International Journal on Document Analysis and Recognition manuscript No.
(will be inserted by the editor)

Mathematical Expression Recognition: A Survey

Kam-Fai Chan, Dit-Yan Yeung

Department of Computer Science, The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
e-mail: `{kchan,dyyeung}@cs.ust.hk`

Received: date / Revised version: date

Abstract. Automatic recognition of mathematical expressions is one of the key vehicles in the drive towards transcribing documents in scientific and engineering disciplines into electronic form. This problem typically consists of two major stages, namely, symbol recognition and structural analysis. In this survey paper, we will review most of the existing work with respect to each of the two major stages of the recognition process. In particular, we try to put emphasis on the similarities and differences between systems. Moreover, some important issues in mathematical expression recognition will be addressed in depth. All these together serve to provide a clear overall picture of how this research area has been developed to date.

Key words: error detection and correction – mathematical expression recognition – performance evaluation – structural analysis – symbol recognition

1 Introduction

With the very rapid increase of Internet users in recent years, there is a growing trend of disseminating and exchanging information via this popular channel. Digital library and distance learning are becoming hot research areas that address issues arisen from the widespread use of the Internet. One of the key vehicles in the drive towards realizing these ideas is to develop cheap and efficient methods for transcribing existing knowledge in the form of paper documents into corresponding electronic form, which is the form that can be processed by today's digital computers and transmitted through the Internet.

Mathematical expressions constitute an essential part in most scientific and engineering disciplines. The input of mathematical expressions into computers is often more difficult than that of plain text, because mathematical expressions typically consist of special symbols and Greek letters in addition to English letters and digits. With such a large number of characters and symbols, the commonly used type of keyboard has to be specially modified in order to accommodate all the keys needed, as done in [21]. Another method is to make use of some extra keys in the keyboard (e.g., function keys) along with a set of unique key sequences for representing other special symbols, as in [48]. Yet another method is to simply define a set of keywords to represent special characters and symbols, as in LaTeX [36].

However, working with specially designed keyboards or keywords requires intensive training and practice. Alternatively, by taking advantage of pen-based computing technologies, one could simply write mathematical expressions on an electronic tablet for the computer to recognize them automatically. In situations where the expressions are already in some printed form, we could just scan in the document for the computer to recognize the expressions directly from the image.

Mathematical expression recognition typically consists of two major stages: *symbol recognition* and *structural analysis*. Character recognition, as the most common type of symbol recognition problems, has been an active research area for more than three decades [47,60]. Structural analysis of two-dimensional patterns also has a long history [50]. However, as emphasized in [5,11,14,40], very few papers had addressed specific problems related to mathematical expression recognition. It is only until recently that more researchers have started to pay attention to this area.

So far, to the best of our knowledge, papers that provide literature survey of the area of mathematical expression recognition research are very rare. In [7], the recognition problem is first defined and then followed by a survey of existing work according to major sub-parts of the recognition problem. However, comparison between different systems with respect to different aspects is generally not provided.

In this paper, we will remedy such shortcoming by putting more emphasis on the similarities and differences between systems. Besides, we will include more recent papers not covered in [7]. First of all, we will discuss some typical properties of mathematical expressions which make their recognition difficult. Then, we will give an overview of the recognition process. Afterwards, we will provide a survey of existing work in mathematic expression recognition, with emphasis on comparison between systems in each major stage of the recognition process. Finally, we will discuss other related issues and future research directions which are then followed by some concluding remarks.

Note that some existing systems for off-line recognition of mathematical expressions deal with data that contain both text lines and mathematical expressions in the same document [16,29,30,42,52,61]. Extraction of mathematical expressions and other similar objects (for example, diagrams, graphical drawings, chemical equations, etc.) from documents requires similar techniques. This indeed can be a large topic by itself [59]. Our survey, however, will only cover the recognition of mathematical expressions. Readers who are interested in those techniques for extracting mathematical expressions from a document may find details in the papers mentioned above.

2 Properties of Mathematical Expressions

In a mathematical expression, characters and symbols can be spatially arranged as a complex two-dimensional structure, possibly of different character and symbol sizes. All the characters and symbols, when grouped properly, form an internal hierarchical structure.

However, proper grouping of symbols in a mathematical expression is not trivial. Firstly, there are two types of symbols. One type includes all basic symbols and the other includes binding, fence and operator symbols. Each type of symbols has its own grouping criteria. Secondly, there are also two types of operators, namely, explicit and implicit operators. Explicit operators are operator symbols while implicit operators are spatial operators. Thirdly, some symbols may represent different meanings in different contexts. These properties together make the recognition process very difficult even when all the individual characters and symbols can be recognized correctly.

2.1 Grouping Basic Symbols

Undoubtedly, every symbol has its own meaning. However, in a mathematical expression, sometimes there is a need for grouping some adjoining symbols together to represent another meaning. The following are some general rules:

1. Digits together usually form a unit when they are of the same size, adjacent to each other, and written on the same horizontal line, e.g., 210 represents an integer value. On the other hand, the same digits but with different sizes and positions may carry a different meaning, e.g., 2^{10} consists of two units which are 2 and 10 respectively.
2. Several letters together may form a unit, like some trigonometric functions such as `tan`, `sin` and `cos`. Before considering a group of letters as a concatenation of variable names representing their multiplicative product, we should first check whether they together form a function name.
3. Symbols other than letters and digits should be considered as separate units.

2.2 Grouping Binding, Fence and Operator Symbols

The presence of some symbols in a mathematical expression may invoke some special grouping methods. The following are three types of such symbols:

1. Binding symbols, such as fraction line, $\sqrt{}$ and $\sum$, dominate their neighboring sub-expressions. For example, in " $\sum_{i=1}^{10} i$ ", three sub-expressions, i.e., "10", "$i = 1$", and "i" are bound to the symbol $\sum$ which gives meaning to the expression as the sum of 1, 2, ..., 10. However, deciding proper relationships among binding symbols and their neighboring subexpressions becomes non-trivial in some nested expressions, for example,
$$\text{“}\sum_{i=1}^{10} \frac{i}{a+b}\text{”} \text{ and } \text{“ } \frac{\sum_{i=1}^{10} i}{a+b} \text{ ”}.$$
2. Fence symbols, such as parentheses, group the enclosed units into one single compound unit. For example, in "$a(b+c)$", "$b+c$" is regarded as a unit that will be evaluated first.
3. Operator symbols, such as $+$, $-$, $*$ and $/$, dominate their operands. For example, in "$a + b$", $+$ imposes an addition operation on its operands a and b.

2.3 Explicit and Implicit Operators

Explicit operators are operator symbols. When consecutive operator symbols exist in an expression, we can apply operator precedence rules to group the symbols into units. However, when those operator symbols are not lined up, we have to use the concept of operator dominance [10]. For example, in " $a + \frac{b}{c}$ ", the meaning is "$a + (b/c)$" due to the fact that the operator $+$ dominates $/$ (where $/$ lies in the range of $+$). However, in " $\frac{a+b}{c}$ ", the meaning becomes "$(a + b)/c$" since $/$ dominates $+$ (where $+$ lies in the range of $/$) in this case.

