Math-Literate Computers

Dorothea Blostein

School of Computing
Queen's University, Kingston, Ontario, Canada K7L 3N6
blostein@cs.queensu.ca

Abstract. Math notation is a familiar, everyday tool widely used in society. Computers need math literacy – the ability to read and write math notation – in order to assist people with accessing mathematical documents and carrying out mathematical investigations. In this paper, we discuss issues in making computers math-literate. Software for generating math notation is widely used. Software for recognition of math notation is not as widely used: to avoid the intrusiveness and unpredictability of recognition errors, people often prefer to enter and edit math expressions using a computer-oriented representation, such as LaTeX or a structure-based editor. However, computer recognition of math notation is essential in large-scale recognition of mathematical documents; as well, it offers the ability to create people-centric user interfaces focused on math notation rather than computer-centric user interfaces focused on computer-oriented representations. Issues that arise in computer math literacy include the diversity of math notation, the challenges in designing effective user interfaces, and the difficulty of defining and assessing performance.

1 Introduction

Math notation is a widely-used two-dimensional language for expressing and reasoning about mathematics. This notation developed over centuries, with many variants and dialects. Math notation is fluid, with users creating new forms of math notation as the need arises. Historically, math notation was written and read by people. The recent invention of the computer has lead to widespread use of electronic representations of mathematical expressions. Electronic representations support services such as typesetting, search, and automated reasoning. We need math-literate computers in order to best combine the convenience of paper-based math notation with the power of computer-based math representations.

Currently, computer generation of math notation is common, but recognition is less commonly used: the task of translating math notation into a computer-processable form is often done manually. With continuing advances in math recognition software, the need for manual entry of computer-oriented math formats will decrease.

Input and output of math notation is carried out in various contexts. Here is an informal description of a few scenarios, with and without math-literate computers.

- A person creates a document containing math expressions:
  - With a math-literate computer, this can be done via handwritten entry. The computer software must be able to cope with the variability of handwriting.

J. Carette et al. (Eds.): Calculemus/MKM 2009, LNAI 5625, pp. 2–13, 2009.

Handwritten expressions can be scanned, or the user can write directly on a data tablet; the tablet has the advantage of making stroke-timing information available for use in the recognition process. The user receives feedback about the recognition result, and is thus available to correct recognition errors.

- With manual entry, a person directly enters the structure of a math expression by typing an ASCII form of the expression (as in LaTeX), or by issuing a sequence of commands to a structure-based math editor.

- Paper documents are converted to electronic form:
  - With a math-literate computer, scanned documents are interpreted by document-recognition software. Layout analysis is used to separate the document into text regions, math expressions, and figures. Text regions are interpreted by optical character recognition (OCR), math regions are interpreted by math recognition, and figures are interpreted by graphics recognition software [32][34]. When a small number of documents are converted, a person can perform checking and correction of the results. When a large document collection is involved, people perform only very limited checking and correction. In that case, subsequent software must make allowance for the possibility of recognition errors in the electronic documents.
  - With manual entry, a person directly enters the structure of the math expressions in the documents. This is can be done for small-scale applications, but is infeasible for large document collections. If manual entry is infeasible and automatic interpretation of math expressions is unavailable, then the math expressions can be left uninterpreted: this leaves math expressions as image regions that can be displayed, but cannot be queried by word-based or symbol-based searches.

- A math document is converted from a notation-oriented electronic form to an information-oriented electronic form (for example, from LaTeX to a symbolic algebra format such as Maple):
  - With a math-literate computer, this conversion is done automatically. External information is required, for example to distinguish function names from variable names (see Section 3.1).
  - With manual entry, a person directly enters the symbolic algebra form of the math expressions.

- A math document is converted from an electronic form to paper (or to display on a computer monitor):
  - This is typically done by computer software. Present-day computers are math-literate in the *writing* direction.

