Handbook on Optical Character Recognition and Document Image Analysis, pp. 000-000
Eds. P.S.P. Wang and H. Bunke

CHAPTER 22

RECOGNITION OF MATHEMATICAL NOTATION*

DOROTHEA BLOSTEIN† and ANN GRBAVEC

Department of Computing and Information Science

Queen's University, Kingston, Ontario, Canada K7L 3N6

Recognition of mathematical notation involves two main components: symbol recognition and symbol-arrangement analysis. Symbol-arrangement analysis is particularly difficult for mathematics, due to the subtle use of space in this notation. We begin with a general discussion of the mathematics-recognition problem. This is followed by a review of existing approaches to mathematics recognition, including syntactic methods, projection-profile cutting, graph rewriting, and procedurally-coded math syntax. A central problem in all recognition approaches is to find a convenient, expressive, and effective method for representing the notational conventions of mathematics.

Keywords: Computer recognition of mathematical notation; Notational conventions; Symbol recognition; Symbol-arrangement analysis; Syntactic methods; Projection-profile cutting; Graph rewriting.

1. Introduction

Over the centuries, people have developed a specialized two-dimensional notation for communicating with each other about mathematics. The notation is designed to represent ideas in a way that aids mathematical thinking and visualization. It is natural and convenient for people to communicate with computers using this same notation. This involves conversion between mathematical notation and internal computer representations. Under current technology, two-dimensional mathematical notation can be generated by computers, but recognition facilities are not widely available: the task of translating mathematics into a computer-processable form usually falls to a human user (Fig. 1). By relieving the user of the burden of translation, a mathematics recognition system enhances the usefulness of computers as a tool for mathematics and document-handling.

---

* This research is supported by Canada's Natural Sciences and Engineering Research Council.

† blostein@qucis.queensu.ca

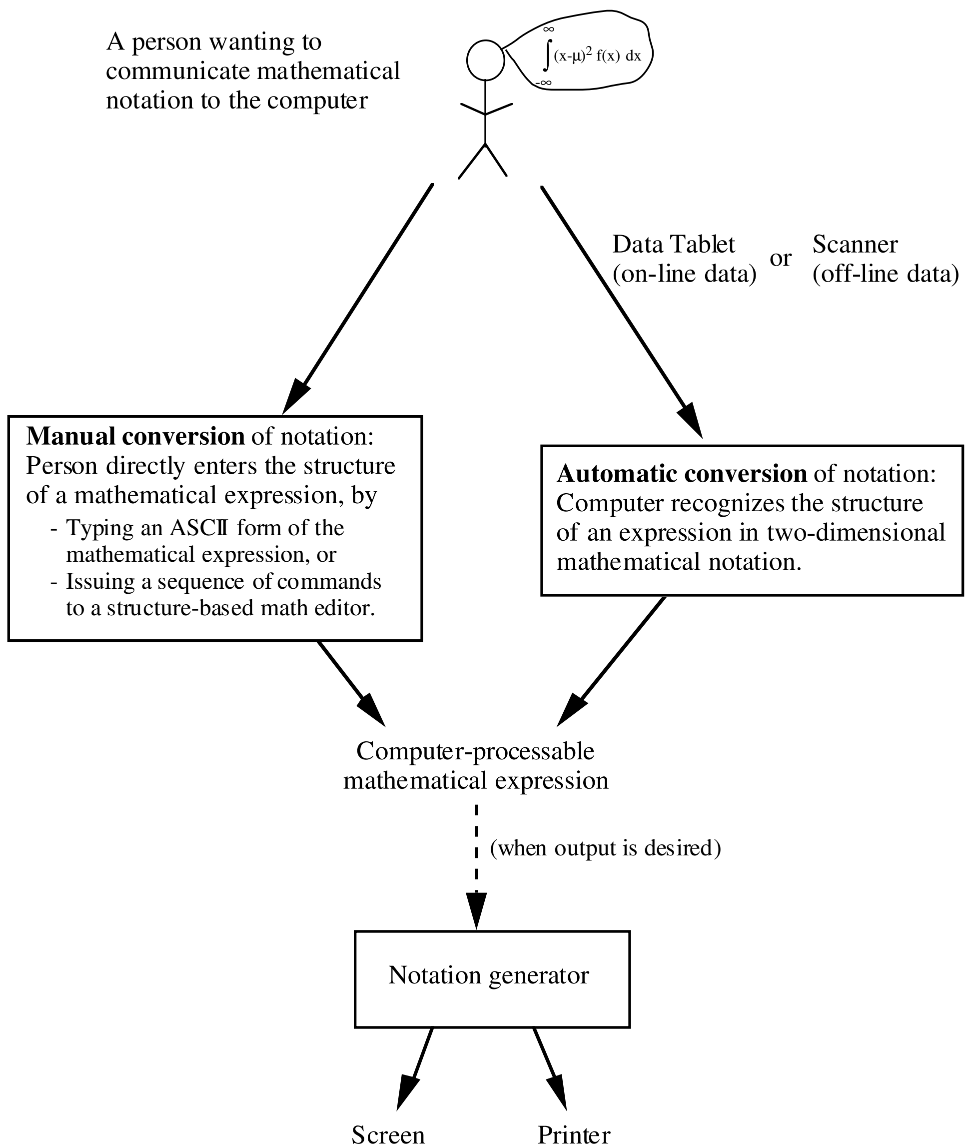


Fig. 1. Conversion between 2D mathematical notation (which people work with) and a computer-processable representation. Input of expressions is shown in the top part of the figure: the automatic-conversion pathway is desirable, but not yet widely available. Currently, most people who need to enter mathematical expressions use manual conversion (typing an ASCII form of the expression, or issuing editor commands to enter the structure of the expression [1]). In contrast, the technology for output of mathematical notation is mature and widely available (bottom part of the figure).

The requirements that must be met by a mathematics-recognition system depend on the application. Two main uses of recognition systems are as follows.

- On-line entry of mathematical notation, for a person creating a document for printing. A handwritten entry on a data tablet is one appropriate input means. A successful recognizer must be able to cope with the variability of handwriting; on-line timing information can be a great help.
- Off-line recognition of mathematical notation. Here, a previously typeset document is scanned; the mathematical expressions it contains must be located and then analyzed. This can be done for a variety of purposes. An example of small-scale use is a reading machine for the visually impaired. Large-scale use arises in the scanning and interpretation of a large collection of technical documents, for the creation of a database.

1.1. Six Processes for Mathematics Recognition

Mathematical notation uses two-dimensional arrangements of symbols to transmit information. Understanding a mathematical expression involves two (possibly concurrent) activities: symbol recognition and symbol-arrangement analysis. The former converts the input image into a set of symbols. The latter analyzes the spatial arrangement of this set of symbols (relative to the given notational conventions of the 2D language for expressing mathematics) to recover the information content of the given mathematical notation.

The two major recognition activities (symbol recognition and symbol-arrangement analysis) are performed by all mathematics-recognition systems. Subdividing further, the mathematics-recognition problem can be discussed in terms of six processes [2]:

$$\left.\begin{array}{l}\text{(1) early processing — noise reduction, de-skewing, etc}\\ \text{(2) segmentation, to isolate symbols}\\ \text{(3) recognition of symbols}\end{array}\right\}\ \text{symbol recognition}$$

$$\left.\begin{array}{l}\text{(4) identification of spatial relationships among symbols}\\ \text{(5) identification of logical relationships among symbols}\\ \text{(6) construction of meaning}\end{array}\right\}\ \begin{array}{l}\text{symbol-arrangement}\\ \text{analysis}\end{array}$$

These processes can be executed in series, or in parallel with later processes providing contextual feedback for the earlier processes. Any implementation must perform these recognition steps, although an implementation can mix steps together, so that the various processes are not clearly delineated in the code. The order of these recognition steps can vary somewhat. For example, partial identification of spatial and logical relationships can be performed prior to symbol recognition; this is done in projection-profile cutting (Sec. 3.2), and in Faure and Wang's structure recognition (Sec. 3.5).