In some mathematical expressions, there also exist implicit operators. Implicit operators (also called spatial operators) determine the relationships between symbols simply by their relative positions. For example, in "a^2", 2 is the superscript of a representing the square of a. However, in "a_2", 2 is the subscript of a representing only a variable name. Although it is somewhat unusual, "$a2$" can be used to represent the multiplication of a and 2.

2.4 Context-Sensitive Roles

Some symbols in mathematical expressions may play different roles in different contexts. Here are some examples:

1. A dot in an expression can be a decimal point or a multiplication operator depending on the position of the dot and its neighboring symbols.
2. A horizontal line may be a fraction line or a minus sign depending on the length of the line and whether there are symbols above and below the line.
3. The same group of characters can sometimes have different meanings in different contexts. For example, "dx" is part of the integral notation in "$\int x^2 dx$" but it represents the multiplication of d and x in "$cy + dx$".

In addition, mathematical notation has many dialects. As analogous to natural languages, it is nearly impossible to design a universal grammar to cover all the dialects. As a result, almost all systems are based on a subset of the mathematical notation only.

3 Overview of the Recognition Process

Both symbol recognition and structure analysis of two-dimensional patterns have been extensively studied for decades. Mathematical expression recognition, which features both of them as the two major stages of the recognition process, is a good subject for studying the integration of the two areas.

Many symbol recognition techniques work under the assumption that the symbols have already been isolated from each other. If this assumption holds, recognition of symbols may simply use some existing method. However, the same situation does not happen in the structural analysis phase. The two-dimensional patterns in different domains usually have very different spatial relationships. Although some techniques used for recognizing other two-dimensional patterns may in principle be also applicable to mathematical expression recognition, such techniques usually require substantial modifications before they can be used. Moreover, some situations are very specific to mathematical expression recognition and they require specially designed methods.

When we write a mathematical expression on a tablet, what we get is a sequence of points. On the other hand, if we scan an expression from a printed document, what we get is a two-dimensional array of pixels. The data in the first case are usually regarded as on-line data while those in the latter case are off-line data. Intuitively, if we are able to segment the data into groups so that each group represents a single symbol, we can then directly apply an existing symbol recognition method to decide its identity. Afterwards, a list of objects with associated attributes (including location, size, and identity) is returned. Finally, we then apply some structural analysis techniques to obtain the hierarchical structure of the expression. Fig. 1 depicts such an intuitive recognition process.

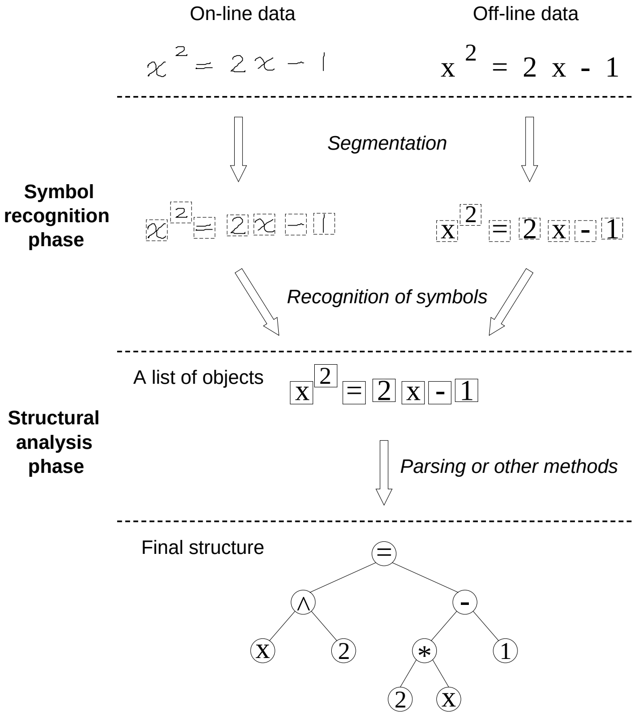

Fig. 1. Overview of an intuitive recognition process

Although the above process is quite commonly used, others may use different approaches in recognizing mathematical expressions. For example, several systems perform structural analysis prior to symbol recognition [5, 51], and some systems perform simultaneous segmentation and recognition of mathematical expressions [12,33, 34].

In the following sections, we will review some existing work for each major stage of the recognition process. In particular, we will highlight the similarities and differences between different approaches or systems.

4 Symbol Recognition

After more than three decades of research, many existing symbol recognition techniques are able to achieve quite satisfactory results. However, many of them can only work with isolated symbols. In a mathematical expression, there usually exist multiple symbols. Before we can apply these symbol recognition techniques, we must first segment the individual symbols from the expression properly.

4.1 Segmentation of Symbols

A naive way to segment symbols is to put all physically separate components (or called connected components) into groups. However, some characters and symbols, such as 'i', 'j', and '=', are composed of multiple components.

As a result, we have to combine the corresponding components together first before we can recognize the individual characters or symbols correctly. In more complicated situations, some symbols, like '$\sqrt{\ }$', usually contain other symbols inside their effective regions. Hence, caution must be taken when segmenting such symbols.

Faure and Wang [18] presented a modular system for segmenting handwritten mathematical expressions. Their system has two segmentation modules. The data-driven segmentation module first builds a relation tree of the given expression. In general, projections on the X and Y axes can be used for deciding how to segment the data. However, for symbols like '$\sqrt{\ }$' and the fraction line, this approach often fails. A mask removal operation is thus applied to segment these symbols and their embedded symbols before projection operations are carried out for the embedded symbols. Afterwards, the knowledge-driven segmentation module attempts to correct the relation tree built by the previous module. For example, it tries to combine different parts together for symbols like 'i', 'j' and '='.

Okamoto *et al.* [51, 52] proposed to partition a given printed expression into components by recursive horizontal and vertical projection profile cutting. Some additional checking steps are required for symbols which contain separate elements (e.g., 'i', 'j' and '=') and symbols which contain some other symbols within their regions (e.g., '$\sqrt{\ }$').

Ha *et al.* [22] used the bounding boxes of the symbols as the clue for extracting them from a printed expression. Their method is called "recursive X-Y cut" in which 'X' refers to the horizontal cut and 'Y' refers to the vertical cut. This method is similar to the projection profile cutting method except for the primitive objects used for projection (one uses pixels and the other uses bounding boxes).

Smithies *et al.* [58] developed a simple progressive grouping algorithm for on-line symbol segmentation. Their system first generates all possible groupings for a small number of strokes (according to a small upper bound). It then looks for the one with the maximal confidence level given by the character recognizer. This algorithm is simple and fast but may sometimes introduce errors that require manual correction.

In general, most of the algorithms work quite well. However, just like other segmentation problems, these algorithms often rely on the use of thresholds. In practice, threshold values cannot be chosen to work well on all possible inputs.