Computer math-literacy is a practically-important subject that offers fascinating research opportunities. In this paper, I present my opinions about challenges and issues that arise in this area. Please note that I am not up-to-date in all the latest publications, but am supplying a retrospective view of the developments in computer processing of math notation over the past twenty years.

2 Diagrams and Notational Conventions

A *diagram* expresses information using a two-dimensional layout of symbols. *Notational conventions* are the constraints that define the mapping between information and two-dimensional layout. A math expression is an example of a diagram, expressed using the notational conventions of math notation. A page of sheet music is another example of a diagram, expressed using the notational conventions of music notation. As illustrated in Figure 1, knowledge of notational conventions is needed both for diagram recognition (*reading* the notation) and for diagram generation (*writing* the notation).

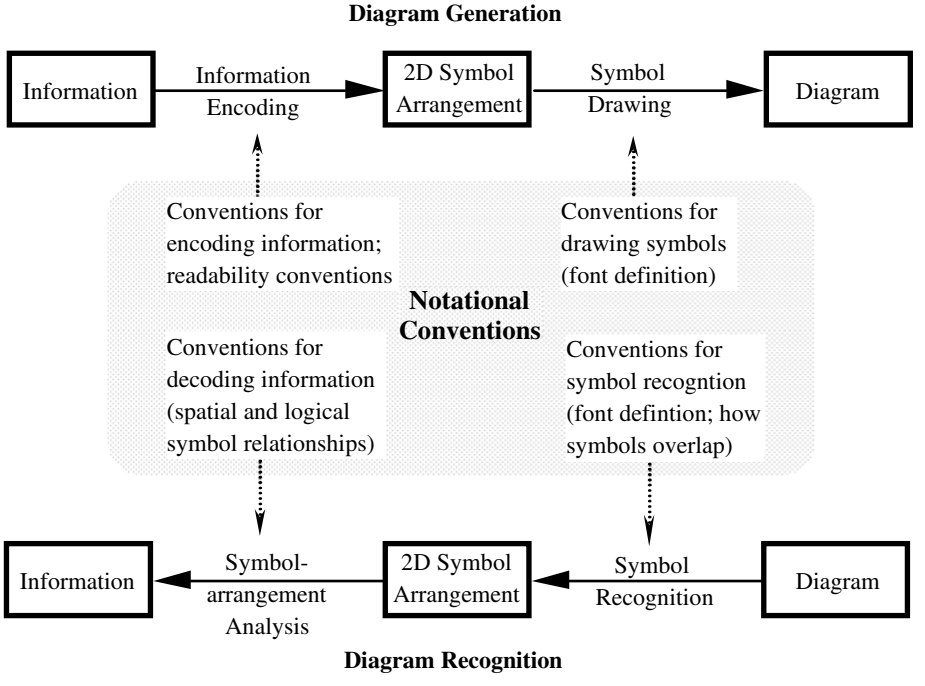

Fig. 1. Notational conventions are used for diagram generation and recognition [4]. When a diagram is generated, notational conventions guide the creation of an aesthetically pleasing diagram that encodes the given information. When a diagram is recognized, notational conventions guide the recognition of symbols and their logical relationships, and dictate how to infer information from this symbol arrangement. This figure illustrates sequential processing steps, but symbol recognition and symbol-arrangement analysis can be concurrent, allowing the use of contextual information to improve symbol-recognition results.

The mapping between information and diagram is not one to one. Many diagrams represent the same information using different layouts. A diagram with *good* layout is easy to read, and is aesthetically appealing. Ideally, diagram generation software automatically chooses a good layout, while diagram recognition software recognizes the information conveyed by the diagram, no matter what the layout of the diagram.

There is significant overlap in the notational conventions used for generation and recognition. However, notational conventions are treated differently due to differences in the two diagram processing tasks. Aesthetic considerations are central in diagram generation: users want nice-looking, readable diagrams. In contrast, diagram recognition systems pay less attention to aesthetics: they are trying to recover the information conveyed by the diagram, and are not trying to judge how nice the diagram layout is. Noise and uncertainty are central in diagram recognition: there is uncertainty about symbol segmentation, symbol recognition, interpretation of the relative placement of symbols, and so on. These problems do not arise in generation. It is interesting to generalize this line of thought, to consider the relationship between the fields of computer vision and computer graphics [21].