For research purposes, it is possible to bypass the symbol-recognition step in order to concentrate on symbol-arrangement analysis. Test-data for the recognition system can be obtained by scanning a mathematical expression, and manually simulating the symbol-recognition step. This approach has been taken in [3] [4] [5] [6]. Here, perfect (error-free) symbol-recognition results are used to test the symbol-arrangement analysis method. To create a complete recognizer, one must: (1) supply an automatic symbol-recognizer, (2) modify the symbol-arrangement analysis so that it can handle errors and uncertainty from the symbol recognizer, and (3) possibly provide feedback from symbol-arrangement analysis to symbol recognition. For example, the work by Dimitriadis *et al.* [7] builds on Anderson's work [3] in this way.

1.2. Notational Conventions

Notational conventions play a central role in mathematics recognition. The notational conventions define the two-dimensional language, the mapping between the marks on the paper and the information that these marks represent. All six of the above recognition processes require knowledge of notational conventions. For example, noise reduction (1) needs knowledge of the notation in order to preserve small or thin symbols, such as the decimal point. Segmentation (2) needs information about how symbols can overlap. Symbol recognition (3) needs information about symbol appearance; perhaps a font defining fixed symbols, and structural descriptions for parameterized symbols such as the integrals and matrix brackets. The identification of spatial relationships (4) requires information about which spatial relationships are significant for encoding information. The identification of logical relations among symbols (5) can require extensive use of notational conventions; for example, large parts of an expression may need to be examined to determine whether a spatial relationship that looks like a subscript is coincidental, or whether it actually has the logical meaning *subscript*. Finally, the construction of meaning (6) relies heavily on the knowledge of notational conventions: this is the step in which the diagram is fully converted into the information it represents. Existing mathematics-recognition systems use a variety of methods to represent and apply notational conventions; these are summarized in Secs. 3 to 6.

1.3. Definition of Mathematical Notation

The first step in designing a mathematics recognition system is to define the recognition problem. We should begin with a definition of mathematical notation: a definition of the syntax and semantics of this two-dimensional language. Unfortunately, mathematical notation, like most diagrammatic notations, is not formally defined. It is only semi-standardized, allowing many variations and drawing styles. Here is a brief review of some sources of information about mathematical notation.

- **Written descriptions:** Various descriptions of mathematical notation have been published. Some presentations are oriented toward human readers who want to solve typesetting problems (e.g. [8] [9] [10]). Others are geared toward a computational treatment of typesetting problems (e.g. [11]). These descriptions are oriented toward generation of mathematical notation; they do not make explicit the knowledge of notational conventions that is needed to solve recognition problems.

- **Descriptions built into mathematics-recognition systems:** Existing mathematics-recognition systems use a variety of methods for representing notational conventions. These are reviewed below.
- **Descriptions built into mathematics-generation systems:** Generators of mathematical notation (which are usually bundled with editors) contain comprehensive, though often proprietary, descriptions of notational conventions.
- **Human experts:** Interviews with a highly-experienced user of mathematical notation can help in defining the notational conventions appropriate for a mathematics recognizer.

Several decades ago, Martin suggested that the first step in automating the processing mathematical notation is to make a study of the notation. He presents a brief list of the notational conventions found in use in technical publications ([12], p. 79).

To simplify the recognition problem, attention can be restricted to a certain style of mathematical notation. For example, during the development of Chou's system [13], testing was conducted with a large set of mathematical expressions, all drawn from a textbook which was known to be typeset in TROFF. This restriction to a certain style of mathematics typesetting increases the uniformity of notational conventions that the recognizer encounters. Clearly, recognition is easier if one knows the technology used to create the document being recognized.

Here we do not undertake a summary of the notational conventions of mathematics. Instead, we limit ourselves to a brief discussion of the factors that influence how mathematical symbols should be grouped during the recognition process. This grouping results from interpreting the appearance of a particular mathematical expression, relative to the notational conventions for mathematics. Relevant grouping factors include the following.

(1) Grouping factors defined for mathematical notation in general:
   - **Operator range:** The *range* of an operator defines the possible spatial locations for operands. The range for operators is established by conventional usage, and can vary from one typesetting style to another. For example, the bounds of an integral can appear above and below the integral symbol, or they can appear to the top-right and bottom-right of the integral symbol. Computationally, operator range can be defined in various ways, including grammar rules [3] [13], division rules in a structure specification scheme [4], and rules for applying projection-profile cuts [14]. Details of the definition of *operator range* vary from one author to another.
   - **Operator precedence:** Operator precedence is defined by a ranking of mathematical operators, to indicate the priority of evaluation of operations. Authors vary in their precise definition of operator precedence.

(2) Grouping factors arising within a particular mathematical expression:
   - **Symbol identity:** The identity of a symbol restricts how it can be grouped with other symbols. In most cases, symbol identity determines whether the

symbol is an operator or an operand. Once the identity of an operator symbol has been recognized, knowledge about the general grouping factors (operator range and operator precedence) is applied in the process of locating the operands.

- **Relative symbol placement:** Relative symbol placement is crucial to the identification of implicit mathematical operators. Whereas explicit operators are represented by a symbol (+ - Σ), implicit operators are indicated by the relative location of operands (implied multiplication, subscript, exponentiation). Eight basic spatial relationships are commonly used in mathematics-recognition systems: left, right, above, below, above right, above left, below right, below left (e.g. [15]). Identification of these relationships can be difficult; Sec. 5.1 discusses methods for distinguishing subscript, in-line, and superscript relations.
- **Relative symbol size and case:** Relative symbol size provides a clue for resolving ambiguous spatial relationships. This is particularly important for implicit operators. For example, a decreased font size provides evidence that the relation between symbols is subscript or superscript, rather than implied multiplication. Symbol size provides a cue for recognition, but not a hard constraint; this is particularly true for handwritten mathematical expressions, where symbol size can be erratic.

These factors (operator range, operator precedence, symbol identity, relative symbol placement, relative symbol size and case) interact in complex ways. Additional terminology can be useful in describing this interaction. For example, Chang's term *operator dominance* is defined for operators in the context of a particular expression. Operator A dominates over operator B if operator B's range nests completely within the range of operator A. Operator dominance provides a partial ordering on the evaluation of operations; operator precedence information is used to complete the ordering [4].

The grouping of symbols into subexpressions is a central problem in mathematics recognition. Below we review various computational approaches to solving this problem. Syntactic methods rely upon parsing to eventually determine the correct groupings (Secs. 3.1, 4.1). Projection-profile cutting exploits the existence of white space to efficiently extract (part of) the structure of an expression (Sec. 3.2). Graph-rewriting rules can be used to encode knowledge of operator precedence, operator range, and other symbol-grouping factors (Sec. 3.3). Procedurally-coded recognition rules can be used as well (Sec. 3.4). An interesting modularization of the mathematics-recognition problem is discussed in Sec. 3.5.

1.4. Problems in Recognizing Mathematical Notation

We now discuss problems that arise in the recognition of mathematical notation. It is interesting to compare the mathematics-recognition domain with other document-analysis domains. A few comparisons are mentioned below, but further work is needed to obtain a deeper understanding of the similarities and differences between mathematics-recognition and other document-recognition problems.

1.4.1. Noise versus small symbols

Various forms of preprocessing are common in document-image analysis. These serve to reduce noise, remove skew, and so on. In preprocessing mathematical notation, it is important to choose algorithms that will preserve small symbols, which are critical to the meaning. These include dots, commas, and symbol annotations such as A'.

1.4.2. Symbol segmentation and recognition

Symbol segmentation is relatively easy in printed mathematical notation, because there is little symbol overlap compared to other notations such as music ([16]; see also the chapter by Bainbridge and Carter in this book). Handwritten mathematical notation may contain many heavily-overlapping symbols; this can be very difficult to segment if only off-line data is available.