4.2 Recognition of Segmented Symbols

After the segmentation step, we have a list of objects with some known attribute values. The only missing values are the identities of symbols. In theory, we can apply any symbol recognition method as long as it is designed for the corresponding data type (i.e., on-line or off-line).

Over many years of research, different approaches have been proposed for symbol recognition, including template matching, structural, neural network and other statistical approaches. Surveys of these approaches can easily be found in the literature [47, 60]. Here, we do not intend to repeat what have been done. Instead, we will just list some typical systems by category according to the symbol recognition approach used:

1. Template matching approaches:
   Several systems, such as Nakayama [49] and Okamoto *et al.* [51, 52], make use of some traditional template matching methods. Others perform template matching based on different measures, e.g., Fateman *et al.*[6, 16] and Miller and Viola [46] used Hausdorff distance.
2. Structural approaches:
   Not many systems are based on structural approaches. A few exceptions are Beláid and Haton [5] and Chan and Yeung [8].
3. Statistical approaches:
   Quite a number of systems, including Chen and Yin [11], Fateman *et al.* [17], Fukuda *et al.* [19], Lee *et al.* [39–42], and Smithies *et al.* [58], are based on traditional statistical approaches. Others, such as Dimitriadis and Coronado [14], Ha *et al.* [22], and Marzinkewitsch [45] use neural networks.

4.3 Simultaneous Segmentation and Recognition of Symbols

All methods discussed in the previous section require the symbols to be segmented before the recognition step. However, some methods, such as those based on *hidden Markov models (HMMs)* [56], do not have this restriction. The HMM approach has proven to be very effective in the area of speech recognition. Some researchers thus attempted to apply this approach to recognize symbols in mathematical expressions.

As mentioned above, the on-line data of a mathematical expression is simply a sequence of points. This is analogous to the case for speech except that speech is a sequence of acoustic signals. Hence, HMM techniques developed for speech recognition can easily be modified for recognizing symbols in on-line mathematical expressions and to achieve simultaneous segmentation and classification.

Winkler *et al.* [32, 43, 64, 66, 67] first generated *symbol hypotheses net (SHN)* for the handwriting input and then used HMMs to find one or more symbol sequences from the SHN. The final classification of the symbols is done by finding the most probable symbol sequence. By keeping all alternatives for the solution, decision making can be delayed. Such technique is called a *soft-decision* approach. Sakamoto *et al.* [57] also used the HMM approach for recognizing characters and symbols in a mathematical expression.

4.4 Summary of Symbol Recognition Methods Used

Besides categorizing different systems according to the symbol recognition approach used, we may also group the systems according to the data type required. Table 1 shows such a categorization.

Table 1. Categorization of symbol recognition methods used in different systems according to the data type required

Data type	Major method	Example
On-line	Structural feature extraction and decision tree classification	Beláid and Haton [5]
	Flexible structural matching	Chan and Yeung [8]
	Feature extraction and nearest neighbor classification	Chen and Yin [11], Fukuda *et al.* [19], Smithies *et al.* [58]
	ART-based neural architecture and elastic matching	Dimitriadis and Coronado [14]
	Hidden Markov model	Winkler *et al.* [32,43,64,66,67], Sakamoto *et al.* [57]
	Three-layered backpropagation network	Marzinkewitsch [45]
	Traditional template matching	Nakayama [49]
Off-line	Template matching based on Hausdorff distance	Fateman *et al.* [6,16], Miller and Viola [46]
	Feature extraction and nearest neighbor classification	Fateman *et al.* [17], Lee *et al.* [39–42]
	Feature extraction and classification through neural network approach	Ha *et al.* [22]
	Traditional template matching	Okamoto *et al.* [51,52]

Note that the set of symbols used in mathematical expressions can sometimes be very large. Hence, some researchers focus on only a subset of them. For example, Zhao *et al.* [68] analyzed the structure of 94 commonly used mathematical symbols and discovered that they are all based on 10 basic elements. Several techniques, such as basic element ordering and reduction of number of standard symbols, have been applied to increase the recognition rate.

5 Structural Analysis

For those systems that perform symbol recognition before the structural analysis phase, we should have obtained a list of objects with associated attributes, such as their location, size, and identity. The next task is to build a hierarchical structure for the objects. The hierarchical structure may be represented as a parse tree or a relation tree.

However, some nodes of the tree may be missing in the list of objects obtained from the previous phase due to the existence of spatial operators in mathematical expressions. Hence, we should first identify all the spatial operators in order to build sub-structures over them and their operands. With all the intermediate sub-structures and the remaining objects, a final structure can then be constructed.

Some researchers in the area of mathematical expression recognition are interested only in the structural analysis phase. Therefore, they bypass the symbol recognition step entirely by assuming that perfect results are always available before the structural analysis phase. On the other hand, some researchers attempt to build complete systems. Some of the systems work for on-line data while others work for off-line data.

5.1 Identification of Implicit Operators between Symbols

In mathematical expressions, the type of spatial operators between symbols is determined based on the relative positions of symbols. In most cases, it involves all the associated attributes of a symbol, especially the center of the symbol, which often refers to the typographical center. Fig. 2 shows the typographical centers of three major types of symbols.

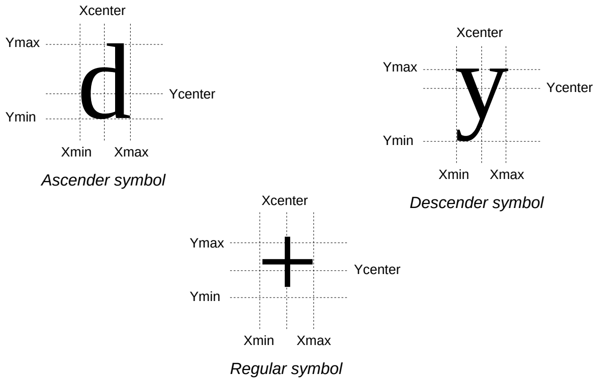

Fig. 2. Typographical centers for different types of symbols

Sometimes, simply based on the relative placement of typographical centers of symbols, we can already determine their spatial relationship, i.e., in-line, subscript, or superscript. We may further decide its corresponding association, namely, implicit multiplication, subscripting, or exponentiation, respectively. Such simple technique has been widely used (e.g., in [2,11,51]).

Wang and Faure [63] proposed a method for automatic labeling of spatial relationships between symbols. The method is designed to make judgment even before knowing the identities of the symbols involved. In other words, it only uses information about the bounding boxes of symbols. This method is certainly useful when the symbol identities are unknown (either because structural analysis is performed before symbol recognition, or due to some ambiguous or failed results). However, in general, using the spatial relationships without knowing the symbol identities may not be sufficient for some cases. In fact, Wang and Faure have pointed out that bounding boxes alone are in some cases ambiguous in terms of revealing the spatial relationships between symbols. Fig. 3 demonstrates that the same configuration of bounding boxes may reveal different spatial relationships. Nevertheless, work done by Wang and Faure is still useful for deciding spatial relationships between ambiguous or unknown symbols.

