Processing Mathematical Notation 20

Dorothea Blostein and Richard Zanibbi

Contents

D. Blostein (✉)
School of Computing, Queen's University, Kingston, Canada
e-mail: blostein@cs.queensu.ca

R. Zanibbi
Department of Computer Science, Rochester Institute of Technology, Rochester, NY, USA
e-mail: rlaz@cs.rit.edu

Abstract

Automated recognition of mathematical notation is required for convenient document search and editing. The recognition problem varies depending on whether the input is a document image, vector graphics such as PDF, or handwritten tablet input. This chapter describes the state of the art in recognition of math notation, discussing the four component problems of expression detection, symbol recognition, layout analysis, and mathematical content interpretation.

Keywords

Graphics recognition • Math detection • Math recognition • Mathematical information retrieval • Syntactic pattern recognition

Introduction

Mathematical notation offers challenging pattern recognition problems, including segmentation ambiguities, symbol recognition challenges, and ambiguity of meaning [5, 58]. Similar problems arise in other branches of document image analysis (▶Chap. 3 (The Evolution of Document Image Analysis)). The following characteristics of math notation are favorable for automated recognition. In comparison with document notations such as tables (▶Chap. 19 (Recognition of Tables and Forms)) or music notation (▶Chap. 22 (Analysis and Recognition of Music Scores)), the semantics of math notation are compact and fairly well standardized. A typical math expression contains a modest number of well-separated symbols, making recognition algorithms computationally tractable. Most symbols in clean math notation are surrounded by white space, making symbol segmentation easier than in domains such as maps or engineering drawings. Also, the visual syntax (the symbol layout) of mathematical notation has a recursive structure, making this notation particularly well suited for syntactic pattern recognition techniques using grammars, tree rewriting, graph rewriting, and other recursive language processing methods.

Offsetting these tractable characteristics, mathematical notation also offers special challenges for automated recognition. As illustrated in Fig. 20.1, mathematical notation presents many types of ambiguities. Symbol recognition is challenging due to the large character set, which includes Roman and Greek letters, digits, and numerous operator symbols. Commas, dots, and other small symbols are common and can be difficult to distinguish from noise. Mathematical symbol recognition is further complicated by the variety of fonts, faces, and font sizes and the frequent occurrence of bold and italicized symbols.

Mathematical notation offers little redundancy, meaning that in many instances it is impossible to guess the identity of a symbol based on context. For example, consider a small mathematical expression consisting of an x followed by a noisy subscript: this subscript could be a Latin or Greek letter, a digit, or some other symbol chosen by the author. Sometimes context does provide helpful redundancy, as in expressions containing a clear subscripting pattern such as repeated subscripts

a $$\begin{array}{c} a \\ \hline b \\ \hline c \end{array}$$ **b** $$p^{a}$$ **c** $$\sum_{i=1}^{100} i^2 + i + y - x$$ **d** $$s \cdot t$$

Fig. 20.1 Ambiguities in mathematical expressions. (**a**) Which division is performed first? (**b**) Is the a superscripted? (**c**) What is the scope of the summation? (**d**) What do s, t, and · represent?

or ascending subscript values. Some recognition systems restrict the mathematical expression language in order to make better use of context. For example, the notation for multiplying a variable by a constant can be restricted by requiring the constant to appear first [36]. The strongest systems in the recent CROHME competition parse handwritten expressions based on the provided context-free expression grammars [37]. An expression grammar defines a set of legal expressions over a fixed symbol set. This reduces the language of expressions that may be recognized, but the constrained expression language makes it easier to construct effective algorithms for symbol segmentation, symbol classification, and layout recognition.

Mathematical notation is a semiformal visual language, with spatial relationships representing interactions between primitive mathematical objects. Formally defining math notation is difficult because the many domains of discourse give rise to dialects, and authors use many variations in notation. The 2010 Mathematical Subject Classification is an extensive categorization of mathematical domains used in math research (www.ams.org/mathscinet/msc/msc2010.html). A recent survey provides further references for the history and typesetting conventions of mathematical notation [58].

Mathematical notation uses six spatial relationships: *horizontal adjacency, above*, *below*, *superscript*, *subscript*, and *contains*. All six of these spatial relationships are illustrated by the expression below in Fig. 20.3a. Some mathematical symbols are *overloaded* with several possible meanings. For example, as shown in Fig. 20.3a, possible meanings for a horizontal line include division, subtraction, part of an $=$ symbol, or mean value of a list ($\bar{x}$). Operators are represented explicitly by symbols or implicitly by spatial relationships. For example, in Fig. 20.3a, implied multiplication is used between the $1/2$ and square root.

Systems and Applications

Research into computer recognition of mathematical notation dates back to the late 1960s [1, 8]. Publicly available math recognition systems are summarized in Table 20.1.

Computer recognition of math notation is useful in many contexts. One such context is document editing: math recognition software can be applied to user-specified images or to user input from pen, keyboard, and mouse. Another context is computer algebra systems, which provide various facilities for entering and editing math notation. Recognition of math notation is also useful in tutoring systems [58].

Table 20.1 Systems for recognition of math notation

Handwritten input	**Research systems**
	Natural Log [34] www.ai.mit.edu/projects/natural-log
	FFES/DRACULAE [47, 59] www.cs.rit.edu/~rlaz/ffes
	MathPad [25] mathpad.com
	MathPaper [29] pen.cs.brown.edu/research.html
	JMathNotes [52]
	PenMath [46] www.orcca.on.ca/penmath
	Mathbrush [24] www.scg.uwaterloo.ca/mathbrush
	Commercial systems
	Web Equation (Vision Objects)
	Pen-based math entry in the Windows operating system [39]
	MathJournal (XThink) www.xthink.com
Mouse and keyboard input	Xpress [41]
Document images	Infty [12, 49] www.inftyproject.org/en/index.html
	Supports document image and pen-based input and speech and Braille output

Mathematical information retrieval is an important application of math recognition. In mathematical information retrieval, a document collection can be searched using queries containing math notation; this is called *query by expression*. To support this, math recognition algorithms need to be applied not only to the query but to all documents in the collection. The documents must be annotated with the location of their math expressions as well as the interpretation (the recognition result) for these math expressions. Important open problems in mathematical information retrieval include creating effective indexing and retrieval algorithms for math notation, along with making effective use of math notation recognition results in the presence of recognition errors [58]. An ambitious project for large-scale annotation of documents in digital mathematics libraries is described in [35]. Mathematical information retrieval is closely tied to other types of document retrieval, including page classification and similarity (▸Chap. 7 (Page Similarity and Classification)), information retrieval from noisy text, navigation into graphic document masses (section "Logo Detection and Removal in Images and Videos" of ▸Chap. 18 (Logo and Trademark Recognition)), and retrieval based on document images and word spotting (▸Chap. 24 (Image Based Retrieval and Keyword Spotting in Documents)).