Symbol recognition in mathematical notation is difficult because there is a large character set (roman letters, greek letters, operator symbols) with a variety of typefaces (normal, bold, italic), and a range of font sizes (subscripts, superscripts, limit expressions). Certain symbols have an enormous range of possible scales (e.g. brackets, parentheses, Σ, Π, $\int$).

1.4.3. Ambiguity in the role of symbols

A characteristic of mathematical notation is that most symbols have a well-defined meaning. That is, “2” always means *two*, “+” always means *plus*, and so on. Compare this to the meaning that a line has in a road map; it could represent a road, a river, a fence, or a large number of other possibilities. However there are a few common symbols in mathematical notation that do feature such ambiguity in their role. A dot can represent a decimal point, a multiplication operator, a symbol annotation such as $\dot{x}$, or noise; a horizontal line can indicate a fraction line or a minus sign. The only way to determine the meaning of such symbols is through the contextual information provided by surrounding symbols. Thus, resolving ambiguity in role is a problem that must be addressed during symbol-arrangement analysis.

1.4.4. Identifying significant spatial relationships

Much of the meaning in mathematical notation is carried by the spatial relationships between the symbols. Although mathematics is a relatively standardized notation, considerable variation is permitted in relative symbol placement. In light of this variability, it is not clear how to define and identify meaningful spatial relationships among symbols in an expression. Spatial relationships are especially critical for the recognition of implicit mathematical operators; these are indicated by the spatial arrangement of operands, rather than by an explicit operator symbol. Superscripts, subscripts, implied multiplication, and matrix structure are indicated implicitly by the layout of operands, whereas addition, subtraction, and division use explicit operator symbols.

A difficult problem is to distinguish between configurations that represent horizontal adjacency and those that represent superscripts or subscripts. One source of difficulty is the continuum of positions between the horizontal adjacency and superscript (or subscript) configurations: 2x 2^x 2^x 2^x 2^x.

Mathematics interpretation is complicated by the fact that symbols in many different positions can potentially affect the meaning of a given symbol. There is a dense web of interdependencies among symbols. A large local neighborhood must be considered to interpret mathematical notation. From a given symbol, "adjacent" symbols may be relatively far away (say 5 to 10 times the symbol size, in every direction). This contrasts with music notation, where there are much stronger constraints on the location of relevant symbols.

1.4.5. Ambiguity of relative symbol placement

Mathematical notation uses subtle spatial relationships to indicate logical relationships: given a certain spatial relationship between two symbols, it can be quite difficult to determine the logical relationship between these symbols. We illustrate this with the subscript relation. (Keep in mind that we cannot locally deduce the baseline for a line of mathematical notation.) The configuration x_i could indicate that the symbol 'i' is a subscript of 'x' (as in the expression $x_i y_j$), or it could be a coincidental alignment (as in the expression $a^x i$). Martin provides an interesting series of examples of ambiguity arising from subtleties of symbol placement ([12], p. 83). Additional examples can be found in [6]. The ambiguity of spatial relationships is greatly increased for handwritten mathematical expressions; people take a lot of freedom with the placement and alignment of symbols. Proper resolution of symbol-relationship ambiguities can require contextual information from the entire expression.

1.4.6. Little redundancy in the notation

Mathematical notation has fairly little redundancy, when compared with other notations such as music [2]. Redundant representation of information provides a recognition system with constraints and cross-checks, particularly useful for the interpretation of noisy images. The relative lack of redundancy in mathematical notation means that less information is available for resolving symbol-recognition ambiguities.

1.5. Scope of Mathematics-Recognition Systems

Comparisons among mathematics-recognition systems are complicated by the fact that each system defines the recognition problem in its own way, placing different limitations on the types of input that should be recognizable. Definition of the recognition problem includes the following considerations.

- The mathematical notation may be handwritten or typeset. In case of handwritten notation, the input data may be on-line or off-line.

- Isolated mathematical expressions may be given as input, or the system may have to first locate mathematical expressions within a text page.
- Recognizers accept different sets of notational constructs. For example, some authors use *mathematical notation* to mean arithmetic mathematical notation, whereas others include matrix notation as well.
- Recognizers make various assumptions about the layout style of the notation. Some systems expect a very regular layout, assuming, for example, that a summation-limit-expression must lie within the x-extent of the Σ symbol. Other systems allow great flexibility in symbol placement.
- Some recognizers begin with the raw image as input, whereas others begin with symbols as input. The latter systems can be tested by human simulation of a perfect symbol recognizer. This human simulation sometimes includes lexical identification as well (e.g. using integers and floating-point numbers as input tokens [3]).

This concludes our introduction of the mathematics-recognition problem. We now turn to a survey of existing work in the area, organized according to major sub-parts of the problem: finding mathematical expressions in a document, methods for symbol-arrangement analysis (syntactic methods, projection-profile cutting, graph rewriting, and procedurally-coded rules), handling noise, identifying subtle spatial relationships, and recognition of handwritten expressions. Various factors are traded off in the design of a recognition system; these include efficiency, simplicity, readability, compactness, and generality.

2. Finding the Mathematical Expressions in a Document

Mathematical expressions are typically embedded in text documents, either as offset expressions, or embedded directly into a line of text. Thus, the first step in mathematics recognition is to identify where expressions are located on the page. (This does not apply to on-line recognition systems, where input comes from a person writing on a data tablet. In this case, text and equations generally are not mixed.)

Most work we survey assumes that the recognition system begins with an isolated mathematical expression. An exception is Lee and Wang, who present a method for extracting both embedded and offset mathematical expressions in a text document [17]. Text lines are labeled as offset expressions based both on internal properties and on having increased white space above and below them. The remaining text lines consist of a mixture of pure text and text with embedded expressions. These lines are converted to a stream of tokens. Certain tokens are recognized as belonging to an embedded mathematical expression; no details of this process are given, except that it is done “according to some basic expressions forms”.

3. Methods for Symbol-Arrangement Analysis

Existing recognition systems use a variety of approaches for representing the notational conventions of mathematics, and for using these to drive the recognition process. Here we review four approaches: syntactic methods, projection-profile cutting, graph-rewriting, and procedurally-coded rules.

3.1. Syntactic Methods

A common strategy for recognizing mathematical notation involves some form of two-dimensional grammar. Syntactic analysis is appealing for mathematical notation because the notation has an obvious division into primitives (i.e. symbols), a recursive structure, and a well-defined syntax (compared to other diagrammatic notations). Grammar rules are used to define the correct grouping of mathematical symbols, and to define the meaning of the resultant grouping.

3.1.1. Coordinate grammars

Anderson presents coordinate grammars for the recognition of commonly-used arithmetic and matrix mathematical notation [3]. This work provides an excellent example for illustrating the methods of syntactic pattern recognition. Since Anderson presents his complete grammars, a precise understanding of his system can be obtained, both at the abstract and detailed levels. Correctly recognized symbols and symbol-groups (integers and real numbers) are assumed as input.

A coordinate grammar operates on *sets* of symbols, rather than on the strings of symbols used in traditional formal language theory. Nonterminals and terminals of the grammar are attributed with bounding-box and center coordinates, as well as with a string m, an ASCII encoding of the meaning of the symbol-set. The final interpretation result is given by the m attribute of the grammar's start symbol.