The acquisition, representation and exploitation of notational conventions are central to computer math literacy. Notational conventions are equally important in processing other types of two-dimensional notation. The research community as a whole is gradually developing general computational approaches to diagram recognition, which are useful in interpreting diagrams of various types. Much has been published about recognition of various types of diagrams, such as math expressions, engineering drawings, maps, music notation, and bar charts [20][31][32][34]. Publications about diagram generation include graph drawing [36] and visual languages [33]. Many aspects of diagrams are discussed in the Diagrams conferences [35].

2.1 Hard and Soft Notational Conventions

We call a notational convention *hard* if it is used consistently, and *soft* if its use is optional. Hard conventions specify how information is encoded in the two-dimensional notation, and soft conventions specify how to make the diagram readable. Here are some examples, informally expressed. In graph drawing, a hard convention is that "an edge drawn between two nodes represents a relation between the nodes". A soft convention is "choose a graph layout that minimizes the number of edge crossings". In math notation, a hard convention is "division can be encoded by drawing a horizontal line with the dividend expression placed above the line and the divisor expression placed below the line". A soft convention is "when breaking an expression into multiple lines, put the break before a major operator". Petre provides related observations about hard and soft conventions, using the term *secondary notation* for the layout aspects of a diagram [28].

Soft conventions can be applied to a greater or lesser degree, and can be ignored in exceptional circumstances. For example, a soft convention in music notation is "leave more space after long notes than after short notes" [2]. However, this convention is sometimes ignored when music notation is printed very densely, in order to end the page at a pause that gives the performer time to turn the page.

The same information can be represented by a large set of diagrams. These diagrams differ in readability and aesthetic appeal. As an example, the layout of a circuit diagram can be changed; this affects the appearance of the diagram without changing the information being conveyed.

Most diagram recognizers ignore soft conventions, relying wholly on the hard conventions. Poor diagram layout is not noticed, and good diagram layout is not exploited. If diagram recognizers could be expanded to make greater use of soft conventions, this

could increase the robustness of recognition: the recognizer can make use of layout and spacing cues. To achieve this, the recognition software needs the ability to reason with constraints that hold "most of the time".

Since diagram generators already use soft conventions, the question arises whether diagram recognizers can be improved by exploiting the knowledge and experience embodied in diagram generators [6]. Specifically, LaTeX software is in widespread use and encodes sophisticated knowledge about the formatting of math notation. Can this generation-oriented knowledge be exploited to improve math-notation recognizers? Possible approaches include reusing generator code to proofread and correct recognizer output, building a model of the generation process into the recognition software, and using a generator to construct cases for a recognizer that uses case-based reasoning [6].

2.2 Sources of Information about the Definition of Math Notation

The design of a math-literate computer system should begin with a definition of math notation: a definition of the syntax and semantics of the two-dimensional language used to express mathematics. Unfortunately, math notation, like most diagram notations, is not formally defined. Rather, it is informally established through common usage. Math notation is only semi-standardized, allowing many variations and drawing styles. The same is true of natural languages, such as English. Building a software model of the notational conventions used in math notation is a complex and time-consuming task.

Sources of information about math notation include written descriptions of math notation, sample documents, coded descriptions built into software for recognizing math notation, coded descriptions built into software for generating math notation, and human experts [5]. Most written descriptions are oriented toward generation of the notation rather than recognition of the notation. Descriptions of math notation for people who want to solve typesetting problems include [12][18][40], and descriptions of math notation oriented toward computational typesetting include [22]. Almost 40 years ago, Martin suggested that the first step in automating the processing mathematical notation is to make a study of the notation, and he went on to present a brief list of the notational conventions found in use in technical publications [25].