Inputs and Outputs of Math Recognition Systems

The input to a math recognition system can take three forms: *vector graphics* such as PDF, *strokes* such as pen strokes on a data tablet, or a *document image*. Figure 20.2 illustrates the use of touch, mouse, keyboard, and image input in the m_{in} system, a recent web-based search interface. The m_{in} interface was influenced by a number of earlier systems, including the pen-based equation editors Natural Log [34] and FFES [47, 59], the Infty math OCR system [12, 49], and the Xpress equation editor [41]. Pen-based interaction is also used in interfaces

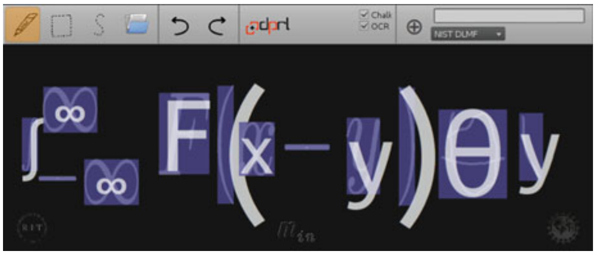

Fig. 20.2 Illustration of inputs and outputs of a math recognition system. This is m_{in}, a web interface for math search that runs on iPads [44]. Symbols may be placed on the canvas using keyboard and mouse. Math recognition can be applied to an uploaded image or to a handwritten input. The recognized symbols appear in *white*, overlayed on top of connected components from the input. In this screen shot, ten symbols have been correctly recognized from an uploaded image: Also, two handwritten symbols have been correctly recognized: θ and y. Once expression editing is complete, a click on the "+" button at *top right* inserts the recognized expression into the text box as LaTeX. Keywords may be added to the text box, in addition to LaTeX; the text box contents can then submitted to math-aware search engines such as NIST DLMF and Wolfram Alpha

to computer algebra systems such as Mathematica and Maple. Examples include MathBrush [24], E-chalk [51], MathPad^2 (also supporting diagram interaction) [25], and MathPaper [29]. Several of these pen-based computer algebra systems provide support for matrices.

The two classes of output produced by math recognition systems are layout trees and operator trees. Layout trees provide sufficient information for typesetting an expression or for searching based on the appearance of an expression (Fig. 20.3b). Operator trees contain the information required to evaluate an expression (Fig. 20.3c). Operator trees may be used to search based on the mathematical semantics of the expression. To evaluate an expression – whether by hand or by a computer algebra system – the operator tree must be supplemented with definitions and values for the variables and operations in the expression.

Four Component Problems in Recognition of Math Notation

Four component problems arise in the recognition of math notation: expression detection, symbol recognition or extraction, layout analysis, and mathematical content interpretation. Figure 20.4 illustrates the input formats and component problems.

The first component problem, *expression detection*, is discussed in section "Expression Detection." Methods for detecting offset expressions are fairly robust, but the detection of expressions embedded in text lines remains a challenge.

The second component problem, *symbol recognition or symbol extraction*, is discussed in section "Symbol Recognition or Symbol Extraction." Symbol recognition is challenging when the input consists of a document image or pen strokes. Hundreds of alphanumeric and mathematical symbols are used, many so similar in appearance that context is necessary for disambiguation. For example, context is

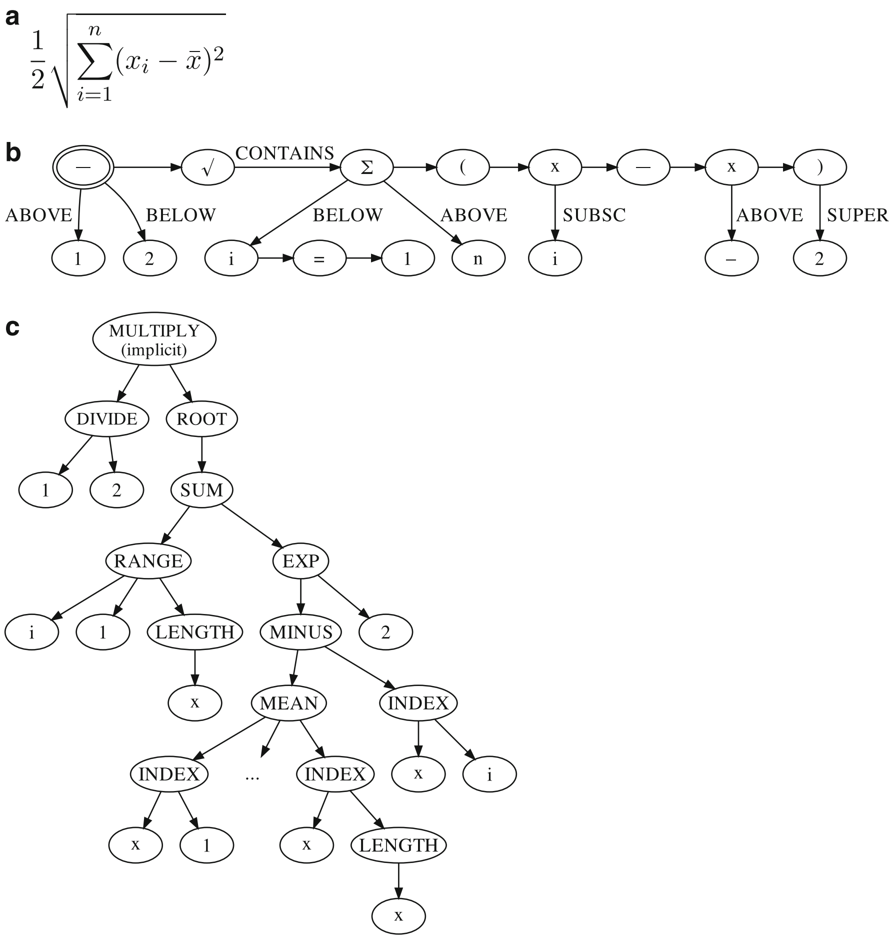

Fig. 20.3 Illustration of the classes of output provided by math recognition systems. (**a**) An image of math notation. (**b**) The layout tree describes the spatial structure of the expression as an organization of symbols into baselines. (The baselines are similar to writing lines used in text; see ▶Chaps. 10 (Machine-Printed Character Recognition) and ▶26 (Online Handwriting Recognition).) Information equivalent to a layout tree is provided by output formats such as LaTeX and Presentation MathML. (**c**) The operator tree represents mathematical operations and their operands. For example, the implied multiplication between the $1/2$ and square root in image (**a**) appears as the root of operator tree (**c**). The operator tree provides information about the meaning of symbols – for example, that n represents the length of vector x. Information equivalent to an operator tree is provided by output formats such as Content MathML and OpenMath

necessary to distinguish O, o, and 0 [36]. Symbol extraction is easier in vector-based representations such as PDF because these representations directly encode symbol locations and labels. Even vector-based representations require some processing to form complete symbols. For example, square root symbols are often typeset with the upper horizontal bar represented separately from the radical sign [3].

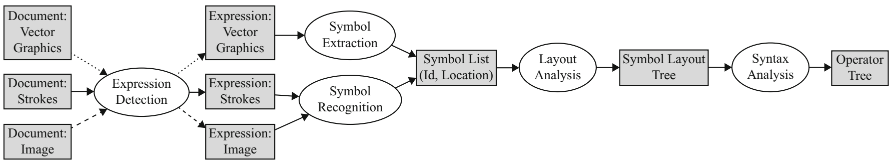

Fig. 20.4 Four component problems in recognition of math notation: expression detection, symbol recognition or symbol extraction, layout analysis, and interpretation of mathematical content. Shown at *left* are the possible input formats, including vector-based document encodings such as PDF files, pen/finger strokes, and document images. Many systems perform recognition in the order shown, but not all. For example, some systems combine Layout Analysis and Mathematical Content Interpretation, producing an operator tree directly using the expected locations of operator/relation arguments [6, 8]

The third component problem, *layout analysis*, is discussed in section "Layout Analysis." In layout analysis, spatial relationships between symbols are used to construct a layout tree (Fig. 20.3b). Correctly identifying the spatial relationships between symbols is often difficult, particularly in handwritten math notation.