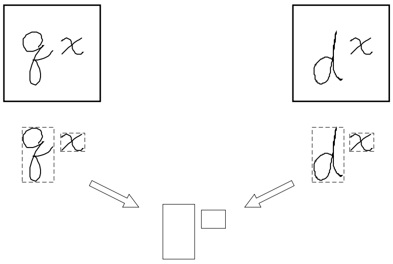

Fig. 3. Are bounding boxes alone sufficient?

Identification of spatial relationships between symbols is indeed quite complicated in some cases. For example, when there exist more than two symbols in an expression, spatial relationships among symbols can only be determined globally. In most cases, we have to locate the first symbol before other relationships can be identified accordingly. Fig. 4 shows an example. Several systems [18,20,29,41,42,62] have made extra effort in dealing with these problems.

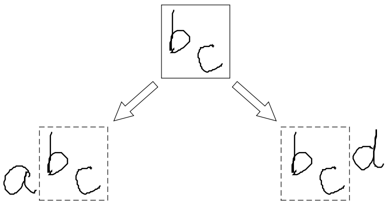

Fig. 4. Subscript and superscript relationships cannot be determined locally

Superscript and subscript expressions usually appear in the upper right and lower right areas of another symbol. However, there are exceptions, for example, "${}_nC_k$" where a subscript expression appears in the lower left area, and "${}^m\prod$" where a superscript expression appears in the upper left area [62]. If the domain of a recognition system includes this type of binding symbols, special care must be taken.

5.2 Previous Work on Structural Analysis Only

Some researchers are mainly interested in the structural analysis of two-dimensional patterns with mathematical expressions being a special subset. As a result, they bypass the symbol recognition stage entirely by assuming that all the characters and symbols have been recognized correctly.

One of the earliest papers on mathematical expression recognition was written by Anderson [2]. He used a purely top-down approach for parsing mathematical expressions. The algorithm starts with one ultimate syntactic goal and tries to partition the problem (i.e., goal) into sub-goals, until either all sub-goals have been satisfied or all possibilities have failed. The algorithm is syntax-directed since it is guided by some grammar rules. However, as stated by Anderson, experiments show that the algorithm is not very efficient due to the partitioning strategy used for the rules, which involve two nonterminal symbols on the right-hand side. As a result, up to $n-1$ partitions can be generated by a set of n characters, and each of these partitions may further generate more partitions. Nevertheless, work by Anderson (especially the use of syntactic rules in guiding the recognition) contributed greatly to mathematical expression recognition in particular and to syntactic pattern recognition in general.

Chang [10] proposed a structure specification scheme for the structural analysis of two-dimensional mathematical expressions. The algorithm mainly makes use of the ideas of operator precedence and operator dominance. It consists of two major steps, namely, grouping operator sequences and building the structural tree. Efficiency is taken into consideration in the proposed algorithm. However, the scheme can only be applied to patterns whose structures are based upon a number of operators. In addition, the algorithms described are quite tedious. It is thus not straightforward to understand how they actually work in practice.

Faure and Wang [18] designed a modular system to recognize handwritten mathematical expressions by building a relation tree. There are mainly two modules, namely, the data-driven module and the knowledge-driven module. One special feature of the system is that it will still work even when some symbols of the expression cannot be recognized.

Pfeiffer [55] designed a parser using generalized two-dimensional context-free grammar to parse two-dimensional structures like mathematical expressions. However, all discussions in that paper are limited to parsing in a theoretical sense with no real examples and experimental results shown.

Grbavec and Blostein [20] used a graph rewriting approach to recognize mathematical expressions. Their model includes four phases:

1. The *build* phase constructs edges to represent spatial relationships between symbols.
2. The *constrain* phase applies domain knowledge to remove contradictions and resolve ambiguities.
3. The *rank* phase uses information about operator precedence to group symbols into subexpressions.
4. The *incorporate* phase interprets subexpressions.

In addition, their system makes use of knowledge about notational conventions, such as operator precedence and operator range, to eliminate the need for backtracking.

Twaakyondo and Okamoto [62] proposed a new method to overcome some over-cutting problems (thus produce incorrect segmentation results) caused by recursive projection profile cutting of mathematical expressions. Two basic strategies are used to decide the layout structure of the given expression. One strategy is to check the local structures of the subexpressions using a bottom-up method (specific structure processing). It is used to analyze nested structures such as subscripts, superscripts, and root expressions. The other strategy is to check the global structure of the whole expression by a top-down method (fundamental structure processing). It is used to analyze the horizontal and vertical relations between subexpressions. The structure of the expression is represented as a tree structure.

Pagallo [53,54] applied constrained attribute grammars to the recognition of multi-dimensional objects, like mathematical expressions. The method first labels some important symbols (e.g., operators) as keywords and then applies certain relevance measure (which is similar to operator precedence) among keywords to guide the parsing and avoid expensive backtracking. However, no testing results are provided.

Lavirotte and Pottier[37,38] defined a class of context-sensitive graph grammars for mathematical expressions. The method is based on a *critical pairs* approach in the sense of the Knuth-Bendix algorithm. By adding contexts into the grammar rules, parsing may become more efficient because no backtracking is needed as a result. However, restrictive assumptions have to be made about symbol placement, and the availability of accurate point size information.

5.3 On-Line Mathematical Expression Recognition Systems

Due to its potential for use as a more natural alternative for entering mathematics into computers, on-line mathematical expression recognition has attracted more attention recently.

Beláid and Haton [5] used two syntactic parsers, i.e., top-down and bottom-up, in the structural analysis phase in order to parse the expressions in a more concise fashion. After recognizing the symbols with a structural approach, top-down parsing is used to divide an expression into sub-expressions and bottom-up parsing is then applied to combine sub-structures into a bigger structure. However, their experiments were only performed on some simple mathematical expressions (arithmetics and some trigonometric functions).

Marzinkewitsch [45] used a graph reduction approach to parse two-dimensional mathematical expressions and translated them into their corresponding string form. The process is guided by a context-free graph grammar. Note that both spatial and temporal features are taken into consideration during the process.

Chen and Yin [11] proposed an on-line handwritten mathematical expression system with not much emphasis put on the structural analysis part. In order to display an expression at the end, only a symbol relation tree that keeps all the spatial relationships between symbols is built. Hence, the major task that the system has to perform is symbol recognition. First, all the symbols are classified by a traditional statistical approach (looking for the nearest neighbor based on a different set of features). Then, contextual information may be used for deciding the final identity of the symbol if ambiguities occur. In addition, an on-line editor for manual correction is provided in case there still exist ambiguous or misclassified symbols.

Nakayama [49] developed a pen-input mathematical formula editor to simplify the problem of entering expressions into a computer. The system allows the user to enter characters and symbols in any order. It uses a pattern matching algorithm to recognize the handwriting. Similar to the previous approach, this system does not require parsing to recognize the structure. Basically, all information about the characters and symbols is kept in a table. During the display of the expression, objects are then checked from left to right, up to down and then translated into character strings. Note that the system imposes certain restrictions to ensure better performance. For example, all characters should be written in a size less than 32 by 32 pixels; otherwise, they will be treated as mathematical symbols or gestures. All components in a superscript must be above the central line. In addition, after the symbol recognition phase, all symbols will be transformed into their corresponding printed form. Certain spacing will be inserted between symbols.

