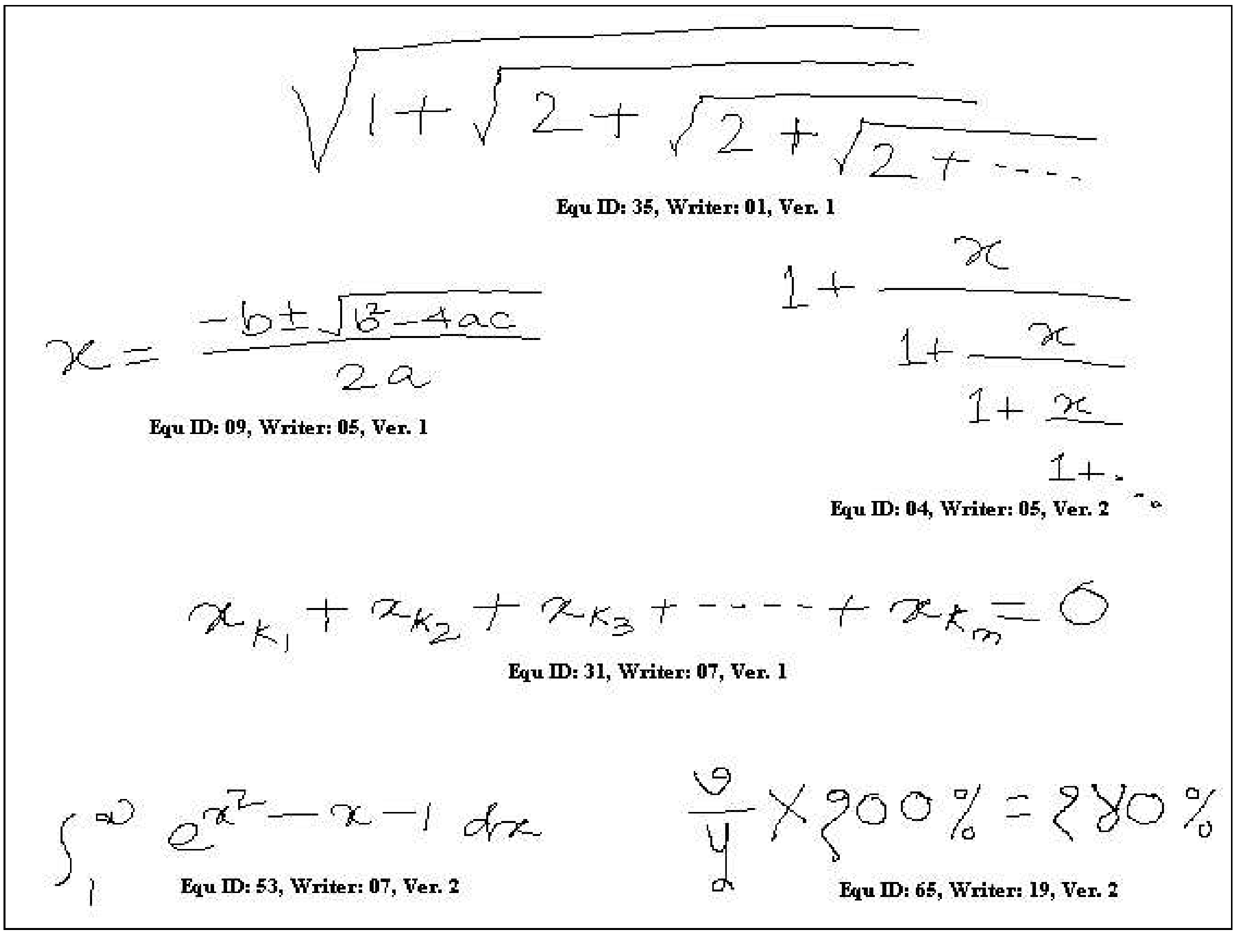

Equ ID: 35, Writer: 01, Ver. 1
Equ ID: 09, Writer: 05, Ver. 1
Equ ID: 04, Writer: 05, Ver. 2
Equ ID: 31, Writer: 07, Ver. 1
Equ ID: 53, Writer: 07, Ver. 2
Equ ID: 65, Writer: 19, Ver. 2

Figure 6.5: Some handwritten expressions used in the experiment.

6.5 Experimental Results and Discussions

At first, a database of handwritten expressions is constructed to test the proposed system. A Genius-make electronic tablet attached with 733 MHz IBM PC is used to capture data. The sampling speed is 40 points per second. One hundred and seventy five (175) expressions are taken from different branches of science. These 175 expressions are collected under three parts. Part-I contains samples for 100 expressions selected from various topics covered in science (mainly Mathematics) books of school and college standard. Table 6.1 gives an idea of how the expressions are divided among different topics and purposes. These 100 expressions are written by twenty students studying at the high school and college level. Two versions of each expressions are recorded from each writer. Versions of an expression are not taken from a writer in a successive manner, rather, they are recorded on different days. The database, therefore, contains `4,000` ($100 \times 20 \times 2 = 4,000$) samples for the first 100 expressions. Some samples (tagged with identification number) from the database are shown in figure 6.5.

Part-II contains samples for 50 expressions taken from the dataset used by Raman [91]. Reasons for considering these expressions are stated earlier in Chapter 2. The main reason is the variability observed in symbols as well as in expression structures. Ten writers selected from those engaged in research and teaching at the university level were asked to write two versions for each expression. This added `1,000` ($50 \times 10 \times 2 = 1,000$) more samples to the database.

In Part-III, the database contains 25 expressions containing digits and letters of Devnagari (Hindi) scripts. These are selected from Hindi medium high school level science books. In some expressions it is observed that Hindi symbols are mixed with Arabic numerals, Roman and Greek letters. Ten native Hindi writers are asked to write each of these 25 expressions twice resulting in a set of `500` ($25 \times 10 \times 2 = 500$) samples.

6.5.1 Groundtruthing of Expressions

In the database, an expression and its corresponding samples are separately groundtruthed. Expressions are truthed into `.exp` files. Generation of truthed data follows the guidelines discussed in Chapter 2. Each of the 175 expressions is given a unique identification key `exp-id` which is also used to name the `.exp` files. Each such file contains a short description about the `source` of expression and some associated comments, if needed. MathML presentation tags are used to encode the expression contents. As explained in Chapter 2 and demonstrated in Appendix-B, a few user-defined tags like ⟨`GC`⟩, ⟨`symbol-level`⟩, etc. are used to truth the expressions. However, tags like ⟨`style`⟩, etc. are not relevant to

Table 6.1: Coverage of the Dataset of Handwritten Expressions

Source	Area	#Expressions	Sample Collection
Part-I: English medium science books (mainly maths) taught at high schools and colleges.	Algebra	22	Expressions are written by 20 students studying at high school/college level. From each writer two different versions are recorded for each expression giving 4,000 samples.
	Calculus	10	
	Differential Equations	10	
	Integrals	10	
	Logic & Set Theory	8	
	Statistics/Prob. Th.	10	
	Trigonometry/Geometry	10	
	Vector	10	
	Misc. (Physics, etc.)	10	
Part-II: AsTeR dataset [91]	Series	5	Ten writers who are in research and teaching at university level and asked to write each of the 50 expressions twice generating 1000 samples.
	Logarithms	3	
	Fractions	8	
	Roots	4	
	Sums	3	
	Super/subscripts	7	
	Limits	2	
	Matrix	1	
	Trigonometry	5	
	Integrals	5	
	Miscellaneous	7	
Part-III: Hindi medium science books taught at high schools.	Elementary level Algebraic problems that use several Hindi numerals and letters.	25	Ten native writers write each expression twice giving 500 samples.

the handwritten data.

Handwritten versions of the expressions are recorded in `.sam` files. Each handwritten sample is tagged with three keys `writer-id`, `exp-id` and `version-id`. Combination of these three attributes form a unique key to identify a sample in the database. The key `writer-id` comes from a writer table containing writer details (like name, age, native language, profession, academic qualifications, etc.). The key `version-id` keeps track of different versions of the same expression written by the same writer.

Next, online data for each sample is recorded in a specific format. Let N be the number of strokes $(S_1, S_2, \ldots, S_N)$ used to write an expression by a writer and P be the total number of points recorded, $p_1, p_2, \ldots, p_N$ (where $P = \sum_i p_i$) be the number of points recorded for N successive strokes and $t_1, t_2, \ldots, t_{N-1}$ be the respective time gaps (measured in milliseconds) between strokes S_i and S_{i+1}. Using these notation the storing format for online data is represented as $< N,\ \{p_i,\ p_i$ number of co-ordinate points, $t_i\} >$. The patterns within curly braces get repeated for N times and S_N being the last stroke, t_N is always set to `NIL` to mark the end of data sequence.

6.5.2 Recognition of Expression Symbols

The total number of distinct symbols present in the database expressions is 3176 of which 1727 come from the set of 100 expressions found in school and college level books. On the other hand, 1072 symbols come from the AsTeR dataset [91] and 377 symbols from a set of Hindi digits and letters. Since the same expression is written by several writers, the total number of samples for these 3176 symbols is 98,060. Details are given in Table 6.2.

Table 6.2: Distribution of Database Samples

Source	#Distinct Symbols	#Writers	#Versions	#Total Samples	#Training Samples	#Test Samples
Part-I	1,727	20	2	69,080	51,807	17,273
Part-II	1,072	10	2	21,440	17,142	4,298
Part-III	377	10	2	7,540	6,037	1,503
Total	3,176	40	–	98,060	74,986	23,074

The symbols in database can be partitioned into seven different groups (number in braces indicates the number of distinct classes under each category): (i) Arabic Numerals (`AN`: 10) (ii) Roman Letters (`RL`: 52), (iii) Greek Symbols (`GS`: 30), (iv) Mathematical Symbols (`MS`: 80), (v) Punctuation Marks (`PM`: 6), (vi) Hindi Numerals (`HN`: 10), and (vii) Hindi Letters (`HL`: 10). The category, `MS` contains the largest number of symbols including binary and relation operators, arrow symbols, bracket symbols and several miscellaneous symbols like *prime* ('$\prime$'), *for all* ('$\forall$'), *there exists* ('$\exists$'), etc.

Therefore, the symbols generate 198 distinct classes for which the samples in the database are divided into two classes, namely, training and test samples. Both of the classifiers (described in section 6.3) are trained on the same set of training samples and are tested with the same test data. The classification results obtained from the classifiers are outlined in Table 6.3 where for an input symbol, only the top rank (class) returned by a classifier is considered.

Analysis of the classification results shows that the classifiers are of comparable power but behave differently for different types of symbols. Classifier-I shows better performance for complex shaped symbols (e.g. Roman letters, Greek symbols, Hindi Letters, etc.) whereas Classifier-II responds better for symbols with less structural complexity (e.g. Numerals, Math operators, punctuation marks, etc.). Next, the similarity between the classifiers is estimated following the equation 4.7 of the Chapter 4 and a value `0.817` is obtained for the similarity index, ρ_C. In the present system, the classifiers are trained on the same data set. Therefore, the value of (ρ_C) may change if classifiers are trained on different training sets.

Table 6.3: Training and testing of the Classifiers

Symbol Type (#Classes)	#Training Samples	#Test Samples	Correct Recognition on Test Set by	
			Classifier I	Classifier II
AN (10)	8,998	2,769	2,453 (88.59%)	2,551 (92.13%)
RL (52)	19,496	6,192	5,676 (91.67%)	5,524 (89.21%)
GS (30)	5,993	1,845	1,702 (92.25%)	1,654 (89.66%)
MS (80)	28,494	8,568	7,747 (90.42%)	7,898 (92.19%)
PM (6)	1,249	692	603 (87.18%)	638 (92.24%)
HN (10)	7,469	2,107	1,868 (88.17%)	1,970 (93.49%)
HL (10)	3,287	901	840 (93.26%)	822 (91.12%)
Total (198)	74,986	23,074	20,889 (90.53%)	21,057 (91.26%)

Finally, the classifiers are combined following three methods outlined in section 4.2.3. Among these three combination techniques, the first two (i.e. the highest rank and the Borda count) do not need any training, whereas the logistic regression needs a training to evaluate its parameters (α, β_1, β_2 in equation 4.11). The training set shown in Table 6.3 is used to compute these parameters. The details of combination results are shown in Table 6.4.

It is to be noted that among the three methods the highest rank method and the logistic regression give comparable performance, the latter being slightly better. On the other hand, the Borda count method does not produce very encouraging results in the present system. This is so because this method does not take into account the differences in the individual classifier capabilities.

