

Baseline Structure Analysis of Handwritten Mathematics Notation

Richard Zanibbi Dorothea Blostein James R. Cordy

*Department of Computing & Information Science,
Queen's University, Kingston, Ontario, Canada*
{zanibbi, blostein, cordy}@cs.queensu.ca

Abstract

The structure of mathematics notation is particularly difficult to recognize in handwritten notation because irregular symbol placements are common. We present an efficient and robust method of parsing handwritten and typeset mathematics notation without backtracking. The system is designed to be easily adaptable to various dialects of mathematics notation. The following strategies are used: (1) separate the analysis of layout, syntax, and semantics, (2) recursively apply search functions and image partitioning to recognize dominant and nested baselines, and (3) use tree transformations to express computations in a compact, efficiently executable form.

1. Introduction

Mathematics notation conveys information using a two-dimensional arrangement of symbols. Recognition software must analyze this spatial structure, in order to convert from a document image to a structural representation such as LaTeX or a semantic representation such as an operator tree or Maple. However, it is difficult to define robust, general and efficient methods for analyzing the spatial structure of mathematics notation. This problem is particularly difficult in handwritten mathematics notation (obtained from scanned document images, or from data tablet input), where irregular placement of symbols is common.

1.1 Summary of Existing Work

Research into automatic recognition of mathematical expressions has been ongoing for over thirty years [3,5]. Methods developed for recognizing the two-dimensional layout of symbols in a math expression can be roughly categorized into syntactic (grammar-based) and algorithmic approaches. Syntactic methods have been used extensively, including coordinate grammars [1,25], attributed string grammars [2,12,13,14,34], stochastic grammars [10,26], structure specification schemes [6], and graph transformation [17,22,23,28]. Algorithmic approaches have included recursively locating vertically stacked groups of symbols using procedural [24] and blackboard-style methods [15,30], recursive baseline location [20], and minimization of penalty functions on symbol relations [16]. Another algorithmic approach, projection profile cutting with subsequent adjustments, has been used to

obtain expression structure directly from pixel maps [18,27,29]. Ambiguities of symbol layout and identity have been handled by constructing multiple interpretations and then eliminating unsyntactic [31] or unlikely [26] interpretations.

We obtain two insights from this literature. First, almost all authors use trees to describe the spatial structure of mathematics notation. In many cases the tree is an explicit data structure; in other cases an implicit parse tree is created. Second, mathematical expressions have a preferred *direction of interpretation*, as used by human readers; this directionality can be exploited by a recognition system [1,14,25]. The direction of interpretation is usually left-to-right; however, Arabic notation is read right-to-left [13]. Our recognition system makes extensive use of trees, tree transformations, and directionality of the notation.

1.2 System Overview

The DRACULAE system (Diagram Recognition Application for Computer Understanding of Large Algebraic Expressions) interprets the symbol layout of large mathematical expressions [32]. The input to DRACULAE is a list of symbols with their spatial locations, from which DRACULAE produces LaTeX and operator tree outputs. This system quickly recognizes symbol layout in a general way even when the semantics of a construct are unknown, and is capable of successfully analyzing large handwritten expressions with poor layout. The amount of search needed to analyze layout is reduced by exploiting the left-to-right reading order of mathematics notation.

We obtained software for on-line entry, segmentation and recognition of handwritten symbols from the Freehand Formula Entry System (FFES) developed by Steve Smithies, Kevin Novins and Jim Arvo [28]. We used FFES to create test expressions using a data tablet and mouse. FFES runs a nearest-neighbour symbol recognizer while a user enters an expression and allows the user to correct any symbol recognition errors that occur. After entering symbols a user may invoke DRACULAE from within FFES and obtain bitmap or style-preserving morph feedback [33] on DRACULAE's interpretation.

2. Separate Analysis of Layout, Syntax, Semantics

DRACULAE is divided into three passes: Layout, Syntax, and Semantics, as illustrated in Figure 1 and Table 1. Using separate passes has the advantage of separating the knowledge

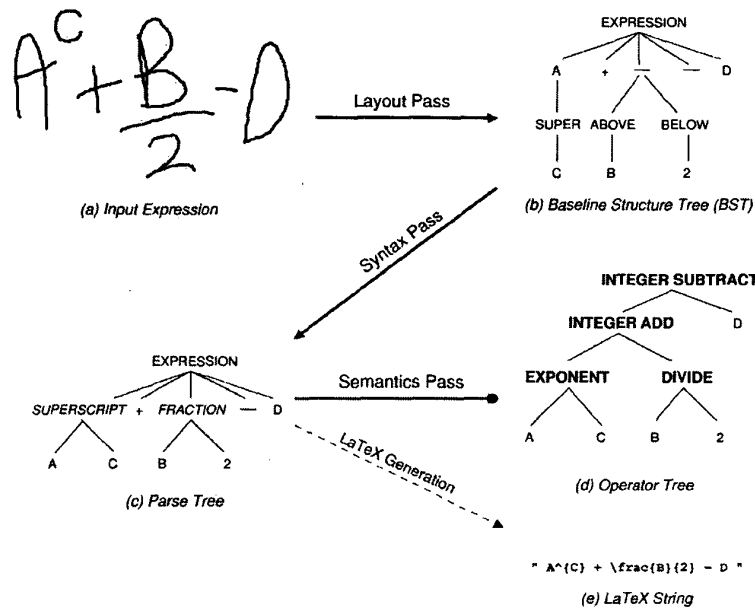


Figure 1. The Three Passes Used to Process a Math Expression.