Many factors influence how mathematical symbols should be grouped during the recognition of math notation. Some grouping factors are defined for math notation in general; these include operator range and operator precedence. Other grouping factors arise within a particular mathematical expression; these include symbol identity (which often determines whether the symbol is an operator or an operand), relative symbol placement, and relative symbol size and case. Further discussion and related references are provided in [5].

2.3 Electronic Representations of Math Notation

A variety of electronic representations of math notation are in active use, including pixel-oriented representations such as JPEG, symbol-oriented representations such as PDF and PostScript, syntax-oriented representations such as LaTeX, and symbolic-algebra representations such as Maple. Research is needed to better understand these representations: how to define the equivalence of documents and the distance between

documents, how to mathematically characterize the mapping between document representations, how to characterize the external information needed to carry out these mappings, and how to characterize the differences between the forward and inverse mappings that occur during document analysis and document production [9].

3 Recognition of Math Notation

In developing software for recognition of math notation, much can be learned from existing research into recognition of other types of diagrams [4]. Over time, our collective experience in recognizing various types of diagrams is giving rise to a general technology for diagram recognition. An appealing analogy is provided by compiler technology: the first compilers were difficult to write, but over time the community developed general techniques for parsing and code generation, which greatly simplify the task of constructing compilers for new source and target languages. Diagram-recognition methods are difficult to generalize, due to the great diversity among diagram notations, and due to the complexity of handling noise and uncertainty. However, algorithms can be shared for common subproblems such as symbol recognition [13][37]. Document recognition contests provide standardized task definitions, including training and testing data, as well as evaluation metrics. Contests have been held for such as dashed-line detection, raster to vector conversion, arc segmentation, symbol recognition, page segmentation, handwriting segmentation, and Arabic handwriting recognition [15].

3.1 External Information Needed for Math Recognition

Math expressions are not self-contained. External information is needed in order to fully understand them. Some of this information, such as the definition of symbols, comes from other parts of the source document. Other information is external: for example, knowledge of symbolic algebra can be applied to find errors in a printed expression or its interpretation. Many dialects of math notation are in use, varying by discipline; the choice of dialect is implicit, and must be inferred by some external means, typically involving familiarity with the math notation used in related publications. Other diagram notations have an analogous need for external information: for example, engineering drawings rely on reader's knowledge of disassembly and kinematics [38], and music notation relies on the reader's knowledge of music theory and performance practice.

The acquisition, representation, and use of external knowledge is a broad and interesting topic, one that is important to the future development of math-literate computers. Without external information, a simple expression such as *a(b)* can be interpreted up to a notation-oriented format such as LaTeX, but further interpretation up to the symbolic algebra level is impossible without the knowledge of whether *a* is a function name or a variable name. When subexpressions are used repeatedly in a document, noticing and exploiting this repetition helps increase the robustness of a recognizer.

3.2 Challenges in Recognizing Math Notation

Many challenges arise in the recognition of math notation [5]. Small symbols, such as dots and commas, are commonly used and are critical to the meaning of the notation; these small symbols are difficult to distinguish from noise. Symbol recognition 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. A few common symbols in math notation have several possible roles: 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 meaning of such symbols must be determined through the contextual information provided by surrounding symbols.

A major challenge in math recognition is identification of the logical meaning of spatial relationships. Implicit mathematical operators are defined entirely by spatial relationships, with no explicit operator symbol; these include superscripts, subscripts, implied multiplication, and matrix structure. Examples of difficult cases are given in many publications from [25] onward. In handwritten notation, the ambiguity of spatial relationships is greatly increased, due to free placement and alignment of symbols. Many researchers (e.g. [39]), have studied the problem of distinguishing horizontal adjacency from superscripts and subscripts: the continuous range of possible symbol placements $2x\ \ 2x\ \ 2^x\ \ 2^x\ \ 2^x$ makes this difficult.