Using a top-down parsing scheme, each grammar-rule application starts with a set of symbols and a syntactic goal (e.g. EXPRESSION, FACTOR, LIMIT). The grammar rule specifies how to subdivide the set of symbols into several subsets, each with its own syntactic subgoal. Partitioning conditions are used to describe spatial constraints. For example, a division rule (syntactic goal DIVTERM) subdivides the set of symbols into a horizontal bar, a set of symbols above the horizontal bar (syntactic goal EXPRESSION), and a set of symbols below the horizontal bar (syntactic goal EXPRESSION). If this partitioning fails (e.g. there is no horizontal bar, or some symbols are to the right or left of the horizontal bar) then the production rule reports failure, causing the parser to try another alternative. On the other hand, successful partitioning gives the parser two syntactic subgoals; if these succeed, then the division-rule's m value is constructed by appropriate combination of the subgoals' m values. Particular care is taken to find efficient partitioning strategies for the grammar rules describing implicit mathematical operations (implied multiplication, subscript, superscript): these rules do not have a terminal symbol (such as a horizontal bar or a plus sign) on the right-hand side, and must therefore try several partitioning alternatives.

In summary, a coordinate grammar provides a clear, well-structured approach to the recognition of mathematical notation. Slow parsing speeds can be a drawback. (Anderson points out that, in general, more efficient recognizers could be achieved by exploiting the special topological features of the notation and sacrificing some of the flexibility of a purely syntax-directed approach.) A major difficulty is to extend the syntactic approach to handle errors or uncertainty from symbol recognition; Sec. 4.1 describes Chou's work in this direction. Also, tests based on bounding-box coordinates provide a fairly crude recognition of spatial relationships. (For example, a limit expression under a Σ is required to have an x-extent that lies strictly within the Σ's x-extent.) Modifications or extensions of the coordinate-grammar formalism could be devised, to reduce restrictions on the symbol layouts permitted in recognizable expressions.

More recently, Belaid and Haton report on a recognition system using both a top-down and a bottom-up parser [18]. The coordinate grammar they use is simpler than Anderson's; a smaller subset of mathematical notation is analyzed, with emphasis on symbol recognition as well as symbol-arrangement analysis. In cases of uncertainty, the symbol recognizer provides lists of alternatives; contextual information applied during parsing can sometimes choose the correct alternative. For example, the parser can choose between "(" and "C". When the symbol alternatives can play similar syntactic roles (such as "S" and "5"), user interaction is necessary to solve the ambiguity.

3.1.2. Structure specification schemes

Chang uses *structure specification schemes* to recognize the structure of mathematical expressions [4]. This is a restriction of the syntactic approach, designed to be efficient in searching for the structure of an expression. Recognition time is $O(n^2)$, for an input expression consisting of n symbols. This time is for symbol-arrangement analysis: correctly-recognized symbols are assumed as input.

Structure specification schemes are based on the existence of operators, which divide the pattern into one or several sub-patterns. The scheme is specified as a set of division rules, one per operator, which indicate this spatial partitioning into subpatterns. Precedence numbers, given as part of a division rule, are used to resolve ordering of operators when neither operator dominates the other. Chang presents precise, detailed descriptions of his algorithms for applying division rules to find a well-formed structure in the given input. (A well-formed structure is equivalent to the parse tree of a syntactically-correct expression.)

A structure specification scheme has less expressive power than the picture processing grammars that Chang used previously. In other words, any structure specification scheme can be translated into a picture processing grammar that describes the same set of patterns; the inverse translation, from grammar to structure specification scheme, is not always possible. To offset this loss of descriptive power, there is the aforementioned increase in efficiency, as well as an increase in usability. (Chang states that a table of division rules is more concise, and easier to design and comprehend than a table of picture-grammar rules.)

Chang presents small-scale tests that illustrate the use of structure specification schemes for the recognition of mathematical notation, and for the recognition of document-layout structure (e.g. the reading order of paragraphs). Two thresholds, t and α, are used. Operators are classified as being on the same line if their centroids are within t. Also, an operator is regarded as being contained in a region if a certain fraction α of its total area (e.g. 70%) is contained in that region.

3.2. Projection Profile Cutting

Projection-profile cutting (also called *structural analysis*) is a technique that has been used in a variety of document-image analysis applications. In recognition of music notation, full-page projections are sometimes used to separate staves, with localized projections used to isolate particular music symbols [16]. Projection profiles have also been used for page-layout analysis, to divide a page into columns and paragraphs (see the chapter by Ha and Bunke in this book). Here we consider the use of projection-profile cutting to express the structure of a mathematical expression.

To analyze the structure of a mathematical expression, Okamoto *et al.* apply recursive projection-profile cutting [14] [19]. This method provides an analysis of symbol-layout structure, prior to the recognition of symbol identity, and without the use of an expensive parsing step. Cutting by the vertical projection profile is attempted first, followed by horizontal cuts for each resulting region. This process repeats recursively until no further cutting is possible. The resulting spatial relationships are represented by a tree structure. Special processing is used for square-roots, subscripts, superscripts; these cannot be handled by a projection profile cut. More extensive special processing is reported in subsequent work [6]. For example, they state that

> *There are many cases in which the symbols of a printed mathematical expression are printed very close to each other or there is no blank space in terms of vertical projection profile in the expression. For this reason the width of a blank space is calculated by deleting 10% of both sides of symbols, except in the case of horizontal bars of fractions or summation symbols which have an important role in the width of symbols ([6], pp. 435–436).*

Sec. 3.4 discusses this work further, in the context of procedurally-coded math syntax. For completeness, we give references to three mathematics-recognition papers written in Japanese, two of which originate from Okamoto's project [20] [21] [22].

Faure and Wang use X and Y projections in their processing of handwritten mathematical expressions [23]. They describe situations under which projection methods fail. These include square root symbols, closely-written symbols, and skew. They define mask-removal operations to address these situations (see Sec. 3.5).

J. Ha *et al.* propose another mathematics recognition system that uses a recursive X-Y decomposition [15]. Based on bounding-box attributes of primitives, an initial expression tree is constructed in a top-down fashion. In a bottom-up traversal, the initial tree is checked for syntax errors. Tree nodes are split or merged to fix syntax errors: this corrects for missing primitives and other errors. This work is in the design stage; an implementation is planned.

It appears that projection profile cutting is a simple and efficient technique, suitable for analyzing some aspects of mathematical-expression structure. A complete system must provide extensive additional processing to cover cases not amenable to projection analysis.

3.3. Graph Rewriting

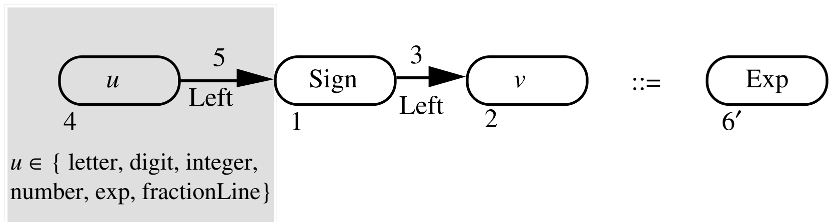
Graph rewriting is a general computational technique, in which information is represented as an attributed graph and computation proceeds by updating the graph through the application of graph-rewriting rules. (A graph-rewriting rule $g_l ::= g_r$ specifies that a subgraph isomorphic to g_l can be replaced by graph g_r. Further information is associated with the graph-rewriting rule, to specify how attribute-values for g_r are computed, and to specify the creation of edges between g_r and the main graph.) Graph rewriting has been applied to mathematics recognition as follows [5]. The existence of a symbol-recognizer is assumed: the input graph contains one node to represent each symbol, with node attributes recording the spatial coordinates of the symbol. Given this initial graph, which contains no edges, graph-rewriting rules are applied to add edges representing potentially-meaningful spatial relationships. Further application of graph-rewriting rules is used to prune and modify these edges, identifying the logically-important relationships from the spatial relationships. Information about operator precedence, stored as edge attributes, helps to determine how symbols are to be grouped into subexpressions. When a subexpression is recognized, the corresponding subgraph is replaced by a single node that represents the subexpression. The final output for a successfully-recognized expression is a single node whose meaning attribute represents the high-level meaning of the input expression as a character string. Fig. 2 illustrates a sample graph-rewrite rule.