Table 6.4: Combination of the Classifiers

Classifiers and their Combinations	% Correct in Top N Choices				
	1	2	3	5	10
1. Classifier-I	90.53%	93.60%	96.25%	96.96%	96.98%
2. Classifier-II	91.26%	94.12%	97.03%	97.69%	97.70%
3. Combination by the Highest Rank	93.18%	96.73%	98.02%	98.92%	98.96%
4. Combination by the Borda Count	91.92%	94.87%	97.41%	98.03%	98.15%
5. Combination by Logistic Regression	93.77%	96.82%	98.09%	98.97%	99.12%

6.5.3 Recognition of structures

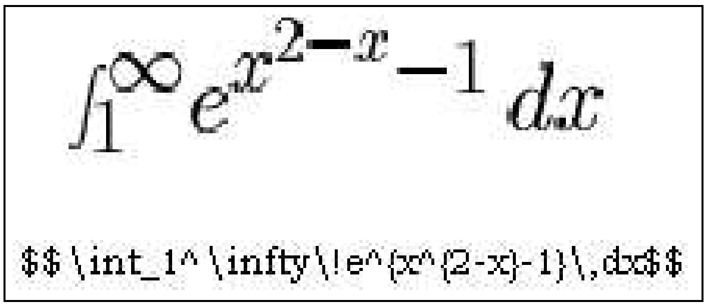
All 5500 handwritten samples for 175 distinct expressions present in the database are used to test our proposed structure analysis technique. The total number of symbols appearing in sample dataset is 98,060. Since detection of *level*s (L) of a symbol plays an important role in its final placement, the test procedure initially finds the accuracy for detection of L-value for each symbol. Next, placement of a symbol is checked by looking at the recognition result of its immediate parent structure that contains the symbol. For example, placement of '2' in e^{x^2} is assessed by checking the recognition result of the superscript structure 'x^2'. Results obtained for detection of the level and placement for each symbol are shown in Table 6.5.

Table 6.5: Structure Recognition Accuracy at Symbol Level

Number	Accuracy	
of Symbols	Detection of Level	Placement
98,060	95,531 (97.42%)	92,049 (93.87%)

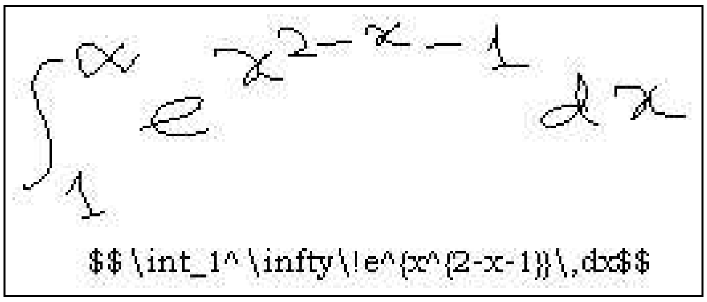
Next, the recognition accuracy for individual structures is measured. Note that expression symbols generate two types of structure: the first type refers to the 2-D structures e.g. script, limit, fraction, root, matrix, etc. and the second type includes different 1-D operators including parentheses, function words like "sin", "lim", etc. After studying the expressions present in the database, fourteen elementary structures have detected.

The first twelve are 2-D in nature. They are (i) *Superscript*, (ii) *Subscript*, (iii) *Fraction*, (iv) *Root*, (v) *Overline*, (vi) *Underline*, (vii) *Overbrace*, (viii) *Underbrace*, (ix) *Ellipses*, (x) *Accent*, (xi) *Matrix* and (xii) *Stacking of symbols.* On the other hand, 1-D structures are grouped into two classes, namely, *Function word* (e.g. 'sin', 'log', 'lim', 'max', etc.) and other 1-D structures (e.g. '+', '-', parentheses, etc).



```
$$ \int_1^ \infty\!e^{x^{2-x}-1}\,dx$$
```

(a)



```
$$ \int_1^ \infty\!e^{x^{2-x-1}}\,dx$$
```

(b)

Figure 6.6: An expression: (a) printed form and (b) handwritten version.

Recognition accuracy for each type of structures is assessed separately. Accuracy is judged by computing the number of structures (as well as operators) properly identified from the total number of such structures present in the expressions. For example, Figure 6.6(b) shows a handwritten version of the expression in Figure 6.6(a) (printed form). The expression contains eight structures. Among them, four are 2-D in nature, namely, two limit symbols of the integration, as well as script structures for e and x. There are four 1-D structures (or operators), namely, integration, two subtractions, and dx. After structural analysis stage, the system generates a TEX string shown in Figure 6.6(b) below the handwritten expression. The correct TEX string is shown below the printed expression in Figure 6.6(a). It is seen that the generated string differs in three positions, namely, (i) superscript of e, (ii) superscript of x, and (iii) placement of minus operator before 1. Hence, 2 out of 4 2-D structures and 3 out of 4 1-D structures are recognized correctly. The details of structure recognition results are presented in Table 6.6. A semi-automatic way of evaluating expression recognition results is presented next in Chapter 7.

Table 6.6: Structure Level Accuracy

Structure Types	Total	Correct Recognition
Scripts	13,246	12,321
Limit	2,588	2,328
Fraction	494	384
Root	248	214
Overline	156	133
Underline	34	26
Overbrace	118	91
Underbrace	52	37
Accent	1,562	1,209
Matrix	38	31
Stacking of Symbols	398	311
Ellipses	2,152	1,941
Function words	1,494	1,399
Parenthesis	1,928	1,843
Other 1-D Structures	10,168	9,936
Summary	34,676	32,204 (92.87%)

Errors encountered in structure analysis stage are mainly attributed to:

- Handwriting Style: The prime reason behind errors in symbols placement is casual handwriting style that create confusion while determining spatial relationship between a pair of symbols. Some restrictions on writing style will definitely help to resolve several ambiguities that arise at the structural analysis stage, but imposition of any such restriction must be done carefully when realizing a practical system.

  At present, the system does not provide any layout structure (like three zones for writing letters/digits or box like structures that appears in different equation editors like one available with Microsoft Word, etc.) related help for the writers. But an interface giving such kind of layout assistance may help the writers to enter expression in a better way. Such an interface may eventually improve the overall system performance.

- Ambiguous role of a symbol: Several symbols like *dot*, *horizontal line segment*, etc. represent different meaning in different context. This leads to ambiguous parsing and selection of a particular parse result is difficult. For example, parsing of the expression in figure 6.7 gives different results due to incorrect placement of the

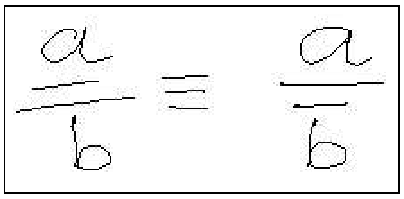


Figure 6.7: An expression with symbols having ambiguous role.

underline of '*a*', *overline* of '*b*' and the *fraction* lines. Use of a *probabilistic* version of the proposed CFG may help us a lot to resolve many of these ambiguities.

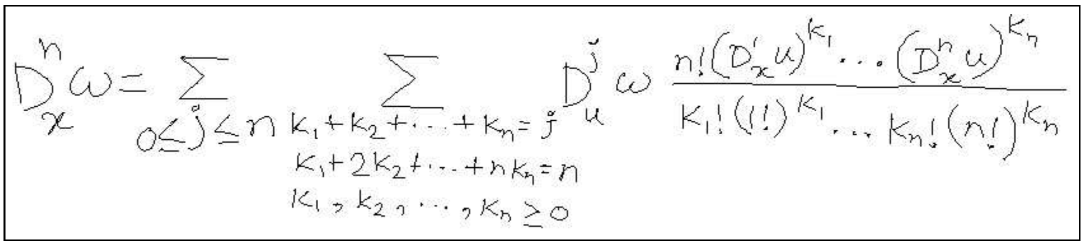
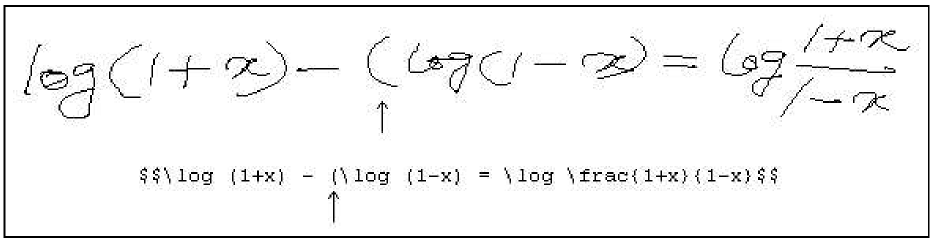


Figure 6.8: Faa-de-bruno's Formula: an expression with high geometric complexity.

- Effect of geometric complexity: The geometric complexity plays a crucial role in the placement of symbols. Test results show that the number of errors in symbol placement increases with higher complexity of expressions. Figures in Table 6.7 also attest to this observation. For example, the expression in figure 6.8 shows a high geometric complexity and its recognition is incorrect due to wrong placement of symbols having high L values.

- Error in Input: Another reason for causing errors during structural analysis is input of an incorrect expression. Figure 6.9 shows an example where the input itself is incorrect (presence of an extra left bracket shown in the figure) and analysis of this expression structure generates an invalid TEX string.

Finally, recognition accuracy at the whole expression level is assessed and results are presented in Table 6.7. Figures in this table show that the recognition accuracy at the expression level is quite low because placement error for a single symbol makes recognition of an expression incorrect. On the other hand, figures show that errors in placement of a few number of symbols are responsible for the majority of the expressions parsed wrongly. Therefore, such a recognition accuracy computed at the expression level



```
$$\log (1+x) - (\log (1-x) = \log \frac{1+x}{1-x}$$
```

Figure 6.9: Input error: a left bracket pointed by an arrow is wrongly written.

does not truly evaluate the system and need for designing a better evaluation strategy is called for. This performance evaluation has been further discussed in the next chapter.

Table 6.7: Expression Level Accuracy

# Expression	# Correctly Parsed Expressions	# Expressions with N Symbol Placement Errors				
		1	2	3	4	≥ 5
5,500	4,121 (74.92%)	522 (37.85%)	409 (29.66%)	243 (17.62%)	122 (8.85%)	83 (6.02%)

Comparing our work with the existing studies is difficult, since each study defines its own dataset for testing. However, we attempt to present a comparison among different works based the nature of test data, accuracy reported for recognition of symbols, structures, whole expressions, etc. Table 6.8 presents such a comparison with relevant comments.

Table 6.8: Comparison of the Studies on Online Handwritten Expression Recognition

Ref.	Dataset				Accuracy Reported			Remarks
Index	#Exp	#Sym	#Writers (Versions)	Size of Test Data	Symbol Level	Struct. Level	Exp. Level	
[4]	8	NR	10 (4)	320 (Exp)	93%	NR	$\frac{314}{320}$	–
[59]	–	82	1 (50)	820 (Sym)	96.9%	–	–	Writer dependent accuracy is reported.
[110]	–	94	20 (5)	NR	90.52%	–	–	Writer dependent accuracy is reported.
[93]	4	NR	20 (2)	160 (Exp)	99.35%	98.46%	NR	Spatial relation between pair of symbols are are detected to compute accuracy
[15]	60	NR	10 (1)	600 (Exp)	99.4%	99.72%	$\frac{532}{600}$	Integrates sym and structure recog. scores.
[101]	NR	NR	NR	NR	80%	92%	NR	Matrix recog. accuracy: 69%.
[112] [111]	4	NR	27 (1)	108 (Exp)	NR	NR	NR	Reported that the writers are satisfied with the output.
[96]	NR	43	1	5,375 (Sym)	$> 99\%$	NR	NR	The system is being used by two German Universities.
Our Work	175	3,176	40 (2)	5,500 (Exp) 23,074 (Sym)	99.12%	92.87%	$\frac{4121}{5500}$	Tested with a large no. of exp. taken from different sources

(NR: Not Reported, Exp: Expressions, Sym: Symbols, Struct: Structure, recog: Recognition)

6.5.4 Processing speed

All of our algorithms are written in *C* and a 733 MHz IBM PC has been used to test the system. The expressions are grouped into three classes based on the number of symbols in them: (i) small size, if number of symbols is less than 7, (ii) medium size, if number of symbols is in between 7 and 15, and (iii) large size, if symbols are more than 15 in number. The time required to process each type of expression and convert it into corresponding TEX string is shown in Table 6.9.

Table 6.9: Processing speed of the proposed system

Expression Type	Time (avg.) required for recognition
Small size	1.82 seconds
Medium size	2.87 seconds
Large size	4.58 seconds

6.6 Summary

In this chapter, a system for online recognition of mathematical expressions is presented. The recognizer uses a multiple-classifier approach. The classifiers have been designed to capture wide variations in shape and size of the large number of symbols that occur in writing mathematics. Because of its high efficiency, the recognizer is applicable in recognizing online handwritten Indian language scripts [40] as well.