The fourth component problem, *interpretation of mathematical content*, is discussed in section "Interpretation of Mathematical Content." Given the mathematical domain of discourse, mathematical content interpretation uses symbol layout information and information about the meaning of symbols to create an operator tree representing the expression semantics. Due to the recursive and nested structure of mathematical notation, context-free grammars are often used to define legal symbol layouts (equivalently, layout trees) and operations (i.e., operator trees). For example, to construct the operator tree in Fig. 20.3c, it is necessary to either know or infer that n denotes the length of vector x. Determining the meaning of symbols and structures is difficult, particularly if limited context is available. For example, the symbol λ can be used to represent a variable, a constant, or a binding function as in the Lambda Calculus. The meaning of λ can be deduced by analyzing the context in which the math notation occurs. In some situations ambiguities remain even if the mathematical context is known. For example, even if the mathematical context is known to be Bayesian probability, the symbol P may represent either a probability mass function or a probability density function.

Expression Detection

The input to a math recognition system can consist of vector graphics such as PDF, pen strokes, or a document image. Different challenges arise in detecting expressions in each of these input types, as discussed in sections "Detecting Expressions in Document Images," "Detecting Expressions in Vector Graphics" and "Detecting Expressions in Pen-Based Input." An overview of methods for expression detection is provided in Table 20.2.

Table 20.2 Methods for detecting math expressions

Document images: offset expressions	Clustering based on visual and layout features of connected components; operator range and dominance [21]
	Geometric features of nearest-neighbor graphs for connected components [11]
Document images: inline expressions	Fuzzy connected component classification, region growing around operators [21]
	Locate text lines, use symbol n-grams to identify text lines containing math [14]
Vector graphics	Projection profiles, rule-based classification of lines as *text* and *math*, followed by baseline extraction to obtain a symbol layout tree [3]
	Visual and layout features, identification of offset and embedded expressions using operator range and dominance and clustering of mathematical symbols [30]
Pen-based input	Gestures in combination with clustering or region growing [25, 52]
	Gestures for delimiting matrix elements [29, 53]

Detecting Expressions in Document Images

Detecting expressions in document images is one part of the *page segmentation* problem, where page regions that contain text, figures, tables, mathematics, images, and other graphical objects/notations are identified (see ▶Chap. 5 (Page Segmentation Techniques in Document Analysis)). Expressions in document images are commonly found using properties of connected components. *Offset* expressions are vertically separated from text (e.g., when defining an equation or function formally in a paper), whereas *embedded* expressions occur in the middle of lines of text. Offset expressions can be distinguished from text lines using attributes such as height, separation, character sizes, and symbol layout. Embedded expressions are difficult to detect reliably, particularly for expressions containing few symbols. Some methods of expression detection make use of OCR, whereas others locate expressions using only geometric features.

Kacem et al. detect offset expressions in images based on simple visual and layout features of adjacent connected components [21]. Embedded expressions are found by coarsely classifying connected components. Regions are grown around components that are identified as operators. The region growing is based on the expected locations for operands, using information about operator range and operator dominance. An operator *dominates* the operators belonging to its operands. During expression evaluation, the dominated operators must be applied before the dominant operator. For example, in the expression $1 * (2 + 3)$, the range of the multiplication operator is to the left and right of the *, with one operand on each side. The * operator dominates the + operator: during evaluation, the addition must be applied before the multiplication. In this example, parentheses make the range of * explicit.

If an image contains touching characters, then a single connected component contains more than one symbol. This can introduce errors into analysis based on connected components. Template matching can be used to address the problem of touching characters – see ▸Chap. 8 (Text Segmentation for Document Recognition).

An alternative approach for detecting embedded expressions first locates text lines and then computes symbol n-grams [14]. Training data provides information about the frequencies of symbol sequences for two classes of text lines: lines that are pure text versus lines that contain embedded expressions. Reported recall rates are as high as 95 % for embedded expressions and 97 % for offset expressions.

Offset expressions can be detected without symbol classification. Drake and Baird distinguish text lines from offset expressions using properties of the neighbor graph for connected components [11]. The reported accuracy for this method is high (over 99 %), but the method has not been applied to embedded expressions.

Detecting Expressions in Vector Graphics

The processing needed to extract content from born-digital documents such as PDF files differs from the processing needed for document images (see ▸Chap. 23 (Analysis of Documents Born Digital)). Unfortunately, vector graphics-based file formats such as PDF do not contain explicit demarcation of math regions. Developing reliable methods for finding math regions in PDF is an important direction for future work, particularly to support mathematical information retrieval. Work has begun on methods for extracting symbols and then automatically detecting the location of expressions in a PDF document.

One approach for detecting offset PDF expressions applies a vertical and a horizontal projection profile cut to a rendered image of the PDF page; the result is used to identify columns and then candidate text lines for offset expressions in a document page [3]. Symbols in the PDF file are joined using lexical rules, font face and size, vertical position, and horizontal separation. Lines are merged to form paragraphs and math regions, with classification of text vs. math performed using rules on layout and lexical structure of symbols on a line. Layout trees for detected offset expressions are created using a parser that performs baseline extraction (see section "Layout Analysis").

Another approach identifies embedded as well as offset expressions [30]. First, symbols are constructed from the PDF symbol primitives. Then "lines" that contain text or math are detected using a branch-and-bound algorithm, making use of baseline information provided within the PDF file. Offset expressions are identified using rules as well as a Support Vector Machine with features based on geometry, symbol, and font properties. Additional features used to classify lines as *math* versus *text* include the presence of specific symbols such as operator symbols, proximity to math symbols, and operator range and dominance. Symbol identities and operator dominance are used to identify expressions embedded in text lines, similar to the clustering strategy employed in [21].

In summary, expression detection in PDF files is nontrivial, even though symbol identities may be obtained almost directly from a PDF file. As for document images, detecting embedded expressions is more difficult than detecting offset expressions.

Detecting Expressions in Pen-Based Input

Pen-based math entry systems are a form of *sketching interface* (see ▸Chap. 28 (Sketching Interfaces)). In pen-based applications, expressions are often segmented using gestures [25, 52]. For example, a gesture is used in the E-chalk system to indicate the end of an expression and request its evaluation. Typically, a gesture gives a partial or approximate indication of the extent of an expression. Additional clustering or region growing methods can be applied, based on the visual features, identity, and distances between recognized symbols. Matrix elements can be detected using similar methods (see section "Syntactic Constraints Applied During Post-Processing").

Symbol Recognition or Symbol Extraction

Recognizing symbols in math notation is a difficult problem, due to the great variation in symbol size and the large number of classes [33], as well as problems caused by touching and over-segmented characters [34, 47]. Because of these factors, standard OCR methods such as described in ▸Chap. 10 (Machine-Printed Character Recognition) do not perform well when applied to math notation. In the early 1990s, it was observed that commercial optical character recognition systems with recognition rates of 99 % or higher fell to 10 % or less when tried on perfectly formed characters in mathematical equations [4]. Heuristics that are effective for text lines and tables fail with math notation because of variations in font size, multiple baselines, special characters, and differing n-gram frequencies. Selected methods for recognizing math symbols are summarized in Table 20.3.