Dimitriadis *et al.* [14,15] also designed a mathematical editor. The system first uses a neural network approach based on *adaptive resonance theory (ART)* for recognizing characters and symbols and then applies an attribute grammar for parsing the structure. In particular, extra effort is made in detecting and correcting errors as, according to the authors, no attempts were made previously in this aspect. Another special feature of the editor is that it can adapt to the writing habits of individual users.

Winkler *et al.* [65] applied a soft-decision approach to both the symbol recognition and structure analysis phases. With the soft-decision approach, the system always provides the user with a set of possible solutions. First, an approach based on HMMs is used to perform simultaneous segmentation and recognition of symbols. The resulting symbols are then sorted and put into groups accordingly. Based on a directed graph which keeps all possible relationships between symbols, a set of alternative answers is created. Finally, the user may choose to verify the answers manually or by some existing mathematical software.

Chan and Yeung [9] developed an on-line mathematical expression recognition system using a structural and syntactic approach. The system first applies a structural method, called flexible structural matching, to recognize the symbols. It then uses a syntactic method, called hierarchical decomposition parsing, to obtain the structure of a mathematical expression. The proposed syntactic method is based on three key ideas, namely, left-factoring, binding symbol preprocessing, and hierarchical decomposition, for making the parsing process more efficient.

Fukuda *et al.* [19] introduced the concept of "Mathematical Element" (ME), which is a set of several characters and symbols. By using some penalty functions based on different mutual spatial relationships between symbols and characters, a number of penalty values are calculated accordingly from different possible configurations of MEs in the target mathematical expression. Finally, the one with the minimal penalty value will be treated as the correct relation.

5.4 Off-Line Mathematical Expression Recognition Systems

Mathematical expressions are often part of a document, especially for documents in scientific and engineering disciplines. Hence, off-line mathematical expression recognition must be achieved before realistic document processing systems can be built for such disciplines.

Okamoto *et al.* [51,52] often emphasized that recognition of a wide variety of mathematical expressions can be done by using only the layout structures of symbols without actually parsing them. The system first applies recursive projection profile cutting to segment the characters and symbols and at the same time build a relation tree. Afterwards, a traditional template matching method is used for the recognition of symbols.

Lee and Lee [39,40] proposed a system for recognizing printed mathematical expressions. First, the system uses a traditional statistical approach to recognize individual characters and symbols. It then applies a procedure-oriented method to translate the expressions from two-dimensional structures into one-dimensional character strings.

Ha *et al.* [22] defined an expression tree as the abstraction of a mathematical expression. After applying the X-Y cut to segment the characters and symbols, the individual objects are recognized using a neural network approach. The construction of the expression tree can be done through top-down (finding all primitive objects) and bottom-up (resolving spatial relationships among objects) processes.

Fateman *et al.* [6,17,16] developed a prototype system that can properly translate noise-free typeset mathematical expressions into Lisp expressions. For the symbol recognition part, different methods are used, such as calculating the Hausdorff distance and computing the gray-value of the scaled character. For the structural analysis part, a simple recursive descent parser is employed. Experiments show that the original bottom-up design is of limited use in the face of noisy data. Hence, a more semantic top-down approach may later replace it for achieving higher levels of performance.

Lee and Wang [41,42] presented a system for segmenting and understanding text as well as mathematical expressions in a document. In understanding the expressions, some feature extraction techniques and a nearest-neighbor algorithm are used to recognize the characters and symbols. A symbol relation tree is then built for representing an expression. In addition, some heuristics are used to correct recognition errors.

Miller and Viola [46] took a relatively new approach for the recognition of mathematical expressions. They used convex hulls for grouping symbols and applied A* search to handle the exponential search space. During the search, a probability is assigned to each interpretation of the given expression. Such a probability depends on the character and symbol models, the context free grammar used, and the probabilistic geometry rules. The system then attempts to find the most probable interpretation. At the end, contextual information is also used to improve the recognition performance.

Inoue *et al.* [29] put quite a lot of emphasis on correctly finding the baseline of the expression. After this important first step, other subscript area, superscript area, and nested substructures can be identified recurrently.

5.5 Simultaneous Segmentation and Recognition of Mathematical Expressions

As mentioned in Section 4.3, HMMs can be used to simultaneously segment and recognize symbols in mathematical expressions. Some researchers even attempted to apply the techniques to the expressions directly.

Chou [12] proposed to use a two-dimensional stochastic context-free grammar for the recognition of printed mathematical expressions. His approach was designed for handling noise and random variations. In the grammar, each production rule has an associated probability. The main task of the process is to find the most probable parse tree for the input expression (an image). Note that pixels are the only terminal symbols in this grammar. In order to map nonterminal characters to pixels, the system uses Hamming distance to compare rectangular arrays of pixels at each location of the image against templates in the font dictionary and obtain their corresponding probability values. At the end, the overall probability of a parse tree will be computed by multiplying together the probabilities for all production rules used in a successful parse. As a consequence, the process is computationally quite expensive.

Note that Chou later in collaboration with Kopec extended his work to recognize documents. Their work [13, 31] has become very influential in document recognition. However, due to the scope of this paper, we will not cover them here. For those who are interested, please refer to these papers for details.

Kosmala and Rigoll [33,34] also used the HMM approach to segment and recognize expressions simultaneously. However, they assumed that the user always writes the expression in a certain order, e.g., when writing a fraction, the numerator should be written first, then followed by the fraction line, and finally the denominator. Such requirement can easily be violated in real applications due to the high variability of writing styles between different writers. In addition, only one level of superscripting or subscripting is allowed. In other words, the system cannot recognize some simple expressions such as a^{b^c}. The above constraints were lifted when a graph grammar was later adapted to the original system [35].

5.6 Summary of Structural Analysis Methods Used

As mentioned in the previous sections, quite a number of mathematical expression recognition systems obtained the structure without parsing. Instead, some procedurally-coded rules were used while others applied parsing techniques with a range of variations. Table 2 shows a summary.

However, it should be noted that most methods for the structural analysis of mathematical expressions are actually based on some kinds of syntax [7]. Whether to define the syntax implicitly or explicitly is just a matter of choice.

For simple expressions, both ways should do the job well. The situation changes if we try to recognize more complex expressions. Rather than adding many *ad hoc* procedurally-coded rules to the system and yet still being uncertain of the correctness of the structural analysis module, explicit rules in a parser may provide a clearer and more concise form for formal verification.

So far, some parsers in existing mathematical expression recognition systems work on string grammars while others work on two-dimensional grammars, e.g., graph grammars. Before we decide what kind of grammar to use, we should know that the expressive power of a grammar can strongly affect the complexity of its parser. In general, graphs have higher expressive power, but they require a more complicated parser. In contrast, the expressive power of string grammars is lower, but their parser usually only has polynomial time complexity.