The proposed technique uses both online and offline data analysis. A context free grammar is used to parse the input expression. Contextual information has been used at different levels to increase the system efficiency. A in-depth analysis of the test results is presented in this chapter. Compared to the previous ones our system has been tested with a dataset of reasonably big size and the expressions used in testing are taken from a globally available dataset. High accuracy achieved both in symbol recognition and structural analysis stages attest the feasibility of our proposed algorithms.

Because of this study, online entry of mathematics into electronic documents will become more user-friendly. This will help to realize an easy preparation of scientific and technical documents in digital domain. Also, such a study is useful in developing electronic chalkboard [96], pen-based calculator programs [16], etc. The proposed system supports entry of Devnagari (Hindi) online text and it eventually helps to prepare scientific and technical Indian language documents [41] in digital domain.

Special Acknowledgement: A paper[1] based on the preliminary version of the study presented in this Chapter was communicated to *Proc. of the 7th Int. Conf. on Document Analysis and Recognition* (ICDAR), Edinburgh, Scotland, 2003 and we sincerely thank the reviewers for their comments and suggestions on that communication.

Later on, the complete work as presented in this Chapter was sent to the *IEEE Transactions on Systems, Man and Cybernetics* for possible publication. We express our sincere gratitude towards the anonymous reviewers of this paper [2] for their detailed comments and number of valuable suggestions that helped us to a great extent to improve the quality of the work described in this Chapter.

[1]U. Garain and B.B. Chaudhuri, "On Machine Understanding of Online Handwritten Mathematical Expressions," in *Proc. of the 7th Int. Conf. on Document Analysis and Recognition* (ICDAR), pp. 349-353, Edinburgh, Scotland, 2003.

[2]U. Garain and B.B. Chaudhuri, "Recognition of Online Handwritten Mathematical Expressions," in *IEEE Transactions on Systems, Man and Cybernetics*, Part-B, vol. 34, no. 6, pp. 2366-2376, 2004.

CHAPTER 7

POST-PROCESSING AND PERFORMANCE EVALUATION

7.1 Introduction

In this chapter, issues related to post-processing and performance evaluation of expression recognition system are discussed. Post-processing module mainly deals with error detection and correction aspect. On the other hand, performance evaluation is done to measure the efficiency of an expression recognition system.

Recognition of expressions is not expected to be free from errors. Therefore, error detection and correction are important post-processing steps. A good error handling module should locate the presence of errors and correct them properly. At the same time, these operations are to be performed fast so that the resultant system does not slow down.

Although error handling is an important issue, very few studies dealing with this aspect have been reported in the literature. The method proposed by Dimitriadis et. al. [28] provides warning messages whenever an error is detected and asks for human assistance to correct the errors. However, the errors detected by this approach are quite simple in nature. Later on, Lee and Wang [69, 70] proposed a set of rules for correcting errors. The rules are formulated based on some heuristics. These rules mainly try to correct substitution (occurred during symbol recognition phase) errors, whenever possible. More recently, Chan and Yeung [15] present a study that identifies different types of errors occurring while online recognition of mathematical expressions is concerned. The authors proposed an extension of the grammar [14] that is used for parsing an expression. In their approach, the grammar is extended to include all the expected errors into its productions (i.e. grammar rules).

In our approach, initially errors are analyzed to classify them into different groups. Several errors are detected by checking the syntactic validity of the generated $\TeX$ string and then an attempt is made to correct them using a set of rules that are formed by using several contextual information and some heuristics. Section 7.2 presents the proposed error handling module.

The quantitative evaluation of expression recognition results is a difficult task since recognition scheme involves two major stages: symbol recognition and structural analysis. The stages are tightly coupled and therefore, if evaluation in one stage is done independent of the other, then it may not reflect true performance of the system. Errors in the symbol recognition stage affect the structure analysis results. This calls for an integrated evaluation mechanism for judging the performance of system dealing with expression recognition.

Chan and Yeung [15] proposed an integrated performance measure consisting of two independent measures: one for recognition of symbols and another for recognition of operators. These two measures are combined with equal weights. The proposed evaluation is based on manual effort. Later on, Okamoto et. al. [84] have presented an automatic approach for evaluating their structure analysis method. They attempted to evaluate the performance by checking whether each typical structure such as scripts, limits, fractions, roots, and matrices, is recognized correctly. In their approach, expressions against which a system is evaluated are groundtruthed into MathML format. More recently, Zanibbi et. al. [113] presented another automatic way of evaluating the performance. In that approach, an expression is visualized as a set of symbols appearing on different baselines. The performance is assessed by separately counting the number of (i) correctly recognized baselines and (ii) properly placed (w.r.t. the corresponding baseline) symbols.

As the methods proposed in [15, 84] count only the number of properly recognized structures, an error in recognizing a simple structure gets the same weight as that of an error in a complex nested structure. On the other hand, the technique proposed in [113] presents more in-depth analysis of the recognition results, but does not provide a single figure of merit for overall performance evaluation. In our study, we present a new performance-index (γ) that uses geometric (or structural) complexity of an expression to measure the overall performance. Section 7.3 presents the proposed technique for performance evaluation.

7.2 Error Detection and Correction

At first, let us consider the different types of errors that may occur in recognizing expressions. We categorize the errors into three groups: (i) *Segmentation errors*, (ii) *Recognition errors*, and (iii) *Structure interpretation errors*. One type of error may influence the others. For example, segmentation errors, in many cases, affect recognition of symbols or analysis of expression structure. Similarly, recognition errors may also generate inaccuracies for understanding expression structure. Error types and influence of one type on the others are discussed below:

7.2.1 Segmentation Errors

Errors that occur during segmentation of an expression into its constituent symbols are known as segmentation errors. Though we discuss such error in connection with recognition of printed expressions, system dealing with recognition of handwritten expressions is also not free from this. However, in our system for handwritten expressions, two subsequent symbols are separated by a time gap relatively larger than that between two strokes of the same character. Hence, online recognition of expression, in our case, do not encounter segmentation problem, but to realize a system that does not impose any limitation on drawing of symbols, one has to consider proper segmentation of expression symbols.

In general, segmentation errors observed in printed expressions are of two types: (i) errors that do not impose any recognition problem and (ii) others, which lead to recognition errors. However, both types of errors result in wrong interpretation of structures. Figure 7.1(a) demonstrates one such example [1] where the *root* symbol is wrongly segmented into parts namely, $\surd$ and a *horizontal line* segment. As far as symbol recognition is concerned, both of these parts are correctly classified by the recognition engine but interpretation of expression structure gets affected by this segmentation error. During interpretation the detached horizontal line is interpreted as an *overline* above the *numerator* part under *square root* operator and therefore, the resultant TEX string (though syntactically a valid one) does not correspond to that of the original expression. The expression after recognition is shown in figure 7.1(b). At present, our system cannot correct such errors since compilation of the generated TEX string does not reveal any syntactic problem. Matching of the input and output expressions at the image level may be helpful to locate and correct such errors. We would like to consider this in future extension of the present study.

The second type of segmentation error has a direct influence on the symbol recognition engine. Touching and broken characters fall in this category. In our system, a special module takes care of segmentation of touching characters, as described in section 4.3 of chapter 4. However, failure in segmentation of a touching character results in symbol recognition errors. Similarly, improper merging of the disconnected parts of a broken character leads to recognition errors (either rejection or insertion type of errors). Segmentation errors of this type are difficult to correct immediately during segmentation phase itself. However, a few of these errors, as demonstrated in figure 7.2 are corrected during interpretation of structures.

[1]This expression has been taken from [44]

$$\gamma^{k}_{\nu,\,k+1} = -\sqrt{\frac{(\nu + k + 1)\,(k - \nu + 1)}{(2k + 1)\,(2k + 2)}}\,\beta^{k}_{k+1,\,k+1}$$

(a)

$$\gamma^{k}_{v,k+1} = -\sqrt{\tfrac{(v+k+1)(k-v+1)}{(2k+1)(2k+2)}}\beta^{k}_{k+1,k+1}$$

(b)

Figure 7.1: Segmentation error: (a) Image of an input expression, (b) The output expression on recognition.

7.2.2 Recognition Errors

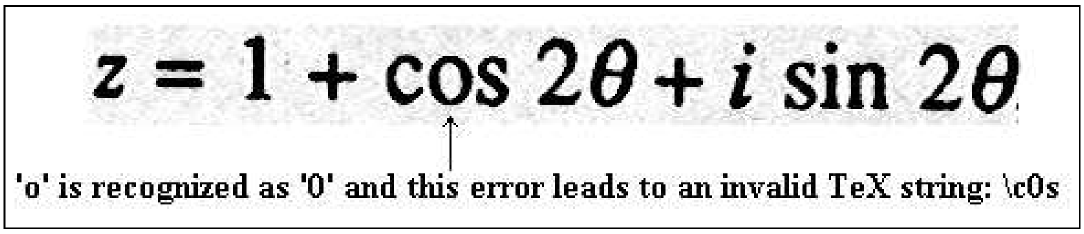
Recognition errors are of three types: (i) *Substitution error*: a symbol is misrecognized as another symbol. (ii) *Rejection error*: the recognition engine cannot classify a symbol and hence, rejected. Rejected symbols are generally signaled by a special mark. (iii) *Insertion error*: One or more extra symbols appear in the recognition result. This occurs mainly due to the segmentation error where a symbol is wrongly segmented into more than one parts (e.g. broken characters).

Substitution errors, as demonstrated in figure 4.10 under section 4.4, occur because of the shape similarity among several characters. Some of these errors affects the structure analysis phase. Figure 7.2(a) demonstrates an example where the character '*o*' in '*cos*' is wrongly recognized as '0', then it generates a string '*c*0*s*' which does not pass through the syntactic validation check when the final TEX string is compiled. This is because TEX has no keyword as 'c0s'. In such cases, other alternatives provided by the recognition engine is used to correct the errors. Similar syntactically invalid TEX strings are generated for substitution errors like character 'C' is recognized as '('. Section 6.4.3 has addressed this issue.

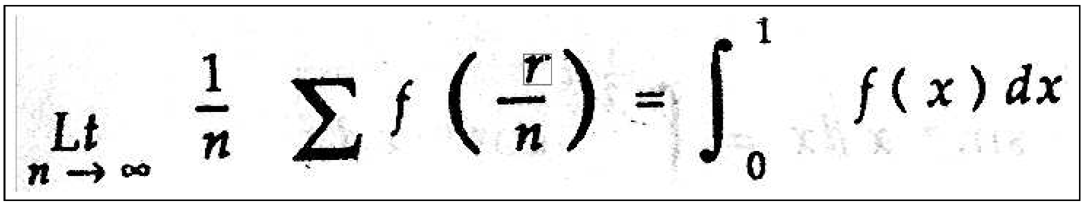
However, many of the substitution errors like 'P' as 'p', '1' as 'l' ('ℓ') do not affect interpretation of structures but reduces the overall recognition efficiency of the system. The method for correction of such errors checks the context around the erroneous character. Occurrence of such errors are suspected by looking at the symbol confusion matrix. Errors where a lowercase character is recognized as uppercase or vice versa (e.g. 'P', 'p'; 'X', 'x'; etc.) are corrected by considering the symbol image and comparing its height (or vertical extent) with its neighboring character (if any) occurring in the same enclosing zone, EZ (explained in Chapter 5). Tables 5.1 and 5.2 are used to resolve such confusions.

On the other hand, wrong occurrence of a ‘l’ (i.e. ℓ) in say, ‘3.14’ or ‘1’ in say, ‘log’ are spotted and corrected by checking the neighboring characters. A set of rules similar to the ones used in [69, 70] is implemented to correct many of these substitution errors.

Rejection errors, in general, do not create problems for the analysis of symbol arrangement. But when a bracket symbol (e.g. ‘(’, ‘)’, ‘[’, ‘]’, etc.) is rejected, the final TEX string shows *missing bracket* error on its compilation. On receiving such error message, further attempt is made to recognize a rejected character as one of the *bracket* symbol.

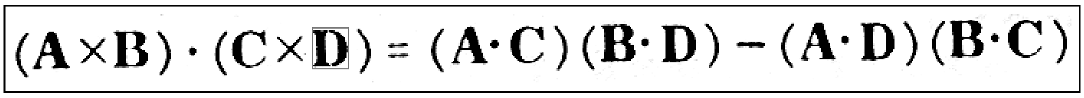

z = 1 + cos 2θ + i sin 2θ
'o' is recognized as '0' and this error leads to an invalid TeX string: \c0s

(a)



The character ‘r’ (marked by the rectangular box) is broken into two parts which are recognized as ‘γ’ and ‘*dot*’. During analysis of symbol arrangement, the *dot* is ignored as its position does not convey any meaning.