Offsetting these challenges, two characteristics of math notation make it relatively easier to process than many other types of diagram notations. Firstly, most symbols in a math expression are surrounded by white space, which greatly simplifies symbol segmentation. (An exception is handwritten mathematical notation, which may contain overlapping symbols; these can be difficult to segment, particularly if off-line data is used.) Secondly, math notation has a relatively regular and recursive syntax, which makes it well-suited for processing using grammar-based and compiler-like techniques.

3.3 Finding the Math Expressions in a Document

Computer math literacy depends on having automated ways of finding math expressions, or having convenient user interfaces for the user to indicate the location of math expressions of interest. This is not a problem in on-line recognition systems, where a person writes input on a data tablet: in this case, text and math expressions are generally not mixed. The situation is different for paper documents, where math expressions are typically mixed with text, either as offset expressions, or embedded directly into a line of text. The first step in math recognition is to identify where expressions are located on the page. This topic, and document layout analysis in general, has been subject of much research e.g. [34][41].

3.4 Computational Methods for Recognizing Math Notation

Many approaches to math recognition have been explored, including syntactic methods, graph transformation, projection-profile cutting, and procedurally-coded rules [5][10][11]. Noise and uncertainty can be handled by producing lists of alternatives

that are passed from one recognition stage to the next, or by executing recognition stages concurrently, using contextual feedback to compensate for noisy input or to reject erroneous input [4]. The many proposals for how to organize a math recognition system are fascinating; each has its own merits and its own advocates. It is difficult to judge which organization is best for a given application.

I have had long-standing interest in development of a general software technology for diagram recognition. My first informal comments, at a workshop in 1990 [1], described a goal of a diagram-recognition technology, analogous to the existing technology available for compilers. Compiler technology makes it (relatively) easy to create compilers with new source and target languages, and similarly a diagram-recognition technology would make it (relatively) easy to create recognizers for new types of diagrams. However, diagram recognition faces added problems due to noise, due to the huge range of types of diagram notations, and due to the natural-language aspects of diagram notations.

Many compiler techniques can be adapted to pattern recognition [7]. Techniques that have been imported include the use of grammars (array grammars, tree grammars, set grammars, graph grammars), and parsing technologies (for example, CYK and Early algorithms for context free grammars, and linear-time LR and LL parsing algorithms for more restricted languages). We illustrate the use of two additional compiler techniques in a math-notation recognition system: use of trees and tree transformation, and a multi-pass control structure, with a clear separation between layout, lexical, syntactic, and semantic analysis [7][42]. The main steps are to (1) find linear structures in the input, and use these as a basis for finding secondary linear structures; (2) organize the linear structures into a tree; (3) divide processing into passes for layout, lexical analysis, syntax, and semantics; (4) use a simple, fixed control structure, such as a sequence of passes; and (5) use tree transformation technology, which provides highly efficient techniques for manipulating trees, and notations for expressing manipulations in a concise, readable form.

3.5 Users Reaction to Math Recognition: The User Interface

Math-recognition software is far less widely used than math generation software. Some of this is due to availability, and to the maturity of the technology. But there are additional, social factors that work against recognition systems. Here are some speculative comments of these factors, with the aim of stimulating discussion and new directions for development of recognition systems.

It seems that the recognition errors made by a computer are quite intrusive, because they are different from the interpretation problems that a person has when reading messy or noisy text. If a person is struggling to read your document, her or she will ask you about semantics: what do you mean here? In contrast, the computer asks about marks on the page: is this a *w*, is that item over there one symbol or two symbols? The computer does not state these questions directly, but a user who is proofreading recognition output – to correct symbol recognition and symbol segmentation errors – is implicitly answering questions like this. Users would find recognition software more inviting if it could move in the direction of allowing users to think more about the meaning of the notation, and less about the marks on the paper. Computer-Human Interaction is a heavily-researched subject that has many ideas to offer [29][30].