The recognition process in [5] is organized into four phases.

Build Add edges between symbols that are related by potentially-meaningful associations. Edges are labeled Above, Below, Left, Superscript, Subscript.

Constrain Apply knowledge of notational conventions to remove contradictions and resolve ambiguities. This includes:

- Remove contradictory associations; remove the more distant of two similar associations
- Disambiguate horizontal lines as fractions or minus signs
- Disambiguate dots as decimals, multiplication signs, or noise
- Disambiguate diagonal associations — exponents and subscripts



Application Condition ≡ ($v \in$ { Letter, Digit, Integer, Number, Exp }) & (prec_in(3)=1)

Embedding ≡ In_{Left} = {(1, 6')}, Out_{Left} = {(2, 6')},
$\text{Out}_{\text{Above}}$ = {(1, 6'), (2, 6')}, $\text{Out}_{\text{Below}}$ = {(1, 6'), (2, 6')},
In_{Super} = {(1, 6')}, $\text{Out}_{\text{Super}}$ = {(2, 6')},
In_{Sub} = {(1, 6')}, Out_{Sub} = {(2, 6')},

Attribute Transfer ≡ {m(6') = m(1) (m(2)); reset_prec(6') }

Fig. 2. A graph-rewrite rule to interpret a signed item as a subexpression. The following steps are used to apply this graph-rewrite rule to a host graph.

1. Locate g_l^{host}, a subgraph of g that is isomorphic to the unshaded part of the left-hand side of the production rule. (If no isomorphic subgraph can be found, report failure of rule application.) This particular rule looks for a node with a Sign label that has a Left-labeled edge connecting it to some node with a variable label denoted v.
2. Test the application conditions. If they fail, report failure of rule application. This rule has both textual and graphical application conditions. The textual application conditions test that the label v is a member of the set {Letter, Digit, Integer, Number, Exp}, and that the precedence attribute prec_in of the Left edge has the value 1. The graphical application condition uses shaded region(s) to specify host-graph structure that must be *absent* for rule application to occur. In this case, rule-application is prohibited if there is an operand in front of the Sign node, since in that case the Sign should be interpreted as an addition or subtraction, rather than as a negation or explicit plus sign.
3. Remove g_l^{host} from the host graph; the remainder of the host graph is called RestGraph. Keep track of the pre-embedding edges (i.e. edges that connected g_l^{host} to RestGraph).
4. Create g_r^{host} by making a copy of the right-hand side of the production rule. In this case g_l^{host} consists of a single node with label Exp (short for Expression).
5. Embed g_r^{host} into RestGraph: use the embedding information to translate pre-embedding edges into post-embedding edges. Here, the embedding phrase In_{Left} = {(1, 6')} specifies that an incoming Left edge to node 1 causes creation of an incoming Left edge to node 6'. Other edge labels are treated analogously, e.g. transferring outgoing Above edges from nodes 1 and 2 to node 6'.
6. Compute attribute values for g_r^{host}, using the attribute transfer function. Here, the meaning of the Exp node (represented by the attribute m, a string) is computed by concatenating the meanings of nodes 1 and 2, with appropriate insertion of parentheses.

Rank Use information about operator precedence to group symbols into subexpressions.

Incorporate Interpret subexpressions

The use of Build and Constrain phases allows recognition of expressions with considerable flexibility in the relative placement of symbols. Build is allowed to form edges generously, assuring that all relevant associations are included. Any excess edges are removed by the Constrain phase.

This graph-rewriting system interprets an expression in a bottom-up manner, with no backtracking. The recursive nature of subexpressions is handled as follows. Graph-rewriting rules in the Rank phase encode information about operator precedence and operator range as edge attributes. An ordering is assigned to the edges around each node, indicating which operator has highest priority locally. Next, Incorporate rules group the highest-priority symbols into subexpressions, replacing them by a single node. When Incorporate cannot find further subexpressions, the Rank rules are re-applied to update edge attribute values. Incorporate and Rank phases alternate until only a single node remains. The meaning attribute indicates the meaning of the entire input expression.

The recognition system consists of approximately 60 graph-rewriting rules. Execution is slow due to the crude graph-rewriting environment. Good results are obtained on the small set of test expressions that have been tried; these test expressions include highly irregular symbol positioning, as could occur in handwritten expressions. The current work does not handle errors from symbol recognition; however, spurious dots and poorly-aligned symbols are interpreted correctly.

In summary, graph-rewriting is a promising approach for mathematics recognition. Graph rewriting offers a flexible formalism with a strong theoretical foundation for manipulating two-dimensional patterns. It has been shown to be a useful technique for high-level recognition of circuit diagrams and musical scores (references in [5]). The organization into Build, Constrain, Rank, and Incorporate phases facilitates the construction and integration of new rewriting rules. Future research must address the error-prone nature of symbol recognition, either allowing the graph-rewriting system to process input consisting of a number of possible interpretations for each symbol, or providing for feedback from the graph-rewriting system to a symbol recognizer.

3.4. Procedurally-Coded Math Syntax

H. J. Lee *et al.* have developed a mathematics recognition system using procedural code to embody the syntax of mathematical notation [17] [24] [25]. After symbol recognition, the mathematical expression is represented by a list of symbols in random order. To recognize appropriate symbol groups (i.e. subexpressions), procedural code is used. After a group of symbols is recognized as a subexpression, it is represented by a bounding box, and further grouping takes place. To indicate the flavor of these rules, a few sample rules are repeated here. A length threshold of 20 pixels is used to classify a horizontal line as a long bar or a short bar. If a long bar has symbols both above and below it, it is treated as a division. If there are no symbols above it, it is treated as boolean negation. If a short bar has no symbols above or below it, it is treated as a minus sign. If it has symbols above or below it, then combination characters (such as = ≥ ≤ ⊇) are formed. Various sources of errors are discussed in [25], including: scanning (noise and broken characters); thinning (e.g. confusion between "x" and "λ"); and expression formation (mainly due to incorrect recognition of subscript versus in-line relations). Other aspects of this system are discussed in Sec. 2 (locating mathematical expressions in text) and Sec. 4.4 (splitting of connected symbols). In summary, a benefit of this procedural approach is fast execution; the authors also find it convenient to code recursively in this way. A disadvantage is that this approach provides a rather ad hoc representation of notational conventions, making the procedural code difficult to maintain or scale up.

The recognition system by Okamoto *et al.* [6] [14] [19] combines projection-profile analysis with an extensive array of procedurally-coded recognition rules. The system has been tested on more than 100 expressions, taken from journals or generated by TeX. Their system is able to correctly recognize most of the expression structures. The variety and complexity of their successfully-recognized expressions is impressive. This system began with the use of projection-profile cutting and has evolved through the addition of specialized code segments to correct particular recognition errors arising in earlier versions of the system. Thus it is difficult to summarize the design of the system; the reader is referred to [6] for a description of the extensive list of processing steps and thresholds used. This paper provides a compelling illustration of the complexity of the mathematics-recognition task, of the many symbol configurations that must be considered. The authors state that syntactic approaches, using parsing, are untenable because the great variety of possible expressions makes it impossible to provide an a priori syntax definition for all possible expressions. This, combined with the computational complexity of parsing, motivates them to instead use a large collection of procedurally-coded recognition rules. Their comment about a priori syntax definition requires some clarification: any recognition method, including procedurally-coded rules, implicitly or explicitly defines the syntax of recognizable expressions. Apparently Okamoto *et al.* find it easier and more practical to provide an implicit syntax definition, coded as procedural rules, rather than an explicit syntax definition, coded as grammar rules.