For typeset symbols, recognition rates as high as 97.7 % (for over 600 classes) have been achieved using Support Vector Machines to reduce common class confusions [33]. A general discussion of techniques for recognizing non-textual symbols is provided in ▸Chap. 16 (An Overview of Symbol Recognition).

Accuracies for online recognition of handwritten mathematical symbols have been reported at rates of over 95 %. (See ▸Chaps. 26 (Online Handwriting Recognition) and ▸28 (Sketching Interfaces) for more information on online handwriting recognition and sketch-based interfaces). Some methods based on Hidden Markov Models (HMMs) extend early work by Winkler [56]. As a general trend, HMMs are being expanded to include more aspects of the math recognition problem. Initially, HMMs were used to perform segmentation and recognition for a time series of pen strokes. Recent examples of this approach include using features based on Freeman Chain Code [15] and using stroke features based on local curvature and pen position [20]. In more recent work, increasing amounts of layout and

Table 20.3 Methods for symbol recognition or symbol extraction

Handwritten input	Hidden Markov Models Isolated classification [15, 20] Simultaneous segmentation [56]
	PCA of preprocessed stroke data with quadratic classifier [34]
	Nearest neighbor Classification of strokes represented by polynomial basis functions [17] Greedy approximate Dynamic Time Warping for elastic matching [32] Using Freeman Chain Coding of strokes [15]
	Minimum spanning tree constraining legal stroke segmentations [34]
	AdaBoost to bootstrap writer-independent classification to writer dependent [26]
	Using symbol layout features Symbol segmentation incorporating layout [54] Dynamic programming for segmentation and classification [45]
Typeset expressions	Using SVMs to reduce frequency of common confusions [33]
Vector graphics	Combining PDF symbol information and connected components [3]

mathematical content information are being incorporated into HMM training and recognition [45]. An open problem is to extend current HMM methods to handle late additions to symbols. An example of a late addition is when a user enters a large expression but delays drawing the dot on top of an “i” until the end of expression entry. A related technique for recognizing handwritten symbols uses Dynamic Time Warping (DTW) to align features along symbol contours [32].

Another group of symbol recognition methods approximate handwritten strokes using linear combinations of basis vectors or parametric curves. Techniques include Principal Component Analysis (PCA) [34] and polynomial basis functions [17]. These methods perform dimensionality reduction on the raw stroke data. For example, Matsakis uses just 15 principal components to represent handwritten symbols which may be comprised of multiple strokes [34]. The basis vectors may be used to regenerate the original data up to a chosen level of fidelity; such features have intuitive appeal because the features can be viewed as strokes in the original input space.

Voting-based methods for classifier combination have been successfully applied to symbol recognition. Golubitsky and Watt use runoff elections in order to combine 1-against-1 SVM classifiers for a set of 280 symbols, with $280 * 279/2 = 39{,}060$ classifiers in total [18]. Majority voting is used in the first runoff election, followed by a tie-breaking runoff election that considers only votes for the top N classes. LaViola and Zelenik apply AdaBoost to an all-pairs classifier ensemble, with a binary classifier for every pair of classes [26]. Each base classifier uses only a single feature, where features are based on strokes and on the output of the Microsoft handwriting recognizer. This work aims to adapt a writer-independent classifier (the Microsoft classifier) to the handwriting of specific individuals through stroke-based features.

Various techniques use contextual information to constrain symbol recognition. A dynamic programming framework optimizes symbol segmentation and recognition in online handwritten input by searching all possible partitions of the stroke sequence to find the partition that optimizes a criterion function based on bigrams and probabilities of spatial relationships [45]. Symbol recognition results can also be corrected in post-processing using techniques such as math-specific n-grams [55] and error-correcting parsing [6].

Symbol recognition is easier in born-digital document representations, such as in PDF (see ▸Chap. 23 (Analysis of Documents Born Digital)), because these provide bounding boxes and symbol labels, along with an indication of the font to use in rendering each symbol. However, identifying exact symbol location requires examination of character font properties, because the boxes used to identify the placement of symbols in the PDF are not necessarily filled by the symbol. One solution is to combine PDF bounding box information with analysis of connected components in the rendered image [3]. Another option is to capture character font properties as characters are produced by a rendering library (e.g., using OpenFont). Symbol recognition in PDF must also allow for symbols that are represented by multiple primitives. For example, fraction lines are sometimes represented as multiple dashes and square root symbols as an upper horizontal bar and a separate symbol for the radical (the "check mark" at the left end of a square root symbol).

Layout Analysis

Analyzing the layout of a mathematical expression is difficult, particularly in handwritten notation. A number of factors contribute to this. Even when symbol identities are known, their spatial relationship may be highly ambiguous in some cases (see Fig. 20.1b). Sometimes operator symbols do not completely cover the extent of their arguments, such as when the numerator or denominator extends past the width of a fraction line. Another complication is interaction between the possible interpretations of symbol identity and symbol placement: because symbol identity constrains legal symbol adjacencies, incorrect symbol classification may lead to identifying invalid spatial relationships. For example, horizontal lines normally cannot have superscripts or subscripts, so misclassifying a symbol as a horizontal line precludes identifying the superscripts or subscripts of the misclassified symbol. Alternative segmentations of the input easily lead to combinatorial explosions in the number of possible symbol and layout interpretations.

Interactions between symbol location, symbol identity, and symbol relationships are endemic to graphics recognition and, indeed, to structural pattern recognition as a whole. These interactions reflect the well-understood interdependency between symbol segmentation and classification. They also reflect the less often discussed dependency between (a) determining location and identity of detected objects and (b) parsing relationships between objects. Consider text as an example. Recognition of symbol relationships is a necessary step for recognizing words and their sequence. OCR and handwriting recognition systems commonly detect the

Table 20.4 Layout analysis

Recursive decomposition	Projection profile cutting [15, 38]
	Baseline extraction [3, 59] w. MST constraining partitioning of non-baseline symbols [34, 52]
	Operator-driven decomposition: via operator dominance [6, 8, 28]
Syntactic methods (grammars and parsing)	Stochastic context-free grammar for symbols and layout [9, 36, 57]
	Fuzzy grammars [16]
	Incremental A* parse: measure consistency of symbol size, style, and repetition [43]
	Graph grammars and graph rewriting [19, 27]
Syntactic constraints applied during post-processing	Error-correcting parsing to correct symbol segmentation and recognition [6]
	LaTeX grammar to constrain handwritten symbols [15]
	Local grammatical rules to correct under-segmentation of vertical operators [54]
	Penalty graph representing symbols and spatial relations; find min-cost layout tree [12]
Matrix recognition	Virtual link networks [22]
	Projections of symbol bounding boxes [52]
	Region growing [28, 53]
	Analyzing the operator tree to correct errors in recognizing matrix structure [23]

adjacency of characters within text lines and use whitespace to segment words. In text, as in math, additional structural relationships must be analyzed to determine the reading order. A linguistic analysis of text – such as a parse tree – provides nonlinear structural relationships: the relationship between subject and object in a sentence is analogous to a superscript relationship in math notation. See ▸Chaps. 15 (Graphics Recognition Techniques), ▸17 (Analysis and Interpretation of Graphical Documents), ▸19 (Recognition of Tables and Forms), and ▸22 (Analysis and Recognition of Music Scores) for related discussions in other domains of graphics recognition.