Predictability and blame-assignment are two reasons why recognition software isn't as popular as it could be. Consider the case of a user who types a big LaTeX expression, and gets an error because of unbalanced parentheses. Is this user upset at the LaTeX software? No, instead the user blames himself or herself: that was my stupid mistake, I forgot a parenthesis, next time I will do better. Consider instead a user of a pen-based math entry system who gets an error because of a misrecognized symbol. This user is likely to blame the recognition software: what stupid software, even my four-year-old son is better at symbol recognition than that, I hope the software will do better next time -- but it is hard for me to figure out how to help it do better.

A basic question is: do people really want automated recognition of math expressions (assuming that the recognition rate is suitably high)? The answer is certainly *yes* for the case of document recognition, but for manual entry of math expressions, the answer probably depends on the person. I will begin by discussing entry of text, and then move on to entry of math expressions. For entering text, I personally would rather type on a keyboard than handwrite on a data tablet – this is because I can type much faster than I can write by hand. On the other hand, a slow typist may prefer writing on a data tablet (with application of OCR software) over typing on a keyboard. In the case of math expressions, my personal preference is to have an entry method that is focused on the 2D math notation. However, it is possible that some people prefer to use LaTeX because they are so practiced that they can type expressions faster than they can handwrite them.

Before designing a user interface for math recognition, it is worth reviewing the attractive properties of paper: ergonomics, contrast, resolution, weight, viewing angle, durability, cost, life expectancy, and editorial quality [19]. Paper also has limitations: erasing is difficult, it is not possible to "make extra room" to expand parts of a diagram, it is hard to find information in a large stack of paper, and so on. A goal for future computer interfaces is to retain the advantages of paper while also providing the editing and search capabilities lacking in paper. Designers reject the use of computers in the early, conceptual, creative phases of designing, preferring to use paper and pencil, which permits ambiguity, imprecision, and incremental formalization of ideas [16]. Computer based tools force designers into premature commitment, demand inappropriate precision, and are often tedious to use when compared with pencil and paper. For further discussion, see [8].

4 Generation of Math Notation

Less is published on the generation of diagram notations than on the recognition of diagram notations. Diagram generation technology is mature and often proprietary. Prominent among publications on diagram generation is Knuth's work on math notation [22]. We extend ideas from Knuth's text spacing algorithm [23] to music spacing [17]. Also, much has been published on algorithms for graph drawing [36]; many diagram notations are based on graphs.

Interesting issues arise in providing a user interface to generated notation. The user should be allowed to modify the generated notation, without the need to repeat all the modifications when the notation is regenerated [3].

5 Performance Evaluation Issues

Although performance evaluation is an active research area in document image analysis, there are few formally defined performance metrics for diagram generation or recognition. Informally, a generator is successful if it generates information-bearing images that the user finds aesthetically pleasing. There are no ground-truth models of ideal generator output. A generator is debugged on a test suite of diagrams, and in response to user feedback. Evaluating the performance of diagram recognition systems involves defining requirements, characterizing the system's range of inputs and outputs, interpreting published performance evaluation results, reproducing performance evaluation experiments, choosing training and test data, and selecting performance metrics [24].

The user interface is critical to the success of a diagram recognition system. It is difficult to define precise goals for a user interface, and even more difficult to quantify performance of a user interface [8]. Separating user-interface performance from recognition performance is difficult: the time that a user spends correcting recognition errors depends both on the number of recognition errors and on the qualities of the user-interface facilities for finding and correcting errors. The performance of different types of visual feedback in a math recognition system is studied in [43]. One possible performance measure is to compare the time and accuracy of automated and unautomated entry of diagrams, as discussed in [8]. Neilsen defines usability attributes: learnability, efficiency, memorability, errors and satisfaction [26]. Measurable usability parameters include subjective user preference measures, which assess how much the users like the system, and objective performance measures, which measure the speed and accuracy with which users perform tasks on the system [27].