(b)



The character ‘D’ (marked by the rectangular box) is broken into two parts which are recognized as ‘I’ and ‘)’. Occurrence of an unmatched ‘)’ leads to TEX syntactic error.

(c)

Figure 7.2: Symbol Recognition error: (a) Generation of invalid TEX string, (b) and (c) Influence of broken characters.

On the other hand, broken characters often impose insertion type errors. In a few cases, the parsing algorithm does not find any suitable production rule to understand the positional meaning of a symbol originated from a broken character and therefore, ignores it. Figure 7.2(b) shows an example where the character ‘r’ is broken into two parts that are recognized as γ and *dot*. In the structure analysis phase, the position of the *dot* is not properly interpreted and hence, ignored. In such an occasion, the final TEX string does not show any syntactic error. However, there are cases when a broken character generates syntactically invalid TeX string. Figure 7.2(c) shows such an example where the character 'D' is broken into two parts, which are recognized as 'I' and ')'. Occurrence of ')' leads to *unmatched parenthesis* problem in the final TeX string. A prior knowledge about the nature of the broken characters may help to remove such errors.

7.2.3 Structure Interpretation Errors

Interpretation of structures being the last stage of expression recognition, errors in earlier modules affect this stage. Moreover, some errors may originate at this stage itself. Therefore, errors encountered during analysis of symbol arrangement are of two types: (i) errors that propagate from previous processing stages namely, segmentation errors and recognition errors and (ii) errors originated from the limitation of the structure analysis technique itself.

Segmentation and recognition errors that affect the interpretation of expression structures are already explained. A few cases are demonstrated in figures 7.1 and 7.2. Many of such errors explained earlier generate syntactically invalid TeX strings and therefore, detected by compiling the final TeX string corresponding to the input expression. Some of these errors are corrected using prior knowledge about the confusing and broken characters.

Errors that are not propagated from any earlier stage and originated during structure analysis only have been analyzed in sections 5.4 and 6.5.3. Main reason for this behavior is attributed to error in detection of symbol level, which leads to error in placement of that symbol in the final expression string.

In our system, a few of the symbol placement errors are corrected by maintaining a set of rules that suspect certain interpretations of symbol arrangement. Lee and Wang [69, 70] pointed out a few such rules like: *a numeral cannot have subscript expression*, etc. In our approach, we extended this set of rules (in total, 26 such rules are there in our system) to tackle some more errors that occur often in recognizing expressions.

For example, consider a case where the arrangement of symbols in an expression fragment, $C^{2^{n_7}}$ is interpreted as $C^{2^{n}7}$ because the system fails to understand '7' as a subscript of 'n'. However, such errors can be detected by maintaining a rule that would suspect the formation of any sub-expression of the form: $\left\langle \text{digit}^{\text{variable}}\text{digit} \right\rangle$. Once the formation of a sub-expression is suspected, the system re-checks its confidence to analyze the arrangement of symbols appearing in that sub-expression. However, since there is no rule to reject (rules only suspect) any interpretation obtained by analyzing the physical layout of a groups of symbols, the system retains the incorrect interpretation, if it fails to locate errors in its second attempt.

7.2.4 Improvement in Overall Recognition Accuracy

As mentioned, our system can do little to detect and correct the segmentation errors. However, several recognition and structure interpretation errors, as explained above, are detected and corrective measures for these errors are also attempted. It is experimentally observed that symbol recognition accuracy is improved by 0.21% and 0.36% for printed and handwritten expressions, respectively. Similarly, the number of correct recognition is increased from 4,348 to 4,376 when recognition of whole expression (printed) is concerned. Details of the improvement achieved due to the proposed error handling routine are outlined below in Table 7.1.

Table 7.1: Improvement due to Error Correction

	Symbol Recognition		Expression Recognition	
	Printed	Handwritten	Printed	Handwritten
#Test Samples	49,180	23,074	5,560	5,500
#Correct Recognitions	48,555	21,636	4,348	4,121
Accuracy	98.73%	93.77%	78.20%	74.93%
#Errors	625	1,438	1,212	1,379
#Error Corrections	104	84	28	29
Improved Accuracy	98.94%	94.13%	78.71%	75.45%

7.3 Performance Evaluation

In this section, we present a new performance-index (γ) that uses geometric (or structural) complexity of an expression to measure the overall performance. As explained in Chapter 2, the structural complexity of an expression is defined by (`GC`) i.e. the number of horizontal lines on which constituent symbols are arranged. Moreover, we view that an error in recognizing a base level (more clearly, dominant baseline [113]) structure would be more severe than the error in recognizing structures at higher levels. This is because symbols placed in horizontal lines other than the baseline are structurally dependent on the base level symbols. Therefore, error encountered in placement of a base level symbol (say, s_0) affects the placement of other symbols structurally related to s_0. In general, placement errors for base level symbols affects regeneration of an expression poorly than errors for other symbols. This situation will become more clear when the following examples are considered.

$$a + b + c = 2 + \alpha \tag{7.1}$$

$$a^2 + b + c = 2 + \alpha_1 \tag{7.2}$$

$$a^2 + b^2 + c^{2^{n_7}} = 2 + \alpha_1 \tag{7.3}$$

$$a^2 + b^2 + c^{2^{n_7}} = \frac{2 + \alpha_1}{\beta^2} \tag{7.4}$$

All the structures in equation 7.1 are in one level and hence, `GC` = 1. On the other hand, the expression in equation 7.2 shows a complexity (`GC = 3`) because a has one level superscript and α has one level subscript. Following this logic, GC-values of equations 7.3 and 7.4 are gradually increasing. One can visualize that error in interpreting position of any base level symbol (for example, 'a' in equation 7.2 or 'c' in equation 7.3) adversely affects the layout of other immediate symbols nested on it. Moreover, as use of non-base level symbols (e.g. script or limit expressions) increase the structural complexity of an expression, any systematic evaluation strategy is expected to consider how a system can recognize simple expressions and then to check the system's response as the complexity increases. Therefore, to evaluate the efficiency of an approach that deals with recognition of expressions, one has to take the geometric complexity of expressions into account. Four expressions in equation 7.1 through 7.4 show a gradual increment in structural complexity of the expressions.

7.3.1 Integrated Performance Measure

In our approach, the same method is followed for evaluating recognition of printed as well as handwritten expressions. The recognition result for an expression (printed or handwritten) is compared with the groundtruth corresponding to that expression. If they do not match (matching procedure is explained later in section 7.3.2), then the result is not correct. Errors originate from two sources, namely, (i) symbol recognition errors and (ii) errors in structure interpretation. Symbol recognition errors is easily computed as

$$\frac{\text{No. of wrongly recognized symbols}}{\text{Total no. of symbols}} \tag{7.5}$$

However, the computation of structure recognition errors is not trivial. This is so because the parsing of an expression may not be fully correct, but some of its symbol arrangements may be interpreted properly, and the system should be given partial credit for it. In our method, the erroneous arrangement of a symbol (s) is penalized by a factor $\frac{1}{\mid i \mid +1}$, where i is the `level` of the symbol, s.

It may be noted that in case of computing structure recognition error only spatial arrangements are important. For example, no structure recognition error is reported if X^m is recognized as X^{rn}. This is so because such symbol classification errors are accounted for by computing symbol recognition accuracy. In the foregoing example, structure recognition error is detected only if identification of *superscript* structure fails.

Assuming a test set ($\mathcal{T}$) contains Z number of expressions, γ_k is computed for all $k = 1, 2, \cdots Z$ and to rate the overall system performance, an average γ_{avg} is computed as follows:

For any test expression, let S_t be the total number of symbols, S_e be number of symbols recognized incorrectly, R_i be the number of symbols in the ith level, and O_i be the number of ith-level symbols for which incorrect arrangement analysis is encountered. Now, the performance index (γ) is defined as

$$\gamma = 1 - \frac{S_e + \sum_i O_i \times \frac{1}{\mid i \mid + 1}}{S_t + \sum_i R_i \times \frac{1}{\mid i \mid + 1}} \,. \tag{7.6}$$

Assuming a test set ($\mathcal{T}$) contains Z expressions, γ_k is computed for all $k = 1, 2, \cdots, Z$, and to rate the overall system performance, an average γ_{avg} is computed as

$$\gamma_{avg} = \frac{1}{Z} \sum_k \gamma_i \,. \tag{7.7}$$

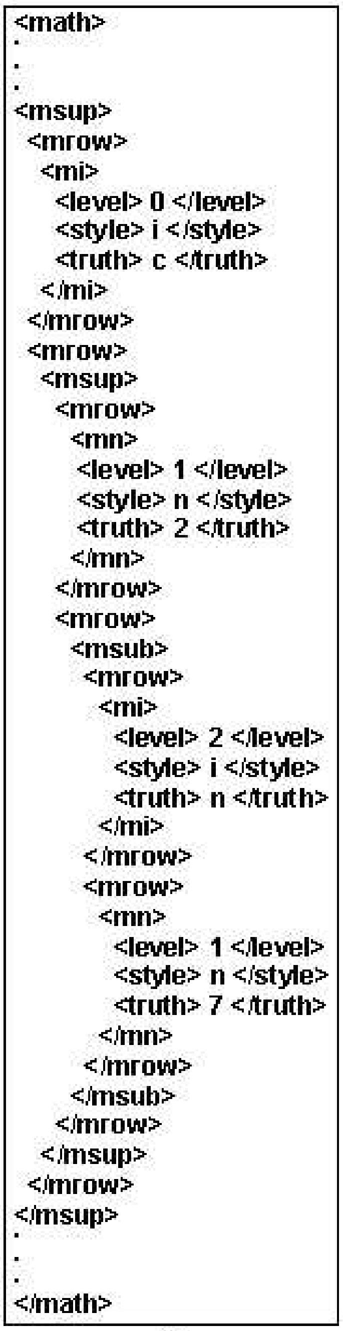
7.3.2 Evaluation Results

The performance of any expression recognition system can be judged following Eq. 7.6 and an average performance can be computed according to Eq. 7.7. For an input expression computation of γ needs detection of symbols for which symbol recognition or placements are wrong. This can be automatically done by comparing two Document Object Model[2] (DOM) trees, one generated from the groundtruth data for the expression and another generated from the recognition output for that expression. Let $\mathcal{G}_D$ be the DOM tree obtained from the groundtruthed data for an expression $\mathcal{E}$ and $\mathcal{R}_D$ be the DOM tree corresponds to the recognition result for $\mathcal{E}$. A comparison between $\mathcal{G}_D$ and $\mathcal{R}_D$ detects

[2]http://www.w3schools.com/dom/ and
for further reference see http://www.w3.org/TR/MathML2/chapter8.html

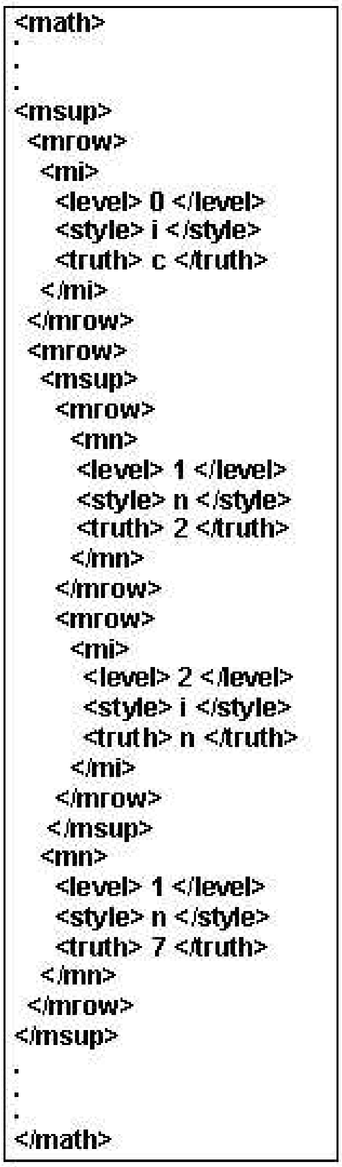
$$a^2 + b^2 + \boxed{c^{2^{n_7}}} = 2 + \alpha_1$$

(a)



```
<math>
.
.
.
<msup>
 <mrow>
  <mi>
   <level> 0 </level>
   <style> i </style>
   <truth> c </truth>
  </mi>
 </mrow>
 <mrow>
  <msup>
   <mrow>
    <mn>
     <level> 1 </level>
     <style> n </style>
     <truth> 2 </truth>
    </mn>
   </mrow>
   <mrow>
    <msub>
     <mrow>
      <mi>
       <level> 2 </level>
       <style> i </style>
       <truth> n </truth>
      </mi>
     </mrow>
     <mrow>
      <mn>
       <level> 1 </level>
       <style> n </style>
       <truth> 7 </truth>
      </mn>
     </mrow>
    </msub>
   </mrow>
  </msup>
 </mrow>
</msup>
.
.
.
</math>
```

(b)



```
<math>
.
.
.
<msup>
 <mrow>
  <mi>
   <level> 0 </level>
   <style> i </style>
   <truth> c </truth>
  </mi>
 </mrow>
 <mrow>
  <msup>
   <mrow>
    <mn>
     <level> 1 </level>
     <style> n </style>
     <truth> 2 </truth>
    </mn>
   </mrow>
   <mrow>
    <mi>
     <level> 2 </level>
     <style> i </style>
     <truth> n </truth>
    </mi>
   </mrow>
  </msup>
  <mn>
   <level> 1 </level>
   <style> n </style>
   <truth> 7 </truth>
  </mn>
 </mrow>
</msup>
.
.
.
</math>
```

(c)

Figure 7.3: Performance evaluation: (a) Image of an expression, (b) Groundtruthed data for the sub-expression marked in (a), and (c) Results obtained on recognition of the sub-expression.