In (a), the irregular symbol placement causes a misleading alignment of $A+B$. Here, the Semantics pass uses integers as the domain for operator tree (d). The Semantics pass can be reconfigured for other domains, such as real numbers or matrices.

bases and analysis routines that are used in each pass. This makes the software better structured, easier to maintain, and easier to adapt to different dialects of math notation.

Mathematics notation has many variants (or *dialects*), all of which use similar spatial structure but vary in semantics. Our recognition system uses the following methods to handle dialects:

- Each recognition pass (Layout, Syntax, and Semantics) has its own data structures to describe the aspects of math notation used in that pass. This separation means, for example, that the semantics of a dialect can be changed without affecting the syntactic description, as when f' can mean differentiation or function inverse or simply an annotated symbol.
- The first recognition pass, the Layout pass, extracts spatial structure from a list of symbols and represents this in a dialect-independent fashion (a *baseline structure tree*; see Section 3.1). The general structural description from the Layout pass is altered in the Syntax and Semantics passes. All three passes may be adapted to conform with different dialects of mathematics notation.

There is no formal definition of mathematics notation that can be used as a standard of correctness. Written descriptions regarding the generation of mathematics notation are available [7,19,21]. However, these are not in a form that can be used as a

specification for a mathematics recognition system. While some formal definitions of mathematics notation have been proposed (e.g. [34]), the notation itself continues to evolve through use in society.

3. The Layout Pass: Construction of a Baseline Structure Tree

The Layout pass identifies the baseline structure of the mathematical expression, producing a *baseline structure tree*. In this tree, every symbol is assigned to one baseline. The baselines are grouped into a hierarchy of dominant and nested baselines. The baseline structure tree explicitly captures important aspects of symbol layout without committing to any particular syntactic or semantic interpretation.

We specify how baselines may be nested relative to individual symbols in a BST using a symbol layout model. The symbol layout model presented in this paper contains four *symbol classes*. These are *Limit* (operators that may have limits, such as sum and integral), *Sqrt*, *Nonscripted* (symbols that are never followed by superscripts or subscripts: unary and binary operators, open brackets, horizontal line), and *Plain* (all other symbols, including alphanumeric symbols and closed brackets). In the symbol layout model we also specify the centroid location for symbols, which is used to test whether a symbol lies within a region. The centroid for each symbol is computed based on its

Pass	Processing Performed	Image Coordinates	Knowledge base used; can be adjusted to the dialect
Layout <u>Input</u> symbols <u>Output</u> BST	Identify the baseline structure of the math expression and create a baseline structure tree (BST). The symbol layout model and search functions are used to construct the BST.	Extensive use of image coordinates. Analyze relative placement of symbols. Amount of white space does not affect interpretation.	Symbol layout model (symbol classes define regions around symbols, e.g. regions around Σ). Knowledge in search functions <i>Start()</i> and <i>Hor()</i> .
Syntax <u>Input</u> BST <u>Output</u> parse tree	Identify grammatical structures such as Σ and the baselines of its limits and body. Identify multi-symbol tokens such as numbers (digits and decimal points), function names (e.g. <i>cos</i>), = and \leq . Create a parse tree.	Image coordinates not used.	Grammar describing structural composition. Specified as a BNF grammar and transformation rules in TXL (see Section 5).
Semantics <u>Input</u> parse tree <u>Output</u> operator tree	Analyze operator precedence and associativity to create an operator tree.	Image coordinates not used.	Grammar describing operator precedence and associativity. Specified as a BNF grammar and transformation rules in TXL (see Section 5).

Table 1. The Three Passes used in DRACULAE to Analyze a Set of Mathematical Symbols.

bounding box coordinates, and reflects whether the symbol is an ascender, descender, or neither.

3.1 Baseline Structure Trees

A baseline structure tree contains two types of nodes: *symbol* nodes and *region* nodes. These nodes are arranged in levels: any path through the tree encounters symbol nodes and region nodes in alternation. The root of the tree, EXPRESSION, is a region node representing the entire image. Every region node in the BST represents an image region which contains a baseline, possibly with nested baselines. The subtree that is rooted at a region node represents the baseline structure of all the symbols in this region. Region nodes represent all mathematically-important spatial relationships other than horizontal adjacency. Horizontal adjacency has special status because it defines baselines. Symbols that are on the same baseline are represented in the tree as ordered siblings. This is illustrated by the tree in Figure 1(b). This tree contains four region nodes (EXPRESSION, SUPER, ABOVE, BELOW) and eight symbol nodes (A + - - D C B 2). The dominant baseline of the whole expression is (A + - - D). The "2" is the sole symbol in a baseline located BELOW the first "-". The "C" is the sole symbol of the baseline located in a superscripted region (SUPER) relative to the "A".