6 Conclusion

Computers should serve people, assisting them in their work. Making computers math literate is an important step in this direction, allowing people to work using the familiar math notation, avoiding the need for them to learn new notations for the convenience of the computers. Computer math literacy provides a smooth transition between paper documents and electronic documents, combining the best properties of paper with the advanced search and evaluation capabilities offered by electronic documents. This paper has summarized some of the issues involved in creating math literate computers. Much progress has been made, and many interesting problems remain to be addressed.

Acknowledgments

Financial support from the Natural Sciences and Engineering Research Council of Canada and from the Xerox Foundation is gratefully acknowledged.

References

[1] Blostein, D.: Structural Analysis of Music Notation. In: Proc. IAPR Workshop on Syntactic and Structural Pattern Recognition, Murray Hill, NJ, p. 481 (1990)

[2] Blostein, D., Haken, L.: Justification of Printed Music. Communications of the ACM 34(3), 88–99 (1991)

[3] Blostein, D., Haken, L.: The Lime Music Editor: A Diagram Editor Involving Complex Translations. Software – Practice and Experience 24(3), 289–306 (1994)

[4] Blostein, D.: General Diagram-Recognition Methodologies. In: Kasturi, R., Tombre, K. (eds.) Graphics Recognition 1995. LNCS, vol. 1072, pp. 106–122. Springer, Heidelberg (1996)

[5] Blostein, D., Grbavec, A.: Recognition of Mathematical Notation. In: Bunke, H., Wang, P. (eds.) Handbook of Character Recognition and Document Image Analysis, pp. 557–582. World Scientific, Singapore (1997)

[6] Blostein, D., Haken, L.: Using Diagram Generation Software to Improve Diagram Recognition: A Case Study of Music Notation. IEEE Trans. Pattern Analysis and Machine Intelligence 21(11), 1121–1136 (1999)

[7] Blostein, D., Cordy, J., Zanibbi, R.: Applying Compiler Techniques to Diagram Recognition. In: Proc. 16th Intl. Conf. on Pattern Recognition, Quebec City, Canada, August 2002, vol. III, pp. 123–126 (2002)

[8] Blostein, D., Lank, E., Rose, A., Zanibbi, R.: User Interfaces for Online Diagram Recognition. In: Blostein, D., Kwon, Y.-B. (eds.) GREC 2001. LNCS, vol. 2390, pp. 92–103. Springer, Heidelberg (2002)

[9] Blostein, D., Zanibbi, R., Nagy, G., Harrap, H.: Document Representations. In: Proc. Fifth IAPR Int'l Workshop on Graphics Recognition (GREC 2003), Barcelona, Spain, July 2003, pp. 3–12 (2003)

[10] Blostein, D.: Graph Transformation in Document Image Analysis: Approaches and Challenges. In: Brun, L., Vento, M. (eds.) GbRPR 2005. LNCS, vol. 3434, pp. 23–34. Springer, Heidelberg (2005)

[11] Chan, K., Yeung, D.: Mathematics Expression Recognition: a Survey. Int'l Journal on Document Analysis and Recognition 3(1), 3–15 (2000)

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

[13] Chhabra, A.: Graphic Symbol Recognition: An Overview. In: Chhabra, A.K., Tombre, K. (eds.) GREC 1997. LNCS, vol. 1389, pp. 68–79. Springer, Heidelberg (1998)

[14] Cushman, W., Ojha, P., Daniels, C.: Usable OCR: What are the Minimum Performance Requirements? In: Proc. ACM SIGCHI 1990 Conference on Human Factors in Computing Systems, Seattle, Washington, April 1990, pp. 145–151 (1990)

[15] Document Recognition Contests, `http://www.icdar2007.org/competition.html`

[16] Gross, M., Do, E.: Ambiguous Intentions: a Paper-like Interface for Creative Design. In: Proc. Ninth Annual Symposium on User Interface Software and Technology (UIST 1996), Seattle, Washington, November 1996, pp. 183–192 (1996)