Table 20.4 summarizes existing techniques for analyzing symbol layout in math expressions. Each of these methods has strengths and weaknesses, and improving the methods is an active area of research. Techniques for layout analysis include recursive decomposition of the input into subregions as well as syntactic methods that use grammars and parsers to produce a layout tree from a set of symbols or pen strokes. Syntactic constraints are often applied during post-processing, to correct the results of symbol recognition and layout analysis.

Generally, simple features are used to identify layout between symbols or sub-expressions. Examples of features include the relative placement of bounding boxes and properties of projection histograms. A common technique is to assign symbols to layout classes and define the properties of each layout class. Membership in a layout class restricts the set of allowable regions that may be associated with a symbol. For example, x might belong to a layout class that allows *adjacent*, *superscript*, and *subscript* regions, but does not allow *above*, *below*, or *contains* regions. Commonly region locations are defined using simple thresholds and the membership of symbols in these regions is tested using a single point, such as the symbol centroid. The vertical position of the centroid within the symbol's bounding box is adjusted according to the layout class of the symbol [59].

It is difficult to devise one layout analysis method that performs well for all inputs. As an alternative, an ensemble analysis could be created by combining the outputs of a set of layout analyzers. Classifier combination is a well-studied area, one that offers ideas and methods that might be adapted to a combination of layout analyzers.

Recursive Decomposition

The simplest methods for layout analysis recursively decompose the input into subregions. Projection profile cutting cuts an image into smaller regions at whitespace gaps in alternating vertical and horizontal projections, producing a tree. Baseline extraction identifies symbols sitting on the main baseline of an expression, partitioning the remaining symbols relative to baseline symbols, and recursively repeating the process in nonempty regions around baseline symbols. Operator-driven decomposition identifies the dominant operator in a region, partitioning remaining symbols into the expected locations for operands and recursively repeating the process in the operand regions. Each of these techniques is described in more detail below.

Projection profile cutting is closely related to X-Y cutting (see [58] for further discussion). A math expression image is decomposed by computing pixel intensity histograms and splitting at gaps in the histograms, alternating between projecting in the vertical and horizontal directions. Cutting stops when no further cuts may be made in a region, as when the region contains a single connected component or a square root symbol [38]. Subsequent steps can be used to merge split symbols, separate symbols within square roots and kerned characters, and incorporate cutting thresholds based on the estimated dominant character height and width [58]. An interesting property of projection profile cutting is that symbol recognition is performed *after* layout analysis. Special handling is used to identify over-segmented symbols, such as an i separated into a base stroke and a dot. The resulting tree of vertical and horizontal cuts may be directly mapped to a symbol layout tree.

Baseline extraction recursively decomposes a math expression by identifying symbols on the main baseline of an expression starting from the left end and partitioning remaining symbols into regions relative to the baseline symbols [59]. The leftmost baseline symbol is not always the leftmost symbol of the expression: for example, limit symbols can extend past the left end of an integral or summation symbol. A technique for identifying the leftmost symbol in a baseline is to examine symbols from right to left, applying tests for operator dominance [59]. Baseline extraction has been used in pen-based math entry systems [41,44,46,52,53], and the technique can be applied to document images as well. A minimum spanning tree can be used to improve the symbol partitioning step: for example, arguments that extend past the end of a summation symbol are clustered together during partitioning. Clustering also helps with the detection of subscripts at the left of a symbol, as when the subexpression “n choose 2” is written as ${}_{n}C_{2}$ [34, 52].

Operator-driven decomposition differs from the other layout analysis methods described in this section, in that it analyzes symbol layout to construct an operator tree rather than a layout tree. The operator tree is produced top-down, starting with the lowest precedence operator at the root and placing primitive arguments (e.g., variables and constants) at the leaves. The method uses operator dominance to identify the operator that has most or all of the remaining symbols in the expected operand locations [6, 8]. To avoid producing invalid operator trees, error-correcting parsing may be used to revise segmentation and classification hypotheses [6]. The earliest example of a pen-based calculator made use of this method [7].

Grammar-Based Syntactic Pattern Recognition Methods

A variety of grammar-based syntactic pattern recognition methods have been used to analyze symbol layout in math notation. Grammars provide explicit models for legal symbol layouts. Stochastic and fuzzy grammars are able to rank different layout interpretations. The way that a grammar constrains the space of possible layouts is analogous to how dictionaries provide word-level constraints for OCR results (see the chapters in ▶Part C (Text Recognition) and ▶Chap. 10 (Machine-Printed Character Recognition) in particular). A drawback is that grammar-based systems tend to be brittle: if symbol or layout recognition errors prevent a legal parse, then no output is produced. This limits the use of grammars in online systems, making it difficult to provide feedback about recognized layout before a valid expression has been entered completely.

Chou uses a stochastic context-free grammar to combine segmentation, symbol recognition, and layout analysis [9]. A probability is associated with each recognized symbol. Grammatical rules define how expressions may be composed bottom-up or decomposed top-down by concatenating or splitting sub-expressions vertically or horizontally. Probabilities are associated with each concatenation rule – these may be set empirically or tuned using the inside-outside algorithm [9]. There is a bias toward small parse trees, because long derivations in a stochastic grammar have low probabilities. To avoid this, some systems use a grammar to constrain the search for layouts, and then compute a confidence or probability based on the set of symbols and spatial relationships identified during parsing. For example, a linear combination of penalties based on probabilities for recognized symbols and layout can be used [2]. A related approach uses fuzzy grammars, where membership functions rather than probabilities are used for symbol classes and spatial relationships [16].

Graph grammars have also been used to perform layout analysis. Graph productions are applied to a host graph in which graph edges represent spatial and logical relationships among symbols [19, 27]. The use of graphs, rather than strings, allows production rules to more conveniently express spatial constraints. However, there is additional computational cost in applying rules.

Syntactic Constraints Applied During Post-Processing

Many layout parsers use layout information to disambiguate symbol recognition results. For example, in segmenting and parsing online handwritten symbols, local grammatical rules can be used to correct under-segmentation of vertical operators such as fractions, square roots, and summations [54]. Variations of the CYK parsing algorithm (a dynamic programming algorithm for parsing context-free grammars) have been used to apply symbol layout information to constrain symbol recognition for stochastic context-free grammars [2].

In penalty graph minimization, candidate relationships between pairs of symbols are defined before minimizing a penalty criterion over a set of possible layout trees [12]. Relationships over pairs of candidate symbol identities are considered using a branch-and-bound variation of Prim's minimum spanning tree algorithm. A set of approximately minimum-cost spanning trees are constructed, using syntactic constraints to ensure that symbols and relationships added to a tree are consistent. For example, two relationships attached to a symbol must assign the same identity to that symbol. After obtaining the final set of candidate trees, penalties are modified according to additional heuristics related to expression syntax and appearance. For example, symbols in exponents are expected to be smaller than the base symbol. The layout tree that has minimum penalty is selected for output.