6 Other Issues

Besides those issues related to the recognition process, many others have been raised in different papers. Here, we will only cover some of the important ones, such as ambiguity resolution, error detection and correction, and performance evaluation.

6.1 Ambiguity Resolution

Mathematical notation is meant to be unambiguous. However, ambiguities may occur when the expressions are not typeset or written properly. This problem was first brought up by Martin [44]. However, no solutions were provided.

Beláid and Haton [5] suggested to use contextual information for resolving ambiguities. They further provided some positive examples where ambiguities can be resolved, namely, multiple answers and misrecognition, as well as some negative examples where ambiguities cannot be resolved, such as confusion. Chen and Yin [11] also discussed some contextual constraints that can be used to resolve ambiguities.

Some frameworks, especially those based on probabilistic models, are by nature good at dealing with ambiguous cases. One example is the two-dimensional stochastic context-free grammar [12]. Others include HMMs used by Winkler *et al.* [32,43,64–67] and Miller and Viola [46].

6.2 Error Detection and Correction

During the recognition of mathematical expressions, errors often occur. In general, there are four types of errors, namely, lexical, syntactic, semantic and logical errors [1]. Although error detection and correction are important steps, very few papers in the mathematical expression recognition literature have addressed these issues.

Dimitriadis *et al.* [14] claimed to be the first attempt in detecting and correcting errors in mathematical expressions. However, the error detection and correction methods used are quite simple. For example, some warning messages, such as "the root symbol should cover all of its terms", may be given when the error is not fatal. However, some other errors, like "the function `tan` does not have arguments", require the user to correct the input before the editor can proceed.

Lee and Wang [42] used some heuristic rules to correct lexical errors. For example, the expression "$x = 5in\ \theta$" will be converted to "$x = \sin\theta$" due to the similarity between '5' and 's'. Other heuristic rules are also used, such as

- For every binary operator P, there must exist two operands that will generally be of the same typeface and size.
- There are no symbols in the subscript position of a numeral.
- Symbols in the same operand generally possess the same properties.

6.3 Performance Evaluation of Recognition Systems

In the past, some researchers put their emphasis purely on theoretical aspects without any experimental results reported. For those who did conduct experiments, their performance evaluation methods can roughly be grouped into three major categories:

1. Performing the test on a set of expressions and categorizing the results according to whether the expressions are correctly or incorrectly recognized [5].
2. Performing the test on a set of expressions and paying attention only to the symbol recognition rate [11,14, 33,40,42].
3. Performing the test on some typical expressions [2, 51,62]. Such expressions are usually written neatly by one or just a few writers. As a result, all the expressions can be recognized correctly. The purpose of this kind of testing is to show that the method works at least for those typical expressions.

Table 2. Summary of structural analysis methods used in different systems based on the type of parsing required

Parsing required	Major method	Example
No parsing	Procedurally-coded rules for building a relation tree	Chen and Yin [11], Lee and Wang [41,42]
	Projection profile cutting for building a relation tree	Faure and Wang [18], Okamoto *et al.* [51,52]
	Determining the correct relation of an expression based on spatial relationships	Fukuda *et al.* [19], Inoue *et al.* [29]
	Recursive X-Y cut for building an expression tree	Ha *et al.* [22]
	Simultaneous segmentation and recognition through HMM approach	Kosmala and Rigoll [33,34]
	Procedurally-coded rules for converting 2-D sub-structures into corresponding linear form	Lee and Lee [39,40], Nakayama [49]
	Fundamental and specific structure processing for building a relation tree	Twaakyondo and Okamoto [62]
	Procedurally-coded rules with some probabilistic measures for building a directed graph	Winkler *et al.* [65]
Top-down parsing	Coordinate grammar	Anderson [2,3]
	Simultaneous segmentation and recognition through stochastic context-free grammar	Chou [12]
	Constrained attribute grammar	Pagallo [54]
Bottom-up parsing	Structure specification scheme	Chang [10]
	Attribute grammar	Dimitriadis *et al.* [14,15]
	Graph grammar	Grbavec and Blostein [20], Kosmala *et. al* [35], Lavirotte and Pottier [37,38], Marzinkewitsch [45], Pfeiffer [55]
	Chart parsing associated with probability vectors	Miller and Viola [46]
Top-down and bottom-up parsing	Description grammar	Beláid and Haton [5]
	Hierarchical decomposition parsing	Chan and Yeung [9]
	Converting 2-D sub-structures into corresponding linear form before performing recursive-descent parsing	Fateman *et al.* [16,17]

7 Discussions on Future Research Directions

Research on mathematical expression recognition started in the 1960s. However, only very few papers in this area were published in the 1970s and 1980s. It is only until recently that mathematical expression recognition has attracted more attention from the research community. Hence, there still exist a lot of research topics that deserve further investigation. Some of them are summarized and briefly discussed here:

1. New methods or designs -
   Mathematical expressions are constructed from a possibly very large set of characters and symbols and its notation has many dialects. All current systems have to impose some restrictions on the symbol set and the grammar used. Recognition systems can be developed for personal use (e.g., those installed on personal digital assistants (PDAs)) or for public access (e.g., those on web servers). For mobile devices, the memory capacity needed for running the system may become a practical concern. On the other hand, when we relax some restrictions, change the target user groups, or port the systems to new platforms, it may require modifications on existing methods or even new design to suit different needs.
2. Ambiguity resolution -
   Mathematical expressions are prone to ambiguities especially when they are not typeset or written properly. In some situations, ambiguous cases can be resolved. However, techniques proposed for ambiguity resolution in mathematical expressions are still at some early stage and it has room for further investigation (probably with more sophisticated use of contextual information).
3. Error detection and correction -
   Papers on error detection and correction in mathematical expression recognition are relatively rare. In most cases, it is still at the detection level, i.e., issuing warnings and reporting errors. More work should be done on correcting errors (again with the help of contextual information).
4. Performance evaluation of recognition systems -
   Mathematical expression recognition consists of two stages. Focus has been put heavily on the symbol recognition stage while performance evaluation of the structural analysis stages has not received sufficient attention. Schemes that provide a proper balance on both stages have yet to be proposed and studied.
5. Potential applications -
   Applications of mathematical expression recognition have mostly been in the form of editor programs [14, 15,49]. Only a few have attempted to design a handwriting interface for computer algebra systems [45]. Other potential applications, such as pen-based calculator programs and pen-based intelligent tutoring systems in mathematics for children, should be explored.

8 Conclusion

With the recent advances in pen-based computing and optical scanning technologies, we already have all the necessary hardware for entering mathematical expressions into computers based on either on-line or off-line data. The key problem that remains is the automatic recognition of mathematical expressions, which is more on the software side.