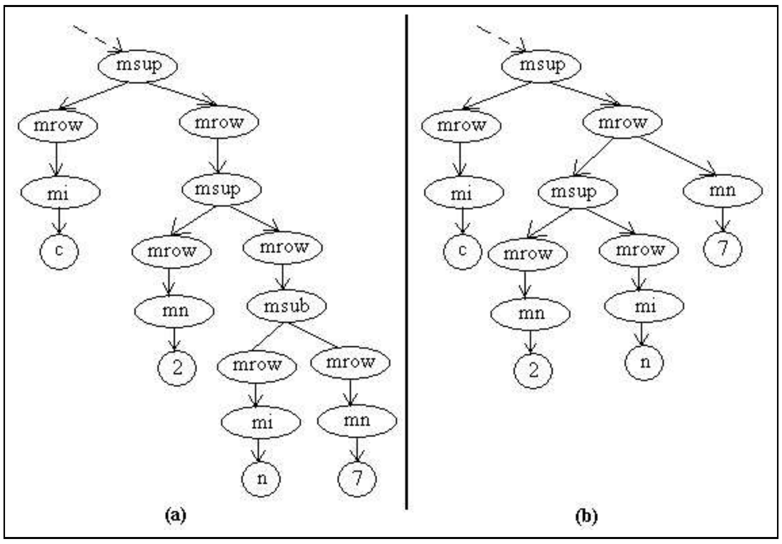


Figure 7.4: DOM representation: DOM trees correspond to (a) figure 7.3(b) and (b) figure 7.3(c).

the errors for recognition of $\mathcal{E}$.

Since matching of two trees is itself a long-standing subject of research[3], we at present ddo not explore much in this area (rather we treat this issue for our specific purpose as a future research problem). In our current approach, matching of two DOM trees is centred on the leaf-nodes only and parsing proceeds in a left to right order. At first, the left-most leaf node of $\mathcal{G}_D$ is picked up and corresponding leaf node in $\mathcal{R}_D$ is matched. Matching considers (i) identities (symbols) of the nodes and (ii) the paths found from the leaf-nodes to the root-nodes in two trees. A mis-match in the first case reports symbol a recognition error, whereas mis-match in the second case indicate symbol placement error.

Leaf-nodes generating mis-matches are marked in $\mathcal{G}_D$ as it represents the groundtruth. Next, manual intervention is invoked to compute the γ (i.e. Eq. 7.6) for $\mathcal{E}$. Manual intervention is required because the above DOM-matching approach identifies the symbols suffering from placement errors but at the same time it may mark certain other symbols which truly speaking do not suffer from placement errors. Actually, this matching method pinpoints the structures (i.e. group of symbols that impose a 2-D structure like scripts, etc.) for which the arrangement of some constituent symbols are incorrect.

Therefore, expressions (about 79% in printed environment and 75% for handwritten ones) that do not suffer from any error need not involve any manual input for computation

[3]We would like to refer to a recent article on this topic: Philip Bille, "Tree Edit Distance, Alignment Distance and Inclusion," available at "citeseer.ist.psu.edu/bille03tree.html".

of γ (i.e. Eq. 7.6) but others require manual effort for this purpose. In this sense, our method of performance evaluation is semi-automatic in nature and finding a fully automated way of doing this is considered as a future research problem.

Let us consider a small expression fragment to understand how evaluation is done following the above described approach. Figure 7.3(b) shows the groundtruth for a nested structure of the expression shown in Fig. 7.3(a). Figure 7.3(c) shows the recognition output when a system incorrectly recognizes the sub-expression $c^{2^{n_7}}$ in Fig. 7.3(a) as $c^{2^{n}7}$, which encounters an incorrect placement of "7". Such incorrect placements can be automatically detected by comparing the DOM trees shown in figure 7.4 where (a) and (b) correspond to the marked-up representations given in figure 7.3(b) and (c), respectively. Matching of the corresponding leaf-nodes of these two trees indicate that the leaf-nodes corresponding to 'n' in two trees do not match as the paths (from 'n' to the root node) vary. Similarly, the leaf-nodes corresponding to '7' also show mis-match. These two nodes are marked in the DOM representing the groundtruth.

As mentioned earlier that since the matching method used here captures the group of symbols imposing a 2-D structure, both 'n' and '7' (which bind themselves in a subscript structure) have been located. Next, manual effort determines which symbols that are really to be penalized. For example, in the above case, placement of 'n' is not incorrect and error originates due to placement of '7' and hence, arrangement of '7' is only to be penalized. To evaluate this recognition result, γ is computed following Eq. 7.6. In this case, $S_t = 4, S_e = 0, R_0 = 1, R_1 = 2, R_2 = 1, O_0 = 0, O_1 = 1, O_2 = 0$. Placement for only one symbol is incorrect at level 1. Therefore, $\gamma = 1 - \frac{1/2}{4+\sum 1+2/2+1/3} = 0.921$.

Tables 7.2 and 7.3 report evaluation results on our dataset of printed expressions and handwritten expressions, respectively. The figures in these tables include the marginal improvement in accuracy due to error correction. Moreover, it is to be noted that the results presented in these two tables assume that the groundtruthed data is free from errors. Also, this has been assumed for other experimental results presented in previous chapters. However, as discussed in Chapter 2, since generation of the groundtruthed data required substantial manual intervention, truthed data is not free from errors. Therefore, the results presented may undergo slight changes once the truthed data assures an error-free representation.

Looking at the Tables 7.2 and 7.3, it may be noted that the recognition efficiency of our system degrades as the structural complexity of the test expressions increases. This is due to the reason that as the geometric complexity of an expression increases detection of the symbol levels as well as their proper placement becomes difficult.

Table 7.2: Performance Evaluation: Recognition of Printed Expressions

Complexity (GC)	#Exp.	Correct Recognition	Performance (Avg. γ)
1	1,042	987 (94.72%)	0.977
2	1,987	1,648 (82.94%)	0.982
3	1,109	839 (75.65%)	0.954
4	801	571 (71.29%)	0.947
5	162	96 (59.26%)	0.903
6	202	117 (57.92%)	0.924
7	93	62 (66.67%)	0.906
8	70	31 (44.44%)	0.927
9	31	8 (25.81%)	0.891
10	16	6 (37.50%)	0.866
11	23	7 (30.43%)	0.876
12	12	3 (25.00%)	0.813
13	7	1 (14.29%)	0.780
14	3	0 (00.00%)	0.715
15	2	0 (00.00%)	0.738
Summary	5,560	4,376 (78.71%)	0.961

Table 7.3: Performance Evaluation: Recognition of Handwritten Expressions

Complexity (GC)	#Exp.	#Samples	Correct Recog.	Performance (Avg. γ)
1	25	780	703 (90.13%)	0.973
2	36	1,200	1,004 (83.67%)	0.962
3	44	1,500	1,163 (77.53%)	0.938
4	31	1,040	719 (69.13%)	0.915
5	8	220	156 (70.91%)	0.905
6	14	400	251 (62.75%)	0.884
7	7	160	91 (56.87%)	0.902
8	5	100	39 (39.00%)	0.877
9	2	40	10 (25.00%)	0.881
10	2	40	8 (20.00%)	0.850
11	1	20	6 (30.00%)	0.883
Summary	175	5,500	4,150 (75.45%)	0.935

7.4 Summary

This chapter deals with some general aspects that are common to both printed and online environment. An error detection and correction approach has been designed to improve the overall expression recognition results. Different types of errors that occur during recognition of expressions are studied in detail.

As the method for interpretation of expression structure progresses in a bottom-up manner, the entire expression is reconstructed by forming (and merging) of smaller symbol groups (sub-expressions). At any stage of this process, whenever a $\TeX$ string is generated for a smaller sub-expression, it is compiled and checked for syntactic validity. Any syntactic failure locates error points and calls for immediate processing to attempt error correction. In case the system fails to correct a detected error, it is left for manual correction at a later stage.

Several errors are corrected using a set of rules that exploit different contextual information. Symbol recognition errors originating due to shape similarity among symbols are corrected by using a character confusion matrix. The knowledge about broken characters is also used to correct some errors due to broken characters. Because of the proposed error handling routine, an overall improvement in recognition efficiency is also observed. At present, our proposed error handling routine does not consider semantic errors (e.g. $1 + 1 = 3$, etc.). However, detection and correction of such errors will be helpful for applications like electronic chalkboard, pen-based calculator, etc.

A new technique is proposed to evaluate efficiency of an expression recognition system. In the proposed performance evaluation strategy, computation of structure recognition result concerns only with the spatial arrangement of symbols and does not count the errors that are already accounted for computing symbol recognition accuracy. However, evaluation of structure recognition technique exploits geometric (or structural) complexity of an expression that is defined by the number of horizontal lines on which the expression symbols are arranged. It is viewed that the error in recognizing a base level (i.e. dominant baseline of an expression) structure would be more severe than the error in recognizing structures at higher levels.

The evaluation technique presented in this chapter considers all these aspects to formulate a performance index that integrates symbol recognition scores with structure recognition results and produces a single figure of merit to judge the overall recognition performance. Experimental results are outlined to demonstrate the efficiency of our proposed system.

CHAPTER 8
CONCLUSION

The motivation behind the present thesis was to provide a realistic computational approach for recognition of printed and handwritten mathematical expressions. Recognition of printed expressions is an essential requirement for the OCR of scientific paper documents. On the other hand, recognition of online handwritten expressions provides a convenient tool for entering mathematics into digital documents. The goal was set at:

- *Development of a representative corpus of printed and handwritten mathematical expressions*: A representative database is required to facilitate a systematic research on automatic recognition of expressions. Moreover, unavailability of suitable corpora of expressions has so far prompted the researchers to define their own dataset for testing their algorithm. As a result, replication of experiments and comparison of performance among different methods have become difficult tasks.

- *Finding mathematical expressions contained in scientific paper documents*: Such a technique helps to successfully upgrade the existing OCR systems (not trained for expression recognition) for converting scientific paper documents into their electronic form. Identification and extraction of expressions keeps an existing OCR system undisturbed while processing documents containing expressions. This is because once expression zones are located, a specially designed module can work for recognition of the expressions.

- *Robust recognition of expression symbols*: As recognition of expressions involves two stages: (i) symbol recognition and (ii) structure analysis, errors in recognizing expression symbols affect the module designed for structure analysis and thereby, the overall error rate in the final recognition of expressions is increased by manifold. Therefore, design of algorithms giving high accuracy for recognizing expression symbols (printed as well as handwritten) were planned.

- *An efficient parsing technique*: Method for analyzing physical layout of expression symbols must be efficient with respect to its accuracy and computational speed to realize practical systems. Moreover, the parsing technique must be general in nature to understand various types of expressions appear in different branches of science.

- *Error detection and correction*: Automatic detection and correction of errors occurring in different stages of recognition helps to reduce manual intervention to obtain error free output. Even if the system fails to correct certain errors, its ability to detect those errors is helpful at a later stage when human interaction is involved to verify and correct the recognition results.

- *Performance evaluation*: Performance evaluation of a system dealing with recognition of expressions is not straight-forward because mathematical expressions basically represent a visual language. Moreover, such a recognition system consists of a number of tightly coupled modules. Therefore, an effective evaluation strategy is needed to understand the capability of an expression recognition approach. Moreover, the evaluation strategy must be general in nature to compare various approaches on recognition of expressions.