Matrix Recognition and Tabular Structures

Matrix recognition is a challenging open problem in layout analysis, closely related to table recognition (see ▸Chap. 19 (Recognition of Tables and Forms)). Matrix recognition is also closely related to the problem of identifying other tabular structures used in math notation, including lists of expressions in derivations, and lists occurring in conditional statements such as the following:

$$t(x) = \begin{cases} 1, & \textit{if } x \geq 0 \\ 0, & \textit{otherwise} \end{cases}$$

Recognition of small typeset matrices is fairly easy, because there is generally more whitespace between matrix elements than between symbols within a matrix element. However, in the absence of large regular gaps of separating whitespace it can be difficult to determine the correct segmentation into matrix elements. Several matrix recognition techniques begin by searching for a large left and right parenthesis or brace, to determine the location of the matrix. Once the matrix location is known, segmentation of matrix elements is achieved through region growing [28,53] or through projections of symbol bounding boxes [52]. In an extension of the virtual link network method, projections of matrix symbols produce a linear system of equations for estimating row and column positions [22]. Recent work allows ellipses (using "…" to indicate a repeating pattern of elements) in pen-based interfaces for computer algebra systems [29, 53]. See also ▸Chap. 28 (Sketching Interfaces) on sketching interfaces.

Interpretation of Mathematical Content

Many present-day math recognition systems perform layout analysis but do not go on to determine the mathematical meaning of the expression. The output from such systems suffices for typesetting purposes, but does not support expression evaluation or interaction with computer algebra systems. In order to support expression evaluation, it is necessary to produce some form of operator tree (Fig. 20.3c) representing logical relationships and domain semantics.

The semantics of a mathematical domain of discourse are difficult to deduce. Most existing systems that interpret mathematical content do so by assuming a given math dialect is used: the math dialect provides the definitions for operators and relations. In future work, content dictionaries such as those provided by OpenMath [10] could be used to define the meanings of mathematical symbols in various dialects.

Table 20.5 lists methods that use parsing techniques to analyze symbol layout in order to directly produce an operator tree. (In contrast, the Operator-Driven Decomposition described in section "Recursive Decomposition" constructs an operator tree top-down.) Various types of attributed grammars have been used, including context-free string grammars [13] and graph grammars [19, 27]. As for the methods discussed in section "Layout Analysis," spatial relationships are determined by testing simple geometric features, but with the output of parsing being an operator tree rather than a symbol layout tree. An alternative approach is to transform a layout tree into an operator tree, using a grammar that defines operator trees in this math dialect. One such approach organizes layout analysis and interpretation of mathematical content into a series of stages similar to a compiler [58].

Table 20.5 Interpretation of mathematical content

Integrated layout analysis and content interpretation	Top-down, via operator dominance (also see Table 20.4) [6, 8, 28]
	Context-free string grammar [13]
	Graph grammar to produce a layout/content graph [19, 27]
Compiler-based approach	Tree transformation to translate symbol layout tree to operator tree [59]

Table 20.6 Datasets

Datasets for math recognition	Infty I-III www.inftyproject.org/en/database.html [50] Infty I: over 20,000 typeset expressions from 30 technical articles (476 pages). Manually created ground provides symbols with bounding boxes and edges of the symbol layout tree in .csv, XML, and MathML. Infty II adds 37 documents (English, French, German). Infty III: database of over 250,000 math characters and symbols
	UW-III www.science.uva.nl/research/dlia/datasets/uwash3.html [40] 25 pages with math content (approximately 100 typeset equations). Ground truth created with double entry and triple verification: LaTeX and labeled bounding boxes for expressions and symbols
	Waterloo/MathBrush www.scg.uwaterloo.ca/mathbrush/corpus [32] 4,655 handwritten expressions by 20 writers. Semiautomated ground truth method provides operator trees, LaTeX, .gif, Microsoft, and SCG ink formats
	CROHME www.isical.ac.in/∼crohme [37] Data used in the ICDAR 2011 online handwritten math recognition contest (∼1,000 handwritten expressions from multiple writers; dataset is being expanded)
	HAMEX www.projet-depart.org [42] Handwritten and audio: 4,350 online handwritten expressions by 58 writers and read aloud in French by 58 speakers. Ground truth for handwriting: INKML files with digital ink, symbol segmentation, and MATHML structure. Ground truth for audio: XML files with transcription of the spoken expressions
	Marmot www.founderrd.com/marmot_data.htm [31] 400 pages from 194 digitally originated PDF documents with 1,575 offset and 7,907 embedded expressions. XML ground truth created semiautomatically
Statistical information about math notation	Empirical study of over 19,000 papers stored in the ArXiv e-Print Archive [48] – Provides frequency of symbols and operators used in different math domains
	N-grams from analysis of LaTeX sources for 3 engineering math textbooks [55]

Validation and Datasets

The choice of data, performance metrics, and evaluation protocols significantly affects the results obtained when comparing document recognition systems. An effective comparison determines the relative strengths and weaknesses of the systems; see ▸Chaps. 29 (Datasets and Annotations for Document Analysis and Recognition) and ▸30 (Tools and Metrics for Document Analysis Systems Evaluation). Meaningful comparison of math recognition system performance is challenging because systems focus on a variety of mathematical domains, layout conventions, and components of the recognition process. For example, it is difficult to compare a high-accuracy system that processes a narrow range of inputs to a lower-accuracy system that processes a broad range of inputs. The CROHME competition was started in order to address this issue for the problem of online handwritten math input [37].

Table 20.6 lists benchmark data sets for training and evaluation of math recognition systems. It remains difficult to characterize the representativeness of a data set, the relevance of the data for a particular application, and the degree of noise tolerance of a math recognition system.

Performance metrics for evaluation of math recognition systems include expression recognition rate, symbol recognition rate, measures of layout structure accuracy based on token placement and baselines, string edit distance applied to Euler strings derived from layout trees, and metrics using bipartite graphs defined over nodes representing strokes or connected components [58].

Conclusion

Technology for recognition of math notation is maturing, as demonstrated by the availability of numerous research systems and several commercial systems. Avenues for future work include the following: Improve segmentation algorithms, both for detecting inline expressions, and for segmenting symbols in handwritten expressions. Increase the reliability of layout analysis through the use of more robust models for symbol layout. This was identified as a key problem in the CROHME 2011 competition [37]. A related problem is to improve the reliability and flexibility of matrix processing. This might be done through the development of parser combination methods for producing ensembles of existing layout analysis methods.

The versatility and robustness of math recognition systems can be increased through the following avenues of investigation:

- **Integrate the recognition stages: segmentation, symbol classification, layout analysis, and interpretation of mathematical content.** Integration has the potential to reduce recognition errors by making effective use of contextual information. At heart, this global optimization is a machine learning problem: the component algorithms in a math recognition system should interact to produce a globally optimal result. Various integration methods in math recognition have been explored, notably grammars and dynamic programming. This is a promising avenue for future development.
- **Create language models with statistical information about math notation.** Stochastic language models will continue to become increasingly sophisticated, extending stochastic grammars [9] using a variety of segmentation and parsing approaches. This development will be fueled by increasing availability of data sets and statistical information about math notation [48, 55].
- **Develop recognition systems that can identify and understand various dialects of math notation.** A starting point is provided by the categorization defined in the Mathematical Subject Classification (www.ams.org/mathscinet/msc/msc2010.html). Develop a model of math notation that can be adapted to dialects, perhaps using content dictionaries such as those provided by OpenMath [10] to define the meanings of mathematical symbols in various dialects.