Mathematical expression recognition consists of two major stages, namely, symbol recognition and structural analysis. In this paper, we have reviewed most of the existing work according to each major stage of the recognition process. In symbol recognition, different methods, including structural, neural network and other statistical approaches, have been used. In structural analysis, some methods parse the mathematical expressions using explicit syntactic rules while others obtain the internal structure without parsing. In both stages, we particularly put our emphasis on the similarities and differences between systems. Note that several methods perform structural analysis before symbol recognition and some perform both stages simultaneously. Besides, we have also described work done regarding other important research issues. All these together serve to provide a clear overall picture of how this research area has been developed to date.

Apparently, some issues in mathematical expression recognition have not yet been fully addressed, such as resolving ambiguities, error detection and correction, performance evaluation, and potential applications. Moreover, more practical problems will emerge when we incorporate such mathematical expression recognition systems into real-world applications that use them.

Acknowledgements. This research work has been supported in part by the Hong Kong Research Grants Council (RGC) under Competitive Earmarked Research Grants HKUST 746/96E and HKUST 6081/97E awarded to the second author.

References

1. A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, Reading, MA, 1986.
2. R. H. Anderson. Syntax-directed recognition of hand-printed two-dimensional mathematics. In M. Klerer and J. Reinfelds, editors, *Interactive Systems for Experimental Applied Mathematics*, pages 436–459. Academic Press, New York, 1968.
3. R. H. Anderson. Two-dimensional mathematical notation. In K. S. Fu, editor, *Syntactic Pattern Recognition Applications*, pages 147–177. Springer-Verlag, New York, 1977.
4. ATCM'98. *Proceedings of the the Third Asian Technology Conference in Mathematics*, Tsukuba, Japan, 1998.
5. A. Beláid and J.-P. Haton. A syntactic approach for handwritten mathematical formula recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(1):105–111, Jan. 1984.
6. B. P. Berman and R. J. Fateman. Optical character recognition for typeset mathematics. In *Proceedings of the 1994 International Symposium on Symbolic and Algebraic Computation*, pages 348–353, Oxford, UK, July 1994.
7. D. Blostein and A. Grbavec. Recognition of mathematical notation. In H. Bunke and P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, pages 557–582. World Scientific, Singapore, 1997.
8. K. F. Chan and D. Y. Yeung. Recognizing on-line handwritten alphanumeric characters through flexible structural matching. *Pattern Recognition*, 32(7):1099–1114, July 1999.
9. K. F. Chan and D. Y. Yeung. An efficient syntactic approach to structural analysis of on-line handwritten mathematical expressions. *Pattern Recognition*, 33(3):375–384, Mar. 2000.
10. S. K. Chang. A method for the structural analysis of two-dimensional mathematical expressions. *Information Sciences*, 2(3):253–272, 1970.
11. L. H. Chen and P. Y. Yin. A system for on-line recognition of handwritten mathematical expressions. *Computer Processing of Chinese and Oriental Languages*, 6(1):19–39, June 1992.
12. P. A. Chou. Recognition of equations using a two-dimensional stochastic context-free grammar. In *Proceedings of the SPIE Visual Communications and Image Processing IV*, volume 1199, pages 852–863, Philadelphia, PA, Nov. 1989.
13. P. A. Chou and G. E. Kopec. A stochastic attribute grammar model of document production and its use in document image decoding. In *Document Recognition II, SPIE Proceedings Series*, volume 2422, pages 66–73, 1995.
14. Y. A. Dimitriadis and J. L. Coronado. Towards an ART based mathematical editor, that uses on-line handwritten symbol recognition. *Pattern Recognition*, 28(6):807–822, 1995.
15. Y. A. Dimitriadis, J. L. Coronado, and C. de la Maza. A new interactive mathematical editor, using on-line handwritten symbol recognition, and error detection-correction with an attribute grammar. In ICDAR'91 [25], pages 885–893.
16. R. J. Fateman and T. Tokuyasu. Progress in recognizing typeset mathematics. In *Proceedings of the SPIE*, volume 2660, pages 37–50, San Jose, CA, 1996.
17. R. J. Fateman, T. Tokuyasu, B. P. Berman, and N. Mitchell. Optical character recognition and parsing of typeset mathematics. *Journal of Visual Communication and Image Representation*, 7(1):2–15, Mar. 1996.
18. C. Faure and Z. X. Wang. Automatic perception of the structure of handwritten mathematical expressions. In R. Plamondon and C. Leedham, editors, *Computer Processing of Handwriting*, pages 337–361. World Scientific, Singapore, 1990.
19. R. Fukuda, S. I, F. Tamari, M. Xie, and M. Suzuki. A technique of mathematical expression structure analysis for the handwriting input system. In ICDAR'99 [28], pages 131–134.
20. A. Grbavec and D. Blostein. Mathematics recognition using graph rewriting. In ICDAR'95 [26], pages 417–421.
21. F. Grossman, R. J. Klerer, and M. Klerer. A language for high-level programming of mathematical applications. In

Proceedings of the International Conference on Computer Languages, pages 31–40, Miami Beach, FL, 1988.