Goals Achieved:

- An elaborate survey of the previous studies on recognition of expressions has been presented. Since recognition of expressions involve different processing components, qualitative comparisons of different methods have been presented under respective chapters presenting discussion on the related topic. This review work will be very much useful for future references.

- A corpus of 400 printed scientific documents containing about 5,560 expressions has been development. The documents are collected from various braches of science to make the corpus a representative one. The statistical analysis of the corpus content will facilitate future research on OCR of printed expressions. A database of handwritten (online) expressions is also constructed. This database contains about 5,500 samples for 175 expressions and the samples are collected from 40 writers. This database is a good source to study several aspects like variation in symbol shape, writing style, etc., which are important for recognition of handwritten expressions. For both printed and handwritten expressions a user-friendly marked-up representation has been proposed to groundtruth the expressions.

- A framework based on multifactorial analysis has been proposed to integrate several aspects contributing to a decision making problem. This framework has been used to solve two different problems namely, extraction of expression zones from document images and segmentation of touching characters. Promising results obtained for both the problems strongly attest the application potential of the proposed framework. Using multifactorial analysis based technique, 98.29% displayed (93.65% embedded) expressions are correctly located in test documents. On the other hand, about 96% touching character images are properly segmented by the proposed approach.

- A multiple-classifier approach has been proposed for recognition of expression symbols. Different techniques for combination of classifiers have been discussed. In case of printed expressions, four classifiers are combined whereas, two classifiers are used for recognition of online handwritten symbols. The proposed approach shows recognition accuracy of 98.73% for recognition of printed symbols (93.77% for handwritten symbols). Accuracies are further improved when for an input symbol, more than one choice (i.e. other than the best one) returned by the recognition engine is considered.

- Method proposed for understanding the physical layout of an expression is simple as far computational aspects are concerned. A context-free grammar has been formulated to analyze the arrangement of expression symbols. Several geometrical aspects are considered for this purpose. Recognition of handwritten expression exploits the spatio-temporal information available under online environment. If recognition of individual structures (i.e. *superscript*, *subscript*, *fraction*, etc.) are considered, the proposed technique is able to understand 93.82% structures for printed expressions (92.87% for handwritten expressions).

- Errors encountered in different modules are analyzed in details and a set of error-correcting rules is formulated. The design of rules exploits several contextual information to improve the overall expression recognition accuracy for both the printed and handwritten expressions. The proposed error handling approach is able to improve symbol recognition accuracy by 0.21% (from 98.73% to 98.94%) for printed expressions and for handwritten expressions, the degree of improvement for recognition of symbols is about 0.36% (from 93.77% to 94.13%). If the recognition of the whole expression is concerned, the error correcting approach shows an improvement of 0.51% (from 78.20% to 78.71%) for recognition printed expressions and for handwritten environment, this improvement is about 0.52% (from 74.93% to 75.45%).

- Performance evaluation has been studied with special emphasis. Performance of each processing module has been analyzed in-depth. Methods have been proposed to compute efficiency for extraction of expression zones, analysis of symbol arrangement in expressions. Finally, a new method for evaluating performance of an expression recognition system has been presented. The proposed performance measure considers several non-trivial issues related to an expression recognition task and provides a single figure of merit to judge the efficiency of a system. A semi-automatic evaluation of the system performance has been demonstrated using the proposed format for groundtruthing of the expressions.

Scope of Future Research:

The study presented here can be extended in several directions. Some of them are highlighted below:

- *Dissemination of the corpus*: Distribution of the corpus described in Chapter 2 is considered as one of our immediate future activities related to the present work. Apart from us, so far no other peer research groups have used this corpus for their research. Very recently, two other Indian groups working in the area of OCR have taken a part of this corpus but they haven't published any results yet. We plan to upload the full corpus (or a part of it in case our institute[1] decides to charge anything to use the corpus) in the Internet so that other interested groups can easily avail the data. This will be done after we finish the second/final level of validation check. A few samples are already available in the net at `www.isical.ac.in/∼utpal` (under *Resources*). Researchers are being invited to post their comments and suggestions in this regard. As we are yet to start the final-level validation check, some suggestions that recommend slight but useful modifications in the groundtruth format (or in some other aspects) may be incorporated during the execution of the final phase.

- *Integration with existing OCR systems*: We would like to integrate the proposed method for recognition of printed expression with an existing OCR system to test the performance of the combined system in recognizing scientific paper documents. For this purpose, we plan to approach some of the software companies who are already in the OCR-related business. Integration of our proposed system with an

[1]As our institute has partially supported the generation of this corpus, final decision on whether the dissemination of the corpus will be free-of-cost or not will be taken by the institute's authority.

existing OCR system needs some modifications but we understand that most of these modifications will be required at the interface level only and won't affect the existing system.

- *New methods and designs*: Studies on analysis of error patterns occurring in expression recognition are few in number. Some attempts have been made in this thesis to design an error handling component but it is experienced that more research is need in this direction. Moreover, performance evaluation of an expression recognition system has room for further investigation. Design of new performance evaluation strategy or modification of the one presented in this thesis can be considered in future work. Moreover, a semi-automatic evaluation (of the performance index, γ in (7.6)) has been proposed and a fully automated evaluation strategy needs further research.

- *Pen-based computing facility*: The module that recognizes online handwritten expressions, if integrated, may provide a batter man-machine interface for computer algebra systems [75]. Other applications like pen-based calculator [16] or Math tutoring systems for children could be explored.

- *Reading aid for the blind*: It would be advantageous for visually impaired people if the recognized expressions were converted into corresponding speech form [91, 48]. Though finding an unambiguous way of converting expressions into speech is quite difficult but if achieved, will be of immense help for the blind people to read scientific documents.

REFERENCES

[1] A.V. Aho, R. Sethi and J.D. Ullman, "Compilers: Principles, Techniques, and Tools," published by *Addison-Wesley Publishing Co.,* 1986.

[2] R.H. Anderson, "Syntax-directed Recognition of Hand-printed Two-dimensional Mathematics," *Doctoral Dissertation*, Dep. of Engineering and Applied Physics, Harvard Univ., 1968.

[3] R.H. Anderson, "Two-dimensional Mathematical Notations," *Syntactic Pattern Recognition Applications*, (Ed. K.S. FU), Springer-Verlag, New York, pp. 147-177, 1977.

[4] A. Belaid and J. Haton, "A Syntactic Approach for Handwritten Mathematical Formula Recognition," *IEEE Trans. on Pattern Analysis and Machine Intelligence,* vol. 6, no. 1, pp. 105-111, 1984.

[5] E.J. Bellagarda, J.R. Bellagarda, D. Nahamoo, and N.S. Nathan, "A Probabilistic Framework for Online Handwriting Recognition," *Proc. of the 3rd Int'l. Workshop on Frontiers in Handwriting Recognition* (IWFHR), pp. 225-234, Buffalo, USA, 1993.

[6] B.P. Berman and R.J. Fateman, "Optical Character Recognition for Typeset Mathematics," *ACM Proc. of Int'l. Symposium on Symbolic and Algebraic Computation* (ISSAC), pp. 348-353, Oxford, UK, 1994.

[7] D. Black, "The Theory of Committees and Elections," *Cambridge University Press, London*, 2nd Ed., 1963.

[8] D. Blostein and A. Grbavec, "Recognition of Mathematical Notation," *Handbook of Character Recognition and Document Image Analysis*, Eds. H. Bunke and P.S.P. Wang, World Scientific Publishing Company, pp. 557-582, 1997.

[9] L. Bovino, G. Dimauro, S. Impedovo, M.G. Lucchese, R. Modugno, G. Pirlo, A. Salzo, and L. Sarcinella, "On the combination of abstract-level classifiers," *Int'l. J. on Document Analysis and Recognition* (IJDAR), vol. 6, no. 1, pp. 42-54, 2003.

[10] R.G. Casey and G. Nagy, "Recursive segmentation and classification of composite character patterns," *Proc. of the 6th Int'l. Conf. Pattern Recognition* (ICPR), pp. 1023-1026, Munich, Germany, 1982.

[11] R.G. Casey and E. Lecolinet, "A Survey of Methods and Strategies in Character Segmentation," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 18, no. 7, pp. 690-706, 1996.

[12] K-F. Chan and D-Y. Yeung, "Recognizing on-line handwritten alphanumeric characters through flexible structural matching," *Pattern Recognition,* vol. 32, no. 7, pp. 1099-1114, 1999.

[13] K-F. Chan and D-Y. Yeung, "Mathematical Expression Recognition: A Survey," *Int'l. J. on Document Analysis and Recognition* (IJDAR), vol. 3, no. 1, pp. 3-15, 2000.

[14] K-F. Chan and D-Y. Yeung, "An Efficient Syntactic Approach to Structural Analysis of On-line Handwritten Mathematical Expressions," *Pattern Recognition*, vol. 33, no. 3, pp. 375-384, 2000.

[15] K-F. Chan and D-Y. Yeung, "Error detection, error correction and performance evaluation in on-line mathematical expression recognition," *Pattern Recognition*, vol. 34, no. 8, pp. 1671-1684, 2001.

[16] K-F. Chan and D-Y. Yeung, "PenCalc: A Novel Application of On-Line Mathematical Expression Recognition Technology," *Proc. of the 6th Int'l. Conf. on Document Analysis and Recognition* (ICDAR), pp. 774-778, Seattle, USA, 2001.

[17] S-K. Chang, "A Method for the Structural Analysis of Two-Dimensional Mathematical Expressions," *Information Sciences*, vol. 2, pp. 253-272, 1970.

[18] E. Charniak, "Statistical Language Learning," *MIT Press*, 1993.

[19] B.B. Chaudhuri and U. Garain, "Automatic detection of italic, bold and all-capital words in document," *Proc. of the 14th Int'l Conf. on Pattern Recognition* (ICPR), pp. 610-612, Brisbane, Australia, 1998.

[20] B.B. Chaudhuri and U. Garain, "An Approach for Recognition and Interpretation of Mathematical Expressions in Printed Document," *Pattern Analysis and Applications* (PAA), vol. 3, pp. 120-131, 2000.

[21] B.B. Chaudhuri and U. Garain, "Extraction of type style based meta-information from imaged documents," *Int'l. J. on Document Analysis and Recognition* (IJDAR), vol. 3, no. 3, pp.138-149, March, 2001.

[22] H.L. Chen and P.Y. Yin, "A system for on-line recognition of handwritten mathematical expressions," *Computer Processing of Chinese and Oriental Languages,* vol. 6, no. 1, pp. 19-39, 1992.

[23] P.A. Chou, "Recognition of Equations Using a Two-Dimensional Stochastic Context-Free Grammar," *Proc. of the SPIE*, Visual Communication and Image Processing IV, vol. 1199, pp. 852-863, 1989.

[24] S.P. Chowdhury, S. Mandal, A.K. Das and B. Chanda, "Automated Segmentation of Math-Zones from Document Images," *Proc. of the 7th Int'l Conf. Document Analysis and Recognition* (ICDAR), pp. 755-759, Edinburgh, Scotland, 2003.

[25] S.D. Connell and A.K. Jain, "Template-based Online Character Recognition," *Pattern Recognition,* vol. 34, no. 1, pp. 1-14, 2001.

[26] D.R. Cox and E.J. Snell "Analysis of Binary Data," 2nd Ed., Burlington, UK, Chapman and Hall, 1989.

[27] I. Daubechies, "The Wavelet Transform, Time-frequency Localization and Signal Analysis," *IEEE Trans. on Information Theory,* vol. 36, no. 5, pp 961-1005, 1990.

[28] Y.A. Dimitriadis and J. L. Coronado, "Towards an ART based mathematical editor, that uses on-line handwritten symbol recognition," *Pattern Recognition,* vol. 28, no. 6, pp 807-822, 1995.

[29] D.G. Elliman and I.T. Lancaster, "A Review of Segmentation and Contextual Analysis Techniques for Text Recognition," *Pattern Recognition*, vol. 23, no. 3/4, pp. 337-346, 1990.

[30] Y. Eto and M. Suzuki, "Mathematical Formula Recognition Using Virtual Link Network," *Proc. of the 6th Int'l Conf. Document Analysis and Recognition* (ICDAR), pp. 762-767, Seattle, USA, 2001.

[31] R.J. Fateman and T. Tokuyasu, "Progress in recognizing typeset mathematics," *Proc. of the SPIE*, vol. 2660, pp. 37-50, San Jose, California, USA, 1996.

[32] R.J. Fateman, T. Tokuyasu, B.P. Berman and N. Mitchell, "Optical Character Recognition and Parsing of Typeset Mathematics," *J. of Visual Communication and Image Representation*, vol 7, no. 1, pp. 2-15, 1996.