[17] Haken, L., Blostein, D.: A New Algorithm for Horizontal Spacing of Printed Music. In: International Computer Music Conference, Banff, September 1995, pp. 118–119 (1995)

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

[19] Hsu, R., Mitchell, W.: After 400 Years, Print is Still Superior. CACM 40(10), 27–28 (1997)

[20] Int'l Journal on Document Analysis and Recognition. Springer Verlag (since 1998)
[21] (Int'l Conf.) Computer Vision/ Computer Graphics Collaboration Techniques and Applications, INRIA Rocquencourt, France, May 4-6 (2009)
[22] Knuth, D.: Mathematical Typography. Bulletin of the American Mathematical Society 1(2), 337–372 (1979)
[23] Knuth, D., Plass, M.: Breaking Paragraphs into Lines. Software – Practice and Experience 11, 1119–1184 (1981)
[24] Lapointe, A., Blostein, D.: Issues in Performance Evaluation: A Case Study of Math Recognition. In: Int'l Conf. Document Analysis and Recognition (ICDAR 2009), Barcelona (July 2009) (to appear)
[25] Martin, W.: Computer Input/Output of Mathematical Expressions. In: Proc. 2nd Symposium on Symbolic and Algebraic Manipulations, pp. 78–87. ACM, New York (1971)
[26] Nielsen, J.: Usability Engineering. Academic Press, San Diego (1993)
[27] Nielsen, J., Levy, J.: Measuring Usability: Preference vs. Performance. Communications of the ACM 37(4), 66–75 (1994)
[28] Petre, M.: Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming. Communications of the ACM 38(6), 33–44 (1995)
[29] Proc. ACM Conference on Human Factors in Computing Systems (CHI). ACM Press, New York (annual since 1982)
[30] Proc. ACM Symposium on User Interface Software and Technology (UIST). ACM Press, New York (annual since 1988)
[31] Proc. IAPR Int'l Workshop on Document Analysis Systems (biennial since 1994)
[32] Proc. IAPR Int'l Workshop on Graphics Recognition (biennial since 1995)
[33] Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (annual since 1988) (earlier name: Proc. IEEE Symposium on Visual Languages)
[34] Proc. Int'l Conf. on Document Analysis and Recognition (biennial since 1991) (sponsored by IAPR and IEEE)
[35] Proc. Int'l Conf. on the Theory and Application of Diagrams (biennial since 2000)
[36] Proc. Int'l Symposium on Graph Drawing (held annually since 1993)
[37] Tombre, K., Tabbone, S., Dosch, P.: Musings on Symbol Recognition. In: Liu, W., Lladós, J. (eds.) GREC 2005. LNCS, vol. 3926, pp. 23–34. Springer, Heidelberg (2006)
[38] Vaxivière, P., Tombre, K.: Knowledge Organization and Interpretation Process in Engineering Drawing Interpretation. In: Proc. IAPR Workshop on Document Analysis Systems, Kaiserslautern, Germany, October 1994, pp. 313–321 (1994)
[39] Wang, Z., Faure, C.: Structural Analysis of Handwritten Mathematical Expressions. In: Proc. Ninth Int'l Conf. on Pattern Recognition, Rome, Italy, November 1988, pp. 32–34 (1988)
[40] Wick, K.: Rules for Typesetting Mathematics, translated by V. Boublik and M. Hejlova, The Hague, Mouton (1965)
[41] Workshop on Document Layout Interpretation and its Applications (DLIA) (1999 and 2001)
[42] Zanibbi, R., Blostein, D., Cordy, J.: Recognizing Handwritten Mathematical Expressions Using Tree Transformation. IEEE Trans. Pattern Analysis and Machine Intelligence 24(11), 1455–1467 (2002)
[43] Zanibbi, R., Novins, K., Arvo, J., Zanibbi, K.: Aiding Manipulation of Handwritten Mathematical Expressions through Style-Preserving Morphs. In: Proc. Graphics Interface 2001, Ottawa, Ontario, June 2001, pp. 127–134 (2001)