22. J. Ha, R. M. Haralick, and I. T. Phillips. Understanding mathematical expressions from document images. In ICDAR'95 [26], pages 956–959.
23. ICASSP'95. *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, Detroit, MI, 1995.
24. ICASSP'96. *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 6, Atlanta, GA, 1996.
25. ICDAR'91. *Proceedings of the First International Conference on Document Analysis and Recognition*, Saint-Malo, France, 1991.
26. ICDAR'95. *Proceedings of the Third International Conference on Document Analysis and Recognition*, Montreal, Canada, 1995.
27. ICDAR'97. *Proceedings of the Fourth International Conference on Document Analysis and Recognition*, Elm, Germany, 1997.
28. ICDAR'99. *Proceedings of the Fifth International Conference on Document Analysis and Recognition*, Bangalore, India, 1999.
29. K. Inoue, R. Miyazaki, and M. Suzuki. Optical recognition of printed mathematical documents. In ATCM'98 [4], pages 280–289.
30. A. Kacem, A. Belaíd, and M. B. Ahmed. EXTRAFOR: Automatic EXTRAction of mathematical FORmulas. In ICDAR'99 [28], pages 527–530.
31. G. E. Kopec and P. A. Chou. Document image decoding using Markov source models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6):602–617, June 1994.
32. M. Koschinski, H.-J. Winkler, and M. Lang. Segmentation and recognition of symbols within handwritten mathematical expressions. In ICASSP'95 [23], pages 2439–2442.
33. A. Kosmala and G. Rigoll. On-line handwritten formula recognition using statistical methods. In *Proceedings of the Fourteenth International Conference on Pattern Recognition*, pages 1306–1308, Brisbane, Australia, 17–20 August 1998.
34. A. Kosmala and G. Rigoll. Recognition of on-line handwritten formulas. In *Proceedings of the Sixth International Workshop on Frontiers in Handwriting Recognition*, pages 219–228, Taejon, Korea, 12–14 August 1998.
35. A. Kosmala, G. Rigoll, S. Lavirotte, and L. Pottier. On-line handwritten formula recognition using hidden Markov models and context dependent graph grammars. In ICDAR'99 [28], pages 107–110.
36. L. Lamport. *$\LaTeX$ – A Document Preparation System – User's Guide and Reference Manual.* Addison-Wesley, Reading, MA, 1985.
37. S. Lavirotte and L. Pottier. Optical formula recognition. In ICDAR'97 [27], pages 357–361.
38. S. Lavirotte and L. Pottier. Mathematical formula recognition using graph grammar. In *Proceedings of the SPIE*, volume 3305, pages 44–52, San Jose, CA, 1998.
39. H.-J. Lee and M.-C. Lee. Understanding mathematical expressions in a printed document. In *Proceedings of the Second International Conference on Document Analysis and Recognition*, pages 502–505, Tsukuba Science City, Japan, 1993.
40. H.-J. Lee and M.-C. Lee. Understanding mathematical expressions using procedure-oriented transformation. *Pattern Recognition*, 27(3):447–457, 1994.
41. H.-J. Lee and J.-S. Wang. Design of a mathematical expression recognition system. In ICDAR'95 [26], pages 1084–1087.
42. H.-J. Lee and J.-S. Wang. Design of a mathematical expression recognition system. *Pattern Recognition Letters*, 18:289–298, 1997.
43. S. Lehmberg, H.-J. Winkler, and M. Lang. A soft-decision approach for symbol segmentation within handwritten mathematical expressions. In ICASSP'96 [24], pages 3434–3437.
44. W. A. Martin. Computer input/output of mathematical expressions. In *Proceedings of the Second Symposium on Symbolic Algebraic Manipulation*, pages 78–89, Los Angeles, CA, 1971.
45. R. Marzinkewitsch. Operating computer algebra systems by handprinted input. In *Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation*, pages 411–413, Bonn, Germany, July 1991.
46. E. G. Miller and P. A. Viola. Ambiguity and constraint in mathematical expression recognition. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 784–791, Madison, Wisconsin, 1998.
47. S. Mori, C. Y. Suen, and K. Yamamoto. Historical review of OCR research and development. *Proceedings of the IEEE*, 80(7):1029–1058, July 1992.
48. Y. Nakayama. Mathematical formula editor for CAI. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computer Systems*, pages 387–392, Austin, TX, Apr. 1989.
49. Y. Nakayama. A prototype pen-input mathematical formula editor. In *Proceedings of ED-MEDIA 93 - World Conference on Educational Multimedia and Hypermedia*, pages 400–407, Orlando, FL, 23–26 June 1993.
50. R. Narasimhan. Labeling schemata and syntactic descriptions of pictures. *Information and Control*, 7:151–179, 1964.
51. M. Okamoto and B. Miao. Recognition of mathematical expressions by using the layout structures of symbols. In ICDAR'91 [25], pages 242–250.
52. M. Okamoto and A. Miyazawa. An experimental implementation of a document recognition system for papers containing mathematical expressions. In H. S. Baird, H. Bunke, and K. Yamamoto, editors, *Structured Document Image Analysis*, pages 36–53. Springer-Verlag, Berlin, 1992.
53. G. M. Pagallo. Method and apparatus for processing graphically input equations. US Patent, 5,544,262, 1996.
54. G. M. Pagallo. Constrained attribute grammars for recognition of multi-dimensional objects. In A. Amin, D. Dori, P. Pudil, and H. Freeman, editors, *Advances in Pattern Recognition*, pages 359–365. Springer-Verlag, Berlin, 1998.
55. J. J. Pfeiffer, Jr. Parsing graphs representing two dimensional figures. In *Proceedings of the IEEE Workshop on Visual Languages*, pages 200–206, Seattle, WA, 1992.
56. L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, Feb. 1989.
57. Y. Sakamoto, M. Xie, R. Fukuda, and M. Suzuki. On-line recognition of handwriting mathematical expression via network. In ATCM'98 [4], pages 271–279.

58. S. Smithies, K. Novins, and J. Arvo. A handwriting-based equation editor. In *Graphics Interface*, pages 84–91, June 1999.
59. Y. Y. Tang and C. Y. Suen. Document structures: A survey. In H. Bunke, P. S. P. Wang, and H. S. Baird, editors, *Document Image Analysis*, pages 85–115. World Scientific, Singapore, 1994.
60. C. C. Tappert, C. Y. Suen, and T. Wakahara. The state of the art in on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(8):787–808, 1990.
61. J.-Y. Toumit, S. Garcia-Salicetti, and H. Emptoz. A hierarchical and recursive model of mathematical expressions for automatic reading of mathematical documents. In ICDAR'99 [28], pages 119–122.
62. H. M. Twaakyondo and M. Okamoto. Structure analysis and recognition of mathematical expressions. In ICDAR'95 [26], pages 430–437.
63. Z. X. Wang and C. Faure. Structural analysis of handwritten mathematical expressions. In *Proceedings of the 9th International Conference on Pattern Recognition*, pages 32–34, Rome, Italy, 1988.
64. H.-J. Winkler. HMM-based handwritten symbol recognition using on-line and off-line features. In ICASSP'96 [24], pages 3438–3441.
65. H.-J. Winkler, H. Fahrner, and M. Lang. A soft-decision approach for structural analysis of handwritten mathematical expressions. In ICASSP'95 [23], pages 2459–2462.
66. H.-J. Winkler and M. Lang. Online symbol segmentation and recognition in handwritten mathematical expressions. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 3377–3380, Munich, Germany, 1997.
67. H.-J. Winkler and M. Lang. Symbol segmentation and recognition for understanding handwritten mathematical expressions. In A. Downton and S. Impedovo, editors, *Progress in Handwriting Recognition*, pages 407–412. World Scientific, Singapore, 1997.
68. X. Zhao, X. Liu, S. Zheng, B. Pan, and Y. Y. Tang. Online recognition of handwritten mathematical symbols. In ICDAR'97 [27], pages 645–648.

Kam-Fai Chan received his B.Sc. degree from Radford University, M.Sc. degree from the University of South Carolina, and Ph.D. degree from the Hong Kong University of Science and Technology, all in computer science. He is currently a postdoctoral research associate in the Department of Computer Science at the Hong Kong University of Science and Technology. His major research interests include Chinese computing, human-computer interaction, logic programming and pattern recognition.

Dit-Yan Yeung received his B.Sc.(Eng.) degree in electrical engineering and M.Phil. degree in computer science from the University of Hong Kong, and his Ph.D. degree in computer science from the University of Southern California in Los Angeles. From 1989 to 1990, he was an assistant professor at the Illinois Institute of Technology in Chicago. He is currently an associate professor in the Department of Computer Science at the Hong Kong University of Science and Technology. His current research interests are in the theory and applications of pattern recognition, machine learning, and neural networks. He frequently serves as a paper reviewer for a number of international journals and conferences, including *Pattern Recognition*, *Pattern Recognition Letters*, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *IEEE Transactions on Image Processing*, and *IEEE Transactions on Neural Networks*.