[33] R.J. Fateman "How to find mathematics on a scanned page," *Proc. of the SPIE*, vol.3967, pp. 98-109, San Jose, California, USA, 1999.

[34] C. Faure and Z.X. Wang, "Automatic Perception of the Structure of Handwritten Mathematical Expressions," *Computer Processing of Handwriting*, Eds: R. Plamondon and C.G. Leedham, World Scientific, Singapore, pp. 337-361, 1990.

[35] G. D. Forney Jr., "The Viterbi Algorithm," *Proceedings of the IEEE,* vol. 61, no. 3, pp. 263-278, March, 1973.

[36] H. Freeman, "On the digital computer classification of geometric line patterns," *Proc. of National Electronics Conference,* vol. 18, pp. 312-324, 1962.

[37] H. Fujisawa, Y. Nakano, and K. Kurino, "Segmentation Methods for Character Recognition: From Segmentation to Document Structure Analysis," *Proceedings of the IEEE*, vol. 80, no. 7, pp. 1079-1092, 1992.

[38] R. Fukuda, F. Tamari, X. Ming, and M. Suzuki, "A Technique of Mathematical Expression Structure Analysis for the Handwriting Input System," *Proc. of the 5th Int'l Conf. Document Analysis and Recognition* (ICDAR), Bangalore, India, pp. 131-134, 1999.

[39] U. Garain and B.B. Chaudhuri, “Compound character recognition by run number based metric distance,” *Proc. IS&T/SPIE’s 10th Int. Symposium on Electronic Imaging: Science & Technology,* SPIE, vol. 3305, pp. 90-97, San Jose, California, USA, 1998.

[40] U. Garain, B.B. Chaudhuri, and T. Pal, “Online Handwritten Indian Script Recognition: A Human Motor Function based Framework,” *Proc. of the 16th Int’l. Conf. on Pattern Recognition* (ICPR), Quebec City, Canada, 2002.

[41] U. Garain and B.B. Chaudhuri, “Input of Handwritten Mathematical Expressions into machine coded Indian Language Documents,” *Proc. of the Indo European Conference on Multilingual Technologies* (IECMT), Eds: R. K. Arora, M. Kulkarni, and H. Darbari, Tata McGraw-Hill Publishing Company Limited (New Delhi), pp. 3-12, Pune, India, 2002.

[42] U. Garain and B.B. Chaudhuri, “Segmentation of Touching Characters in Printed Devnagari and Bangla Scripts using Fuzzy Multifactorial Analysis,” *IEEE Transactions on Systems, Man and Cybernetics, Part C,* vol. 32, no 4, pp. 449-459, 2002.

[43] U. Garain and B.B. Chaudhuri, “On Development and Statistical Analysis of a Corpus for Printed and Handwritten Mathematical Expressions,” In: *The 4th IAPR Int’l. Workshop on Graphics Recognition* (GREC), pp. 429-439, Canada, 2001.

[44] P. Garcia and B. Couasnon, “Using a Generic Document Recognition Method for Mathematical Formulae Recognition,” *Proc. of Int’l Workshop on Graphics Recognition* (GREC), LNCS, vol. 2390, pp. 236-244, Eds. D. Blostein and Y.-B. Kwon, Springer-Verlag, Berlin Heidelberg, 2002.

[45] A. Grbavec and D. Blostein, “Mathematics recognition using graph rewriting,” *Proc. of the 3rd Int’l. Conf. on Document Analysis and Recognition* (ICDAR), pp. 417-421, Montreal, Canada, 1995.

[46] I. Guyon, M. Schenkel, and J. Denker, “Overview and Synthesis of On-Line Cursive Handwriting Recognition Techniques”, *Handbook of Character Recognition and Document Image Analysis,* Eds. H. Bunke and P.S.P. Wang, World Scientific Publishing Company, pp. 183-225, 1997.

[47] J. Ha, R.M. Haralick and I.T. Phillips, “Understanding Mathematical Expressions From Document Images,” *Proc. of the 3rd Int’l. Conf. on Document Analysis and Recognition* (ICDAR), pp. 956-959, Montreal, Canada, 1995.

[48] B. Hayes, “Speaking of Mathematics,” *American Scientists,* vol. 84, no. 2, pp. 110-113, 1996.

[49] T.K. Ho, J.J. Hull, and S.N. Srihari, “Decision Combination in Multiple Classifier Systems,” *IEEE Trans. on Pattern Recognition and Machine Intelligence*, vol. 16, no. 1, pp. 66-75, 1994.

[50] J.F. Hull, "Recognition of Mathematics Using a Two-dimensional Trainable Context-free Grammar," *Master's thesis*, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 1996.

[51] D.P. Huttenlocher, G.A. Klandermann, and W.J. Rucklidge, "Comparing Images using the Hausdorff Distance," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 15, no. 9, pp. 850-863, 1993.

[52] K. Inoue, R. Miyazaki, and M. Suzuki, "Optical Recognition of Printed Mathematical Documents," *Proc. of Asian Technology Conference in Mathematics* (ATCM), Springer-Verlag, pp. 280-289, 1998.

[53] Institute of Electrical and Electronics Engineers (IEEE), "Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries," New York, 1990.

[54] A. K. Jain and B. Yu, "Document Representation and its Application to Page Decomposition," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 20, no. 3, pp. 294-308, 1998.

[55] J. Jin, X. Han and Q. Wang, "Mathematical Formulas Extraction," *Proc. of the 7th Int'l Conf. Document Analysis and Recognition* (ICDAR), Edinburgh, Scotland, pp. 1138-1141, 2003.

[56] B.H. Juang and L.R. Rabiner, "The segmental k-means algorithm for estimating the parameters of hidden Markov models," *IEEE Trans. on Accoust. Speech, Signal Processing,* vol. 38, no. 9, pp. 1639-1641, 1990.

[57] A. Kacem, A. Belaid and M. Ben Ahmed, "EXTRAFOR: automatic EXTRAction of mathematical FORmulas," *Proc. of the 5th Int'l Conf. Document Analysis and Recognition* (ICDAR), pp. 527-530, Bangalore, India, 1999.

[58] A. Kacem, A. Belaid and M. Ben Ahmed, "Automatic extraction of printed mathematical formulas using fuzzy logic and propagation of context," *Int'l. J. on Document Analysis and Recognition* (IJDAR), vol. 4, no. 2, pp. 97-108, 2001.

[59] M. Koschinski, H.-J. Winkler, and M. Lang, "Segmentation and Recognition of Symbols within handwritten Mathematical Expressions," *Proc. of IEEE Int'l. Conf. on Acoustics, Speech, and Signal Processing* (ICASSP), vol. 4, pp. 2439-2442, Detroit, USA, 1995.

[60] A. Kosmala and G. Rigoll, "Recognition of On-Line Handwritten Formulas," *Proc. of the 6th Int'l. Workshop on Frontiers in Handwriting Recognition* (IWFHR), pp. 219-228, Taejon, Korea, 1998.

[61] A. Kosmala and G. Rigoll, "On-Line Handwritten Formula Recognition using Statistical Methods," Proc. of the 14th Int'l. Conf. on Pattern Recognition (ICPR), pp. 1306-1308, Brisbane, Australia, 1998.

[62] A. Kosmala, G. Rigoll, S. Lavirotte, and L. Pottier, "On-Line Handwritten Formula Recognition using Hidden Markov Models and Context Dependent Graph Grammars," *Proc. of the 5th Int'l Conf. Document Analysis and Recognition* (ICDAR), pp. 107-110, Bangalore, India, 1999.

[63] V.A. Kovalevsky, "Character Readers and Pattern Recognition," *Spartan Books*, Washington, D.C., 1968.

[64] L. Lamport, "LaTeX - A Document Preparation System- User's Guide and Reference Manual," *Addison-Wesley*, Reading, MA, 1995.

[65] S. Lavirotte and L. Pottier, "Optical Formula Recognition," *Proc. of the 4th Int'l. Conf. on Document Analysis and Recognition* (ICDAR), pp. 357-361, Ulm, Germany, 1997.

[66] S. Lavirotte and L. Pottier, "Mathematical formula recognition using graph grammar," *Document Recognition V, the Int'l Society for Optical Engineering*, vol. 3305, pp. 44-52, 1998.

[67] H.J. Lee and M.C. Lee, "Understanding Mathematical Expressions in a Printed Document," *Proc. the 2nd Int'l Conf. on Document Analysis and Recognition* (ICDAR), pp. 502-505, Japan, 1993.

[68] H.J. Lee and M.C. Lee, "Understanding mathematical expressions using procedure-oriented transformation," *Pattern Recognition*, vol. 27, no. 3, pp. 447-457, 1994.

[69] H.J. Lee and J.-S. Wang, "Design of a Mathematical Expression Recognition System," *Proc. of the 3rd Int'l Conf. on Document Analysis and Recognition* (ICDAR), pp. 1084-1087, Montreal, Canada, 1995.

[70] H.J. Lee and J.-S. Wang, "Design of a Mathematical Expression Understanding System," *Pattern Recognition Letters*, vol. 18, no. 3, pp. 289-298, 1997.

[71] S. Lehmberg, H.J. Winkler and M.Lang, "A soft-decision approach for symbol segmentation within handwritten mathematical expressions," *Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing* (ICASSP), pp. 3434-3437, 1996.

[72] H.-X. Li, "Multifactorial Functions in Fuzzy Sets Theory," *Fuzzy Sets and Systems*, vol. 35, no. 1, pp. 69-84, 1990.

[73] H.X. Li and V.C. Yen, "Fuzzy sets and fuzzy decision-making," *CRC Press*, 1995, USA.

[74] W.A. Martin, "Computer input/output of mathematical expressions," *Proc. of Int'l Symposium on Symbolic Algebraic Manipulation*, pp. 78-89, Los Angeles, CA, 1971.

[75] R. Marzinkewitsch, "Operating Computer Algebra Systems by handprinted Input," *Proc. of Int'l Symposium on Symbolic Algebraic Computation*, pp. 411-413, Bonn, Germany, 1991.

[76] E.G. Miller and P.A. Viola, "Ambiguity and Constraint in Mathematical Expression Recognition," *Proc. of the National Conf. of Artificial Intelligence*, American Association of Artificial Intelligence, pp. 784-791, Madison, Wisconsin, 1998.

[77] J. Mitra, U. Garain, B.B. Chaudhuri, H.V.K. Swamy, and T. Pal, "Automatic Understanding of Structures in Printed Mathematical Expressions," *Proc. of the 7th Int'l Conf. Document Analysis and Recognition* (ICDAR), pp. 540-544, Edinburgh, Scotland, 2003.

[78] S. Mori, C.Y. Suen, and K. Yamamoto, "Historical Review of OCR Research and Development," *Proceedings of the IEEE*, vol. 80, no. 7, pp. 1029-1058, 1992.

[79] A. Nomura, K. Michishita, S. Uchida, and M. Suzuki, "Detection and Segmentation of Touching Characters in Mathematical Expressions," *Proc. of the 7th Int'l Conf. Document Analysis and Recognition* (ICDAR), pp. 126-130, Edinburgh, Scotland, 2003.

[80] S. Nouzumi, K. Inoue, R. Miyazaki, and M. Suzuki, "Optical Recognition System of Printed Japanese Mathematical Documents," *Proc. of the 3rd IAPR Workshop on Document Analysis Systems* (DAS) pp. 197-200, Nagano, Japan, 1998.

[81] M. Okamoto and B. Miao, "Recognition of mathematical expressions by using the layout structure of symbols," *Proc. of the 1st Int'l Conf. Document Analysis and Recognition* (ICDAR), vol. 1, pp. 242-250, Saint Malo, France, 1991.

[82] M. Okamoto and A. Miyazawa, "An Experimental Implementation of Document Recognition System for Papers Containing Mathematical Expressions," *Structured Document Image Analysis*, (Eds. Baird, Bunke, Yamamoto), Springer Verlag, pp. 36-53, 1992.

[83] M. Okamoto, S. Sakaguchi, and T. Suzuki, "Segmentation of touching characters in formulae," *Proc. of the 3rd IAPR Workshop on Document Analysis Systems* (DAS), pp. 283-289, Nagano, Japan, 1998.

[84] M. Okamoto, H. Imai and K. Takagi, "Performance Evaluation of a Robust Method for Mathematical Expression Recognition," *Proc. of the 6th Int'l Conf. Document Analysis and Recognition* (ICDAR), pp. 121-128, Seattle, USA, 2001.