3.5. Data-driven and Knowledge-driven Modules

Faure and Wang present an interesting, systematic description of the mathematics-recognition problem [23]. The emphasis is on extracting the structure of handwritten expressions, with symbol recognition performed only in restricted circumstances. (The authors plan to add modules for symbol recognition and full syntax recognition later.) Evidence is cited that humans recognize the structure of an expression prior to performing complete symbol recognition. For example, when a person is reading a handwritten expression aloud, syntactic structures such as a division bar are recognized correctly, even before noticing that certain symbols in the numerator or denominator are illegible.

Input data is provided by people writing on a digitizing tablet. The system works with freely-written material – the users are not restricted to particular styles of mathematical notation, and the expressions users write deviate from a horizontal writing line. The sample expressions shown in the paper [23] exhibit significant, non-uniform skew, and many instances of hasty writing (such as a division bar that is significantly shorter than the numerator, and a Σ symbol with a gap between the two strokes used to form the symbol).

The recognition system is organized as a collection of independent modules, which communicate through a shared memory. The shared memory contains the input data, as well as the representation tree, which is updated as recognition progresses. (This architecture is reminiscent of a blackboard architecture, although here the shared memory does not trigger module application, but a fixed order of module application is used. Document-image-analysis systems that use a blackboard architecture are reviewed in [2].) Currently four modules are used:

- Acquisition
- Data-driven segmentation. This module builds an initial relational tree. It does not take writing-order into account, and thus applies to off-line data as well. No contextual information is available to the procedures of this module: the information available to an operator is restricted to the expression-subimage to which it is applied. Projections onto the X and Y axis are used; when these fail (e.g. due to a square root symbol, closely-written symbols, or excessive skew), a mask-removal operation is attempted. The first step is to detect a mask: first, special routines look for square roots and fraction bars; if these fail then any long or tall stroke is sought. Next the mask is removed, and the remaining symbols are analyzed using X or Y projections.
- Knowledge-driven segmentation. This module corrects the relational tree produced by the previous module. It uses domain-specific knowledge, looking for a set of specific patterns, and updating the relational tree by pattern-dependent corrections. This module has knowledge about a subset of the lexicon, the syntactic rules related to these symbols, the writing order (e.g. the long stroke is written before the dot in "i" or "j"), and symbol shape (e.g. the strokes of "E", "F", and "Σ" may not be connected). The patterns that this module looks for are structured as a hierarchy of frames. Once relational-tree nodes have been identified for correction, a complex procedure is needed to update the remainder of the tree in accordance with the corrected nodes.
- Line labeling. This module labels the spatial relationships (subscript, superscript, same line) between components of a writing line. Statistical labeling is used, as discussed in Sec. 5.1.2.

Modules for symbol recognition and full syntax analysis will be added as system development continues. This system provides an interesting subdivision of the mathematics-interpretation problem into subproblems, and it demonstrates the ability to cope with challenging handwritten input. The organization of this system as a set of independent modules imposes a welcome structure on the software. A structured approach eases the task of designing, coding, and debugging the complex and diverse rules needed during mathematics recognition.

4. Noise, Distortions, and Connected Symbols

Recognition of mathematical notation is greatly complicated by noise and distortions in the input. Various computational approaches can be used to deal with this problem; these include stochastic grammars, deterministic grammars with confidence-values for tokens, and postprocessing error-correction rules.

4.1. Stochastic Grammars

Chou uses a stochastic grammar to recognize noisy typeset equations [13]. The most likely parse is taken as the correct interpretation of the input. Each production rule in a stochastic grammar has an associated probability. The probability of a particular parse tree is computed by multiplying together the probabilities of each production used in the parse. Two-dimensional patterns are described by allowing each production rule to use either vertical or horizontal concatenation (Sec. 5.1.1). Square-roots are not recognized by the grammar described in [13]. To add them, the square-root sign would have to be treated as two symbols, so that the necessary spatial relationships can be described by horizontal and vertical concatenation.

A dynamic programming algorithm is presented for finding the most likely parse; complexity of this algorithm is $O(n^3)$. An iterative algorithm is used to learn grammar-parameters (such as production-rule probabilities) from training data: parsing all the training expressions gives an estimate of the expected number of times each production rule is used.

Conceptually, the stochastic grammar goes all the way down to the pixel level. As a practical matter, this is implemented in two parts. First, symbol candidates (with associated probabilities) are generated by exhaustive template matching (match every symbol-template to every image location; see Sec. 4.4). Next, the stochastic grammar is used to find the most likely parse of the symbol candidates; the calculation of a parse-probability includes contributions from the symbol-probabilities of all symbols used in the derivation.

Testing reported in [13] is created by copying bitmaps from a previewer for the eqn/troff typesetting system; noise is added by flipping each image bit with probability 0.01. The resulting images are quite challenging, due both to added noise and to the previewer's low resolution (100 dots per inch). Subsequent testing (mentioned during

Chou's demonstration at SSPR90, Murray Hill, New Jersey) involved scanning all equations in a mathematics textbook.

In summary, Chou mentions the following advantages of stochastic grammars for document recognition. The approach treats the interpretation process by parsing from the grammar-start-symbol to the pixel level, with no hard decisions made at any intermediate stage. The result is statistically optimal, given the noise model and grammar model. Noise, ambiguity, and touching characters are handled well. Grammar-parameters can be trained to achieve optimal performance on particular printers, fonts, layouts and scanners. To offset this, the following disadvantages are mentioned: computational expense, skew sensitivity (preprocessing must include good skew correction), and difficulty in accounting for kerning.

More recently, Kopec and Chou are performing document-image analysis using methods adapted from communication theory [26]. Here a finite-state model of the image-creation-process is included as part of the recognition system. This interesting, highly-promising approach is illustrated on the interpretation of the phone book's yellow pages columns. Experiments with the interpretation of other notations, such as music notation, have been undertaken as well.

4.2. Error Correction with a Coordinate Grammar

Dimitriadis *et al.* report on on-going work that extends Anderson's coordinate grammar to add error detection and correction [7]. Unfortunately, this is a preliminary report with few details. Dimitriadis' Ph.D. thesis (in Spanish; Dept. of Automatic Control and Systems, University of Valladolid, cited as "expected 1991") may provide more information. On-line data from a data tablet is processed by grouping strokes and applying elastic matching to recognize symbols; each symbol is represented by a series of candidates, with confidence measures. Parsing starts by considering the candidates with highest confidences; if the resulting global confidence figure is rather low, other candidates are tried, and a globally optimum decision is made. At the time of their report, this parsing scheme was in the planning stage; they do not discuss the danger of a combinatorial explosion of candidate-combinations.

4.3. Postprocessing Error-correction Rules

Heuristic rules for error-correction are discussed in [17]. Examples of these rules are as follows. Knowledge of special function names can correct character confusions ("5in" is corrected to "sin"; "c0sx" is corrected to "cos$x$"). The operands of a binary operator normally appear in the same font type and font size; this can be used to correct a character confusion such as "$/ < i < n$" to "$1 < i < n$". Knowledge of legal subscripts can eliminate further confusions, such as correcting 1_2 to l_2 and 5_2 to S_2.

4.4. Connected Symbols

Connected symbols are common both in typeset and handwritten mathematical expressions. Separation of such symbols is a difficult problem, which has been extensively studied in OCR and other domains as well. Here we review a few approaches to symbol-separation that have been used in mathematics-recognition systems.

Anderson assumes correctly-recognized symbols as input [3]. A grammar rule (number A56) shrinks each symbol's bounding box down to a center point. This enables subsequent coordinate-tests to yield the desired results. For example, in the expression $\frac{y}{x}$ the modified bounding box of the "y" lies wholly above the division bar.

To recognize symbols, Lee and Lee begin with thinning followed by curve detection [25]. Connected symbols are separated using a dynamic programming algorithm. First, the curve list is matched to each of the reference symbols. If the maximum similarity value is lower than a threshold, the curve list is passed to the dynamic programming algorithm. Here the curve list undergoes curve splitting and loop merging, and is put into a canonical order. Then the reference symbols are matched to the sequence of curves at the start of the curve list.