- **Adopt more formal problem statements for math recognition.** The language of layouts and expressions accepted by math recognition systems should be explicitly defined. This could be done by formally defining languages of layout trees and operator trees. Such formality is needed in order to precisely characterize the scope of a dataset and the range of inputs accepted by a math recognition system. The language models should explicitly define the types of noise that can be tolerated by a system and the types of errors that can be corrected by a system.
- **Detect the mathematical domain of discourse.** To process math expressions in large document repositories, methods are needed for identifying the mathematical domain of discourse. This requires analysis of the document text to find information such as symbol definitions of the type "Let X be ....."

An effective user interface is essential in creating a usable math recognition system; indeed, doing this for document recognition systems in general is ongoing work (►Chap. 3 (The Evolution of Document Image Analysis)). Future directions for interfaces supporting math recognition include:

- Improve the convenience and effectiveness of methods for displaying recognition results to the user. Existing methods include two-window displays, morphing symbols to visualize the recognized symbol layout and displaying symbol recognition results using images placed behind user input or placed on top of user input in a transparent layer [58].
- Increase the use of multimodal input, such as handwritten math expression data combined with audio, allowing a user to both draw and speak an expression [42].
- Increase the predictability of performance of math recognition systems. In order for recognition systems to effectively compete with direct entry systems (such as LaTeX or structure-based editors), the user of a recognition system must feel confident about how to interact with the system in order to obtain highly reliable recognition results.

An important trend is increased integration of recognition and retrieval [58]. This requires advances in user interfaces for convenient query by expression, as well as advances in retrieval algorithms to make effective use of noisy recognition results.

Acknowledgements Financial support from the National Science Foundation, USA, (Grant No. IIS-1016815) and the Natural Sciences and Engineering Research Council of Canada are gratefully acknowledged.

Cross-References

►Analysis and Interpretation of Graphical Documents
►Analysis and Recognition of Music Scores
►Analysis of Documents Born Digital
►Analysis of the Logical Layout of Documents
►An Overview of Symbol Recognition
►Continuous Handwritten Script Recognition
►Graphics Recognition Techniques
►Handprinted Character and Word Recognition

►Image Based Retrieval and Keyword Spotting in Documents
►Language, Script, and Font Recognition
►Logo and Trademark Recognition
►Machine-Printed Character Recognition
►Online Handwriting Recognition
►Online Signature Verification
►Page Segmentation Techniques in Document Analysis
►Page Similarity and Classification
►Recognition of Tables and Forms
►Text Segmentation for Document Recognition

References

1. Anderson R (1977) Syntax-directed recognition of hand-printed two-dimensional equations. PhD thesis, Harvard University, Cambridge, Jan 1968. Portions of this thesis appear as a chapter. In: Fu KS (ed) Syntactic pattern recognition, applications. Springer, pp 147–177
2. Awal A-M, Mouchére H, Viard-Gaudin C (2009) Towards handwritten mathematical expression recognition. In: Proceedings of the 10th international conference on document analysis and recognition, Barcelona, pp 1046–1050
3. Baker J, Sexton A, Sorge V, Suzuki M (2011) Comparing approaches to mathematical document analysis from PDF. In: Proceedings of the 11th international conference on document analysis and recognition, Beijing, pp 463–467
4. Berman B, Fateman R (1994) Optical character recognition for typeset mathematics. In: Proceedings of the 1994 international symposium on symbolic and algebraic computation, Oxford, pp 348–353, July 1994
5. Chan K-F, Yeung D-Y (2000) Mathematical expression recognition: a survey. Int J Doc Anal Recognit 3:3–15
6. Chan K-F, Yeung D-Y (2001) Error detection, error correction and performance evaluation in on-line mathematical expression recognition. Pattern Recognit 34(8):1671–1684
7. Chan K-F, Yeung D-Y (2001) Pencalc: a novel application of on-line mathematical expression recognition technology. In: Proceedings of the 6th international conference on document analysis and recognition, Seattle, pp 774–778
8. Chang S-K (1970) A method for the structural analysis of two-dimensional mathematical expressions. Inf Sci 2(3):253–272
9. Chou P (1989) Recognition of equations using a two-dimensional stochastic context-free grammar. In: Visual communications and image processing IV, Philadelphia. SPIE, vol 1199, pp 852–863
10. Dewar M (2000) Openmath: an overview. ACM SIGSAM Bull 34:2–5
11. Drake D, Baird H (2005) Distinguishing mathematics notation from English text using computational geometry. In: Proceedings of the 8th international conference on document analysis and recognition, Seoul, pp 1270–1274
12. Eto Y, Suzuki M (2001) Mathematical formula recognition using virtual link network. In: Proceedings of the 6th international conference on document analysis and recognition, Seattle, pp 430–437
13. Fateman R, Tokuyasu T (1996) Progress in recognizing typeset mathematics. Proc Int Soc Opt Eng 2660:37–50
14. Garain U (2009) Identification of mathematical expressions in document images. In: Proceedings of the 10th international conference on document analysis and recognition, Barcelona, pp 1340–1344
15. Garain U, Chaudhuri BB (2004) Recognition of online handwritten mathematical expressions. IEEE Trans Syst Man Cybern 34(6):2366–2376