[85] H. Okamura, T. Kanahori, W. Cong, R. Fukuda, F. Tamari, and M. Suzuki, "Handwriting Interface for Computer Algebra Systems," *Proc. of the 4th Asian Technology Conference in Mathematics* (ATCM), pp. 291-300, Guangzhou, China, 1999.

[86] T. Pavlidis and J. Zhou, "Page segmentation and classification," *Computer Vision, Graphics, and Image Processing* (CVGIP), vol. 54, pp. 484-496, 1992.

[87] I. Phillips, "Methodologies for using UW Databases for OCR and Image Understanding Systems," *Document Recognition V, Proc. of the SPIE*, vol. 3305, pp. 112-127, San Jose, CA, USA, 1998.

[88] I. Phillips and A. Chhabra, "Empirical Performance Evaluation of Graphics Recognition Systems," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 21, no. 9, pp. 849-870, 1999.

[89] R. Plamondon and S.N. Srihari, "On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey," *IEEE Trans. on Pattern Analysis and Machine Intelligence,* vol, 22, no. 1, pp. 63-84, 2000.

[90] L.R. Rabiner and B.H. Juang, "An introduction to hidden Markov models," *IEEE Trans. on Accoust. Speech, Signal Processing (ASSP),* vol. 3, no. 1, pp. 4-16, June 1986.

[91] T.V. Raman, "Audio System for Technical Readings," *Doctoral Dissertation*, Cornell University, USA, 1994.

[92] J. Rocha and T. Pavlidis, "A Shape Analysis Model with Applications to a Character Recognition System," *IEEE Trans. on Pattern Recognition and Machine Intelligence*, vol. 16, no. 4, pp. 393-404, 1994.

[93] Y. Sakamoto, M. Xie, R. Fukuda, and M. Suzuki, "On-Line Recognition of Handwriting Mathematical Expression via Network," *Proc. of Asian Technology Conference in Mathematics* (ATCM), pp. 271-279, Tsukuba, Japan, 1998.

[94] S. Smithies, K. Novins, and J. Arvo, "A handwriting-based equation editor," *Proc. of Graphics Interface,* pp. 84-91, Kingston, Ontario, Canada, 1999.

[95] M. Suzuki, F. Tamari, R. Fukuda, S. Uchida, and T. Kanahori, "INFTY – An Integrated OCR System for Mathematical Documents," *Proc. of ACM Symposium on Document Engineering* (DocEng), pp. 95-104, Grenoble, France, 2003.

[96] E. Tapia and R. Rojas, "Recognition of On-line Handwritten Mathematical Formulas in the E-Chalk System," *Proc. of the 7th Int'l. Conf. on Document Analysis and Recognition* (ICDAR), pp. 980-984, Edinburgh, Scotland, 2003.

[97] C.C. Tappert, C.Y. Suen, and T. Wakahara, "The State of the Art in On-Line Handwriting Recognition," *IEEE Trans. on Pattern Analysis and Machine Intelligence,* vol. 12, no. 8, pp. 179-190, 1990.

[98] J.-Y. Toumit and H. Emptoz, "A character matching method for mathematical object detection," *Proc. of RECPAD*, pp. 83-90, Lisbon, Portugal, 1998.

[99] J.-Y. Toumit and H. Emptoz, "From the segmentation to the reading of a mathematical document," *Proc. of Conf. on Machine Graphics and Vision*, pp. 483-504, Borki, Poland, 1998.

[100] J.-Y. Toumit, S. Garcia-Salicetti, and H. Emptoz, "A Hierarchical and Recursive Model of Mathematical Expressions for Automatic Reading of Mathematical Documents," *Proc. of the 5th Int'l Conf. Document Analysis and Recognition* (ICDAR), pp. 119-122, Bangalore, India, 1999.

[101] K. Toyozumi, T. Suzuki, K. Mori, and Y. Suenaga, "A System for Real-time Recognition of Handwritten Mathematical Formulas," *Proc. of the 6th Int'l Conf. Document Analysis and Recognition* (ICDAR), pp. 1059-1063, Seattle, USA, 2001.

[102] S. Tsujimoto and H. Asada, "Major Components of a Complete Text Reading System," *Proceedings of the IEEE*, vol. 80, no. 7, pp. 1133-1149, 1992.

[103] H.M. Twaakyondo and M. Okamoto, "Structure Analysis and Recognition of Mathematical Expressions," *Proc. of the 3rd Int'l. Conf. on Document Analysis and Recognition* (ICDAR), pp. 430-437, Montreal, Canada, 1995.

[104] Z.X. Wang and C. Faure, "Structural analysis of handwritten mathematical expressions," *Proc. of the 9th Int'l. Conf. on Pattern Recognition* (ICPR), pp. 32-34, Rome, Italy, 1988.

[105] P.-Z. Wang and M. Sugeno, "The factor fields and background structure for fuzzy subsets", *Fuzzy Mathematics*, vol. 2, no. 2, pp. 45-54, 1982.

[106] P.-Z. Wang, "A factor space approach to knowledge representation", *Fuzzy Sets and Systems*, vol. 36, pp. 113-124, 1990.

[107] H.J. Winkler, H. Fahrner, and M. Lang, "A Soft-Decision Approach for Structural Analysis of Handwritten Mathematical Expressions," *Proc. of IEEE Int'l. Conf. on Acoustics, Speech, and Signal Processing* (ICASSP), vol. 4, pp. 2459-2462, Detroit, USA, 1995.

[108] H.J. Winkler and M. Lang, "Symbol segmentation and recognition for understanding handwritten mathematical expressions," *Progress in Handwriting Recognition*, Eds. A. Downton and S. Impedovo, pp. 407-412, World Scientific, Singapore, 1997.

[109] H.J. Winkler and M. Lang, "On-line symbol segmentation and recognition in handwritten mathematical expressions," *Proc. of IEEE Int'l. Conf. on Acoustics, Speech, and Signal Processing* (ICASSP), pp. 3377-3380, Munich. Germany, 1997.

[110] Z. Xuejun, L. Xinyu, Z. Shengling, P. Baochang, and Y. Tang, "On-Line Recognition of Handwritten Mathematical Symbols," *Proc. of the 4th Int'l. Conf. on Document Analysis and Recognition* (ICDAR), pp. 645-648, Ulm, Germany, 1997.

[111] R. Zanibbi, K. Novins, J. Arvo, and K. Zanibbi, "Aiding Manipulation of Handwritten Mathematical Expressions through Style-Preserving Morphs," *Proc. of Conf. on Graphics Interface,* pp. 127-134, Ottawa, Ontario, Canada, 2001.

[112] R. Zanibbi, D. Blostein and J.R. Cordy, "Baseline Structure Analysis of Handwritten Mathematics Notation," *Proc. of the 6th Int'l. Conf. on Document Analysis and Recognition* (ICDAR), pp. 768-773, Seattle, Washington, 2001.

[113] R. Zanibbi, D. Blostein and J.R. Cordy, "Recognizing Mathematical Expressions Using Tree Transformation," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 24, no. 11, pp. 1455-1467, 2002.

[114] X. Zhao, X. Liu, S. Zheng, B. Pan, and Y.Y. Tang, "On-line Recognition of Handwritten Mathematical Symbols," *Proc. of the 4th Int'l. Conf. on Document Analysis and Recognition* (ICDAR), pp. 645-648, Ulm, Germany, 1997.

[115] K.G. Zipf, "Human Behavior and the Principal of Least Effort, an Introduction to Human Ecology," *Addison-Wesley,* Reading, Mass., 1949.

List of publications related to the thesis

1 B.B. Chaudhuri and U. Garain, "An approach for processing mathematical expressions in printed document," *Document Analysis Systems: Theory and Practice,* LNCS 1655, Eds: Seong-Whan Lee, Y. Nakano, Springer, pp. 310-321, 1998.

2 U. Garain and B.B. Chaudhuri, "An approach for processing mathematical expressions in printed document," In *IAPR workshop on Document Analysis Systems* (DAS), pp. 376-385, Nagano, Japan, 1998.

3 B.B. Chaudhuri and U. Garain, "Automatic detection of italic, bold and all-capital words in document," In: *Proc. of the 14th Int'l. Conf. on Pattern Recognition* (ICPR), pp. 610-612, Brisbane, Australia, 1998.

4 U. Garain and B.B. Chaudhuri, "Compound character recognition by run number based metric distance," In: *Proc. IS&T/SPIE's 10th Int. Symposium on Electronic Imaging : Science & Technology,* SPIE vol. 3305, pp. 90-97, San Jose, California, USA, 1998.

5 B.B. Chaudhuri and U. Garain, "An Approach for Recognition and Interpretation of Mathematical Expressions in Printed Document," *Pattern Analysis and Applications* (PAA), vol. 3, pp. 120-131, 2000.

6 U. Garain and B.B. Chaudhuri, "A Syntactic Approach for Processing Mathematical Expressions in Printed Documents," In: *Proc. of the 15th Int'l. Conf. on Pattern Recognition* (ICPR), vol. 4, pp. 523-526, Barcelona, Spain, 2000.

7 B.B. Chaudhuri and U. Garain, "Extraction of type style based meta-information from imaged documents," *Int'l. Journal on Document Analysis and Recognition* (IJDAR), vol. 3, no. 3, pp.138-149, 2001.

8 U. Garain and B.B. Chaudhuri, "On Development and Statistical Analysis of a Corpus for Printed and Handwritten Mathematical Expressions," In: *The 4th IAPR Int'l. Workshop on Graphics Recognition* (GREC), pp. 429-439, Canada, 2001.

9 U. Garain and B.B. Chaudhuri, "Input of Handwritten Mathematical Expressions into machine coded Indian Language Documents," In: *The Indo European Conference on Multilingual Technologies* (IECMT), Eds: R. K. Arora, M. Kulkarni, and H. Darbari, Tata McGraw-Hill Publishing Company Limited (New Delhi), pp. 3-12, Pune, India, 2002.

10 U. Garain and B.B. Chaudhuri, “Segmentation of Touching Characters in Printed Devnagari and Bangla Scripts using Fuzzy Multifactorial Analysis,” *IEEE Transactions on Systems, Man and Cybernetics,* Part C, vol. 32, no. 4, pp. 449- 459, 2002.

11 U. Garain, B.B. Chaudhuri, and T. Pal, “Online Handwritten Indian Script Recognition: A Human Motor Function based Framework,” In: *Proc. of the 16th Int. Conf. on Pattern Recognition* (ICPR), vol. III, pp. 164-167, Quebec City, Canada, 2002.

12 U. Garain, B.B. Chaudhuri, “On Machine Understanding of Online Handwritten Mathematical Expressions,” In: *Proc. of the 7th Int. Conf. on Document Analysis and Recognition* (ICDAR), pp. 349-353, Edinburgh, Scotland, 2003.

13 J. Mitra, U. Garain, B.B. Chaudhuri, H.V.K. Swamy, and T. Pal, “Automatic Understanding of Structures in Printed Mathematical Expressions,” In: *Proc. of the 7th Int. Conf. on Document Analysis and Recognition* (ICDAR), pp. 540-544, Edinburgh, Scotland, 2003.

14 U. Garain and B.B. Chaudhuri, “A Novel Approach for Machine Recognition of Online Handwritten Mathematical Symbols and Automatic Interpretation of Mathematical Expressions,” In: *Proc. of the 90th Indian Science Congress,* Section: Information and Communication Sciences, Part-III, pp. 39-40, Bangalore, India, January, 2003.

15 U. Garain, B.B. Chaudhuri, and R.P. Ghosh, “A Multiple Classifier System for Recognition of Printed Mathematical Symbols,” in the 17th Int’l Conf. on Pattern Recognition (ICPR), pp. 380-383, Cambridge, UK, 2004.

16 U. Garain, B.B. Chaudhuri, and A. Ray Chaudhuri, “Identification of Embedded Mathematical Expressions in Scanned Documents,” in the 17th Int’l Conf. on Pattern Recognition (ICPR), pp. 384-387, Cambridge, UK, 2004.

17 U. Garain, B.B. Chaudhuri, “Recognition of Online Handwritten Mathematical Expressions,” in *IEEE Transactions on Systems, Man and Cybernetics*, Part-B, vol. 34, no. 6, pp. 2366-2376, 2004.

18 U. Garain and B.B. Chaudhuri, “A Corpus for OCR of Printed Mathematical Expressions,” in *Int’l. Journal of Document Analysis and Recognition* (IJDAR), (in Press), 2005.

19 U. Garain and B.B. Chaudhuri, “Segmentation of Touching Symbols for OCR of Printed Mathematical Expressions: An Approach based on Multifactorial Analysis”, accepted in the 8th *Int’l. Conf. on Document Analysis and Recognition* (ICDAR), Seoul, Korea, 2005.