Chou's stochastic grammar does not require explicit separation of connected symbols [13]. Instead, the lexical access stage matches all templates in the font dictionary to every position in the image. (The font dictionary has separate entries for different font sizes.) A match is retained if the hamming distance (number of differing pixels) between the template and the image is below the three-sigma threshold. (Using a noise model in which image bits are independently flipped with probability P_e, the three-sigma threshold includes all matches where the number of disagreeing bits is within three standard deviations of the mean number of disagreeing bits. The probability of a greater number of disagreeing bits is less than 0.13%.) The result is a list of all characters that have significant probability of being present in the image. These hypothesized characters are then processed by the stochastic grammar (Sec. 4.1).

Systems that accept on-line input face a rather different segmentation problem. Where off-line systems must separate symbols that overlap in the image, on-line systems must gather together the set of strokes that form a single symbol. As described in Sec. 6, Chen and Yin [27] do this by analyzing the bounding boxes of successive strokes.

5. Identifying Spatial Relationships

It is difficult to recognize the spatial and logical relationships among symbols in a mathematical expression (Secs. 1.3 and 1.4). Here we review techniques for recognizing subscript, in-line, and superscript relations, and also briefly discuss the recognition of matrix notation.

5.1. Distinguishing Subscript, In-line, Superscript

The identification of subscripts, in-line, and superscript relations is a complex and important problem. For example, the system reported in [6] makes recognition errors that are typically errors in subscript and superscript recognition; this is a mature system that is capable of correctly recognizing many complex expression structures.

5.1.1. Global thresholds

Some authors address this problem by choosing global threshold parameters to distinguish the relations (e.g. [3] [4] [27]). As another example, Chou's stochastic grammar [13] encodes horizontal and vertical concatenation criteria into each production rule. Two examples are as follows. An <*Expression*> and a <*Factor*> can be horizontally concatenated only if both have the same pointsize, and their baselines differ by at most one pixel. A <*Symbol*> and a <*Subscript*> can be horizontally concatenated only if the subscript has a smaller pointsize and a lower baseline.

5.1.2. Statistical labeling

Extensive discussion of subscript and superscript identification is provided by Wang and Faure [28]. This paper highlights the complexity of the task, and offers an interesting approach to its solution. Given a sequence of symbol-bounding-boxes that are part of the same writing line, the task is to label the relationships between pairs of symbols as exponent (E), in-line (L), or subscript (S). (Note that these are spatial relationships, not logical relationships. The symbol pairs in the expression a^2b^3 are labeled E S E; later processing must determine that the b is not logically a subscript of the 2.) Both adjacent and non-adjacent pairs of symbols must be labeled. For example, x^{a_i} and $x^a i$ have the same pairwise labelings (E followed by S); to distinguish these expressions, the x-to-i relationship must be labeled as either E or L. Processing takes place on bounding-box information alone; symbol identity is recognized later. Thus, constraints based on symbol identity (Sec. 5.1.3), cannot be applied here. Similarly, character-baseline information is unavailable. (Ambiguity arises in cases such as y_c versus bc; these have similarly-configured bounding boxes, but different spatial relations among the symbols.)

Training data (35 expressions produced by 7 writers) is used to construct a recognizer. Two features are measured from each pair of bounding boxes: the height-ratio and vertical offset. A modified Ho–Kashyap algorithm is used to place decision boundaries in this feature space. The space is divided into six regions (exponent, either exponent or in-line, in-line but on the exponent side, in-line but on the subscript side, either in-line or subscript, subscript). This is done separately for each of the three following zones: taller box followed by a shorter box; two boxes of approximately equal height; shorter box followed by a taller box.

During classification, the feature space is used to calculate an initial estimate of the probabilities of the E, L and S labels for the given symbol pair. The final labeling is found by searching for the most-probable labeling that also satisfied the contextual constraints. These constraints eliminate impossible situations such as the following: in the symbol-triple ab^c, if ab is labeled L and b^c is labeled E, then $a\ {}^c$ must be labeled E. Testing results (using expressions produced by writers who did not participate in the production of training data) are promising. In a comparison with human labeling (based on a presentation of bounding boxes), the system had slightly higher error rates than the humans. The authors propose addressing this by introducing directionality into the recognition process.

This labeling process constitutes one recognition module in the complete recognition system [23] discussed in Sec. 3.5.

5.1.3. Constraints based on symbol identity

Another approach to sub- and superscript recognition is mentioned in [17]. Recognition is based on spatial coordinates as well as local context. The system makes use of a priori information regarding legal configurations for superscripts and subscripts: for example, a^x and x_n are legal, whereas B_* and $X^\&$ are not. Similar constraints are used in [6].

5.2. Recognizing Matrices

Matrix-recognition can be restricted to explicit matrices, or it can include linked vectors, in which a line connecting two matrix elements denotes an indeterminate number of intervening elements.

Anderson presents a coordinate grammar for matrix notation, which recognizes both explicit and linked matrices [3]. The matrix recognition is heavily dependent on the availability of three parameters:

mhmax	maximum separation between two horizontally adjacent matrix elements
mvmax	maximum separation between two vertically adjacent matrix elements
vmax	maximum vertical separation between adjacent characters belonging to the same matrix element

The coordinate grammar recognizes rows, columns, or diagonals one at a time. Therefore it never obtains an overall view of matrix structure. For example, suppose that the input is a “matrix” with a variable number of entries in each row. The grammar does not notice this; it parses the input as an EXPLICITARRAY and reports the number of matrix columns as equal to the number of entries in the last row of the matrix. It would be interesting to explore whether the coordinate-grammar formalism can produce a fuller analysis of matrix structure.

In [6], analysis of explicit matrices begins by finding a pair of delimiters of the same size and type. Next a horizontal projection profile is computed for the delimited region; if this profile exhibits periodicity (indicating several rows of similar height), then a vertical projection profile is used to separate the columns. Finally, each matrix entry is analyzed as an expression in its own right.

Another approach to matrix recognition is mentioned in [17]. First a pair of enclosure symbols are found. Symbols within this area are grouped based on proximity; no details of the proximity metric are given.

6. On-line Recognition of Handwritten Expressions

A data tablet offers the possibility of using timing information to assist with the recognition of a mathematical expression.

Chen and Yin describe a system for recognizing handwritten mathematical notation [27]. The emphasis of this work is on symbol segmentation (i.e. collecting the strokes that make up a single symbol) and symbol recognition. Symbol segmentation proceeds by surrounding each stroke with a frame (a bounding box), and testing consecutive frames for overlap in X and Y projections. Special tests are performed to unify two-part symbols such as i, j, $=$, $\leq$, $\geq$. Also, special tests are performed for the square-root symbol, which obscures gaps in the X and Y projections of symbols inside it. Symbol recognition is performed using a nearest-neighbor algorithm, with features coded from stroke types (line, clockwise curve, counterclockwise curve, circle). Multi-character classes are created for confusion-prone symbols, such as "1" versus "/", "0" versus "O", or "C" versus "(". After nearest-neighbor classification, specific contextual constraints are tested to disambiguate symbols in these multi-character classes. For example, if two operands are found on either side of a "1 or /" character, then this character is classified as a "/", otherwise as a "1". If the system rejects a character, or fails to disambiguate it, the user is asked for correction. Recognition results are presented for seven writers entering eight test expressions; the overall symbol-recognition rate is 94%.

On-line data is analyzed in [7]; few details are given, but they do mention the use of elastic matching (dynamic time warping).

The system by Faure and Wang [23] [28] is geared toward recognizing the structure of handwritten mathematical notation. On-line information is used as needed during the knowledge-driven segmentation module. Considerable variability in the handwriting can be tolerated. Sec. 3.5 provides further discussion of this work.