16. Genoe R, Fitzgerald JA, Kechadi T (2006) An online fuzzy approach to the structural analysis of handwritten mathematical expressions. In: Proceedings of the IEEE international conference on fuzzy systems, Vancouver, pp 242–250, July 2006
17. Golubitsky O, Watt SM (2010) Distance-based classification of handwritten symbols. Int J Doc Anal Recognit 13(2):133–146
18. Golubitsky O, Watt SM (2010) Improved classification through runoff elections. In: Proceedings of the international workshop document analysis systems, Boston, pp 59–64
19. Grbavec A, Blostein D (1995) Mathematics recognition using graph rewriting. In: Proceedings of the 3rd international conference on document analysis and recognition, Montreal, pp 417–421
20. Hu L, Zanibbi R (2011) HMM-based recognition of on-line handwritten mathematical symbols using segmental k-means initialization and a modified pen up/down feature. In: Proceedings of the international conference on document analysis and recognition, Beijing, pp 457–462
21. Kacem A, Belaid A, Ben Ahmed M (2001) Automatic extraction of printed mathematical formulas using fuzzy logic and propagation of context. Int J Doc Anal Recognit 4(2):97–108
22. Kanahori T, Suzuki M (2002) A recognition method of matrices by using variable block pattern elements generating rectangular areas. In: Graphics recognition – algorithms and applications. LNCS, vol 2390. Springer, pp 320–329
23. Kanahori T, Sexton A, Sorge V, Suzuki M (2006) Capturing abstract matrices from paper. In: Mathematical knowledge management. LNAI, vol 4108. Springer, pp 124–138
24. Labahn G, Lank E, MacLean S, Marzouk M, Tausky D (2008) Mathbrush: a system for doing math on pen-based devices. In: Proceedings of the eighth IAPR workshop on document analysis systems (DAS 2008), Nara. IEEE Computer Society, pp 599–606
25. LaViola J, Zeleznik R (2004) Mathpad2: a system for the creation and exploration of mathematical sketches. ACM Trans Graph (Proc SIGGRAPH 2004) 23(3):432–440
26. LaViola J, Zeleznik R (2007) A practical approach to writer-dependent symbol recognition using a writer-independent recognizer. IEEE Trans Pattern Anal Mach Intell 29(11): 1917–1926
27. Lavirotte S, Pottier L (1997) Optical formula recognition. In: Proceedings of the 4th international conference on document analysis and recognition, Ulm, pp 357–361
28. Lee H-J, Wang J-S (1997) Design of a mathematical expression understanding system. Pattern Recognit Lett 18(3):289–298
29. Li C, Zeleznik R, Miller T, LaViola J (2008) Online recognition of handwritten mathematical expressions with support for matrices. In: Proceedings of the 19th international conference on pattern recognition, Tampa, pp 1–4
30. Lin X, Gao L, Tang Z, Lin X, Hu X (2011) Mathematical formula identification in PDF documents. In: Proceedings of the 11th international conference on document analysis and recognition, Beijing, pp 1419–1423
31. Lin X, Gao L, Tang Z, Lin X, Hu X (2012) Performance evaluation of mathematical formula identification. In: Proceedings of the 10th IAPR international workshop on document analysis systems, Gold Coast, pp 287–291
32. MacLean S, Labahn G, Lank E, Marzouk M, Tausky D (2011) Grammar-based techniques for creating ground-truthed sketch corpora. Int J Doc Anal Recognit 14(1):65–74
33. Malon C, Uchida S, Suzuki M (2008) Mathematical symbol recognition with support vector machines. Pattern Recognit Lett 29(9):1326–1332
34. Matsakis N (1999) Recognition of handwritten mathematical expressions. Master's thesis, Massachusetts Institute of Technology, Cambridge, May 1999
35. Michler G (2003) How to build a prototype for a distributed digital mathematics archive library. Ann Math Artif Intell 38:137–164
36. Miller E, Viola P (1998) Ambiguity and constraint in mathematical expression recognition. In: Proceedings of the 15th national conference of artificial intelligence, Madison, pp 784–791, July 1998
37. Mouchère H, Viard-Gaudin C, Kim DH, Kim JH, Garain U (2011) CROHME2011: competition on recognition of online handwritten mathematical expressions. In: Proceedings

of the 11th international conference on document analysis and recognition, Beijing, pp 1497–1500
38. Okamoto N, Miao B (1991) Recognition of mathematical expressions by using the layout structures of symbols. In: Proceedings of the 1st international conference on document analysis and recognition, Saint-Malo, pp 242–250
39. Panic M (2009) Math handwriting recognition in Windows 7 and its benefits. In: Intelligent computer mathematics. LNCS, vol 5625. Springer, Berlin/Heidelberg, pp 29–30
40. Phillips I (1998) Methodologies for using UW databases for OCR and image understanding systems. In: Proceedings of the document recognition V, San Jose. SPIE, vol 3305, pp 112–127
41. Pollanen M, Wisniewski T, Yu X (2007) Xpress: a novice interface for the real-time communication of mathematical expressions. In: Proceedings of the workshop on mathematical user-interfaces, Linz, June 2007
42. Quiniou S, Mouchère H, Peña Saldarriaga S, Viard-Gaudin C, Morin E, Petitrenaud S, Medjkoune S (2011) HAMEX – a handwritten and audio dataset of mathematical expressions. In: Proceedings of the 11th international conference on document analysis and recognition, Beijing, pp 452–456
43. Rhee TH, Kim JH (2009) Efficient search strategy in structural analysis for handwritten mathematical expression recognition. Pattern Recognit 42(12):3192–3201
44. Sasarak C, Hart K, Pospesel R, Stalnaker D, Hu L, LiVolsi R, Zhu S, Zanibbi R. (2012) m$_{in}$: a multimodal web interface for math search. In: Symposium on human-computer interaction and information retrieval, Cambridge. Online: https://sites.google.com/site/hcirworkshop/hcir-2012
45. Shi Y, Soong FK (2008) Symbol graph based discriminative training and rescoring for improved math symbol recognition. In: Proceedings of the international conference on acoustics, speech, and signal processing, Las Vegas, pp 1953–1956
46. Smirnova E, Watt S (2008) Communicating mathematics via pen-based computer interfaces. In: Proceedings of the 10th international symposium on symbolic and numeric algorithms for scientific computing (SYNASC 2008), Timisoara, pp 9–18
47. Smithies S, Novins K, Arvo J (1999) A handwriting-based equation editor. In: Proceedings of the graphics interface, Kingston, pp 84–91, June 1999
48. So CM, Watt SM (2005) Determining empirical characteristics of mathematical expression use. In: Proceedings of the mathematical knowledge management. LNCS, vol 3863. Springer, pp 361– 375
49. Suzuki M, Tamari F, Fukuda R, Uchida S, Kanahori T (2003) INFTY: an integrated OCR system for mathematical documents. In: Proceedings of the ACM symposium on document engineering 2003, Grenoble, pp 95–104
50. Suzuki M, Uchida S, Nomura A (2005) A ground-truthed mathematical character and symbol image database. In: Proceedings of the 8th international conference on document analysis and recognition, Seoul, pp 675–679
51. Tapia E, Rojas R (2003) Recognition of on-line handwritten mathematical formulas in the E-chalk system. In: Proceedings of the 7th international conference on document analysis and recognition, Edinburgh, pp 980–984
52. Tapia E, Rojas R (2004) Recognition of on-line handwritten mathematical expressions using a minimum spanning tree construction and symbol dominance. In: Graphics recognition, recent advances and perspectives. LNCS, vol 3088. Springer, Berlin/New York, pp 329–340
53. Tausky D, Labahn G, Lank E, Marzouk M (2007) Managing ambiguity in mathematical matrices. In: Proceedings of the 4th Eurographics workshop on sketch-based interfaces and modeling, Riverside California, pp 115–122
54. Toyozumi K, Yamada N, Mase K, Kitasaka T, Mori K, Suenaga Y, Takahashi T (2004) A study of symbol segmentation method for handwritten mathematical formula recognition using mathematical structure information. In: Proceedings of the 17th international conference on pattern recognition, Cambridge, vol 2, pp 630–633

55. Watt SM (2008) An empirical measure on the set of symbols occurring in engineering mathematics texts. In: Proceedings of the 8th IAPR international workshop on document analysis systems (DAS 2008), Nara, pp 557–564
56. Winkler H-J (1996) HMM-based handwritten symbol recognition using on-line and off-line features. In: Proceedings of the international conference on acoustics speech and signal processing, Atlanta, pp 3438–3441
57. Yamamoto R, Sako S, Nishimoto T, Sagayama S (2006) On-line recognition of handwritten mathematical expressions based on stroke-based stochastic context-free grammar. In: Proceedings of the 10th international workshop on frontiers in handwriting recognition, La Baule, Oct 2006
58. Zanibbi R, Blostein D (2012) Recognition and retrieval of mathematical expressions. Int J Doc Anal Recognit 15(4):331–357
59. Zanibbi R, Blostein D, Cordy JR (2002) Recognizing mathematical expressions using tree transformation. IEEE Trans Pattern Anal Mach Intell 24(11):1455–1467

Further Reading

Readers looking to expand their knowledge of math recognition are directed to the survey papers [5] and [58]. Tables 20.1–20.6 above direct the reader to references relevant to particular aspects of math recognition.