Recognition of handwritten expressions is treated in [18], as discussed in Sec. 3.1.1. Timing information (from the data tablet) is used in extracting strokes for symbol recognition, but is not used during syntactic analysis.

7. Summary and Conclusions

Mathematics recognition is a practically-important problem. Recognition is difficult because mathematical notation is only semi-standardized, conveys meaning through subtle use of spatial relationships, involves a large alphabet of symbols and a large range of font sizes, and contains little redundancy in its representation of information. Comparison among existing mathematics-recognition systems is complicated by the myriad ways in which the mathematics-recognition problem can be defined. Input may be typeset or handwritten, on-line or off-line, input expressions may be isolated or mixed with text, recognizable expression layouts may be restricted to a greater or lesser degree. In addition, some recognizers assume pixels as input, whereas others assume that symbol-recognition has already taken place. This great variation in problem-definition makes it difficult to compare the strengths and weaknesses of existing mathematics recognition systems. Research is needed to clarify the definition of mathematical notation and its dialects. Such a definition could provide a basis for more meaningful comparison among mathematics-recognition systems.

A variety of system architectures are used for mathematics recognition. Syntactic analysis involves parsing an expression using a two-dimensional grammar, such as a coordinate grammar or stochastic grammar, after symbol recognition has been performed. A grammar provides a clear, explicit representation of mathematical-notation syntax, although parsing can be computationally expensive. Projection-profile cutting determines the structure of an expression by recursively dividing the image into subregions (using minima in the horizontal and vertical projection profiles); this can be done before symbol recognition is performed. This simple and efficient technique is suitable for solving a restricted part of the mathematics-recognition problem. Graph rewriting represents symbols and symbol-relationships as a graph, rewriting parts of the graph as recognition proceeds. This offers a flexible means of applying extensive knowledge of notational conventions; however, current graph-rewriting environments execute quite slowly. Procedurally-coded rules provide step-by-step instructions for recognizing an expression. Unless a systematic organization of rules is provided, such a system can become ill-understood and unmanageable due to the unexpected interactions among the unstructured, often ad hoc recognition rules. The data-driven and knowledge-driven modules of Faure and Wang illustrate one approach for systematically organizing procedurally-coded recognition rules.

Whatever recognition approach is used, the structure of recognizable expressions must be characterized. Such a syntax for mathematics may be defined implicitly or explicitly. A major objective is to define the syntax cleanly, to provide a unifying framework for handling the myriad details and exceptions that arise during mathematics recognition.

References

[1] Y. Nakayama, Mathematical formula editor for CAI, *Proc. ACM SIGCHI Conf. on Human Factors in Computing Systems*, Austin, Texas, April 1989, 387–392.

[2] D. Blostein, General diagram-recognition methodologies, *Proc. Int. Workshop on Graphics Recognition*, University Park, Pennsylvania, Aug. 1995, 200–212.

[3] R. Anderson, Two-dimensional mathematical notation, in *Syntactic Pattern Recognition, Applications*, ed. K. S. Fu (Springer -Verlag,1977) 147–177.

[4] S. Chang, A method for the structural analysis of two-dimensional mathematical expressions, *Information Sciences* **2**, 3 (1970) 253–272.

[5] A. Grbavec and D. Blostein, Mathematics recognition using graph rewriting, *Third Int.Conf. on Document Analysis and Recognition*, Montreal, Canada, Aug. 1995, 417–421.

[6] H. Twaakyondo and M. Okamoto, Structure analysis and recognition of mathematical expressions, *Proc. Third Int. Conf. on Document Analysis and Recognition*, Montreal, Canada, Aug. 1995, 430–437.

[7] Y. Dimitriadis, J. 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, *Proc. First Int. Conf. on Document Analysis and Recognition*, Saint Malo, France, Sept. 1991, 242–250.

[8] T. Chaundy, P. Barrett, and C. Batey, *The Printing of Mathematics* (Oxford University Press, 1957).

[9] K. Wick, *Rules for Typesetting Mathematics*, translated by V. Boublik and M. Hejlova (The Hague, Mouton, 1965).

[10] N. Higham, *Handbook of Writing for the Mathematical Sciences* (SIAM, Philadelphia, 1993).

[11] D. Knuth, Mathematical typography, *Bulletin of the American Mathematical Soc.* **1**, 2 (1979).

[12] W. Martin, Computer input/output of mathematical expressions, *Proc. 2nd Symp. on Symbolic and Algebraic Manipulations*, New York, 1971, 78–87.

[13] P. Chou, Recognition of equations using a two-dimensional stochastic context-free grammar, *Proc. SPIE Visual Communications and Image Processing IV*, Philadelphia PA, Nov. 1989, 852–863.

[14] M. Okamoto and B. Miao, Recognition of mathematical expressions by using the layout structure of symbols, in *Proc. First Int. Conf. on Document Analysis and Recognition*, Saint Malo, France, Sept. 1991, 242–250.

[15] J. Ha, R. Haralick, and I. Phillips, Understanding mathematical expressions from document images, *Proc. Third Int. Conf. on Document Analysis and Recognition*, Montreal, Canada, Aug. 1995, 956–959.

[16] D. Blostein and H. Baird, A critical survey of music image analysis, in *Structured Document Image Analysis*, eds. H. Baird, H. Bunke, and K. Yamamoto, (Springer-Verlag, 1992) 405–434.

[17] H. Lee and J. Wang, Design of a mathematical expression recognition system, *Proc. Third Int. Conf. on Document Analysis and Recognition*, Montreal, Canada, Aug. 1995, 1084–1087.

[18] A. Belaid and J. Haton, A syntactic approach for handwritten mathematical formula recognition, *IEEE Trans. Pattern Analysis and Machine Intell.*, **6**, 1 (1984) 105–111.

[19] M. Okamoto and A. Miyazawa, An experimental implementation of document recognition system for papers containing mathematical expressions, in *Structured Document Image Analysis*, eds. H. Baird, H. Bunke, and K. Yamamoto (Springer-Verlag 1992) 36–53. (Earlier version in *Proc. IAPR Workshop on Statistical and Structural Pattern Recognition*, Murray Hill, New Jersey, June 1990, 335–350.)

[20] A. Murase, T. Satou, and M. Nakagawa, Prototyping of METAH, a recognition system for on-line handwritten mathematical expressions, *Information Processing Society of Japan SIG Notes*, **93**, 35 (1993) 25–32 (in Japanese).

[21] M. Okamoto and H. Higashi, Mathematical expression recognition by the layout of symbols, *Trans. IEIEC* **J78-D-II**, 3, (1995) 474–482 (in Japanese).

[22] M. Okamoto and H. Twaakyondo, Mathematical expression recognition by projection profile characteristics, *Trans. IEIEC* **J78-D-II**, 2 (1995) 366–370 (in Japanese).

[23] C. Faure and Z. Wang, Automatic perception of the structure of handwritten mathematical expressions, in *Computer Processing of Handwriting* (World Scientific, 1990) 337–361.

[24] H. Lee and M. Lee, Understanding mathematical expressions in a printed document, *Proc. Second Int. Conf. Document Analysis and Recognition*, Tsukuba, Japan, Oct. 1993, 502–505.

[25] H. Lee and M. Lee, Understanding mathematical expressions using procedure-oriented transformation, *Pattern Recognition* **27**, 3 (1994) 447–457.

[26] G. Kopec and P. Chou, Document image decoding using markov source models, *IEEE Trans. Pattern Analysis and Machine Intelligence* **16**, 6 (1994) 602–617.

[27] L. Chen and P. Yin, A system for on-line recognition of handwritten mathematical expressions, *Computer Processing of Chinese and Oriental Languages* **6**, 1 (1992) 19–39.

[28] Z. Wang and C. Faure, Structural analysis of handwritten mathematical expressions, *Proc. Ninth Int. Conf. on Pattern Recognition*, Rome, Italy, Nov. 1988, 32–34.