The extent of an image region depends on the *walls* defined by other symbols in the expression. For example, in Figure 1(b), the SUPER region of "A" is walled by the "+": the maximum x coordinate of the SUPER region equals the minimum x coordinate of the "+".

3.2 Identifying Baseline Structure of an Expression

By exploiting reading order, the baseline structure tree can be constructed efficiently, without backtracking, even when symbol layout is irregular. Here is a summary of the processing steps. Extensive research went into defining the search functions *Start()* and *Hor()* used in steps 4 and 5. Space does not permit a full explanation of these functions.

1. Sort the input symbols by leftmost bounding box coordinate.
2. Look up the *symbol class* and *centroid* for each symbol.
3. Initialize: The Baseline Structure Tree is a single EXPRESSION node. R is the image region that contains the entire expression. L is the sorted list of symbols from step 1.
4. Compute $S_1 = Start(L)$ to find the symbol S_1 which starts the dominant baseline in region R [32]. *Start()* checks for cases in which symbol S_1 is not the leftmost symbol in list L. For example, the limits of a Σ can begin to the left of the Σ .
5. Find the rest of the symbols in the baseline that begins with symbol S_1 . *Hor()* finds the next symbol in a baseline; it handles irregular layouts such as those in Figure 1 [32]. Compute $S_2 = Hor(S_1, L)$, $S_3 = Hor(S_2, L)$, and so on until *Hor* returns null.
6. The symbols S_1, S_2, \dots, S_n are the dominant baseline in region R. Add these symbols to the baseline structure tree: insert n symbol nodes as offspring of the region node representing R.

7. The symbols in the dominant baseline (S_1, S_2, \dots, S_n) partition region R into subregions. All symbols have ABOVE and BELOW regions. (For Limit symbols, these regions are labeled UPPER and LOWER; they may extend to the left and right of the symbol.) Sqrt symbols have a CONTAINS region. Symbols in classes Plain and Sqrt have SUPER and SUBSC regions. Assign each remaining symbol in L (any symbol other than S_1, S_2, \dots, S_n) to one of these subregions.
8. Add region nodes to the baseline structure tree to represent the non-empty sub-regions found in step 7. Apply steps 4 to 8 to the symbol lists in each of these regions.

This algorithm is illustrated using the expression in Figure 1(a). First, the dominant baseline is found (steps 4 and 5). *Start()* finds A and *Hor()* finds $+ - - D$. Note the robustness of these search functions: starting at the $+$, *Hor()* finds the $-$, despite the alignment of the $+$ and the B .

Once the symbols in the dominant baseline have been found, step 6 extends the baseline structure tree to be EXPRESSION with five branches, leading to $A, +, -, -, D$. Step 7 defines regions around these symbols. The nonempty subregions are added to the tree in step 8: A has offspring SUPER, and “-” has two offspring, ABOVE and BELOW. Steps 4 to 8 are recursively applied to these three non-empty subregions to complete the construction of the baseline structure tree. Figure 1(b) shows the final tree.

In summary, the Layout pass recursively applies search functions and image partitioning to recognize dominant and nested baselines. The search function *Start()* is used to locate the leftmost symbol of the dominant baseline, and the search function *Hor()* is used to locate successive symbols in a baseline. This use of search functions was inspired by the Positional Grammar work of Costagliola et al. [11]. The directionality present in mathematics notation made it possible for us to adapt these ideas for use in our Layout pass.

4. The Syntax and Semantics Passes

The syntax pass converts a Baseline Structure Tree into a parse tree. This involves tree transformations which reorganize the tree to identify tokens comprised of multiple symbols (e.g. numbers), and grammatical structures comprised of multiple baselines (e.g. fractions, or the limit and body baselines associated with a Σ).

The semantics pass converts a parse tree into an operator tree. This involves tree transformations which reorganize the tree according to operator precedence and associativity. The processing done in this stage is analogous to processing done by the semantic analysis phase of a compiler. Neither the syntax or semantic passes depend on image coordinates, though the syntax pass collects coordinate information for user interface purposes.

5. Tree Transformation vs. Graph Transformation

As was illustrated in Figure 1, trees are the central data structure used in our recognizer. To construct and modify these trees, a programming-language construct called *tree transformation* is used throughout the implementation. A tree transformation rule searches a host tree for a subtree that matches the rule's *pattern* (left hand side); this subtree is then locally transformed according to the rule's *replacement* (right hand side). The TXL language specifies tree transformations in a compact, abstract manner [8,9]. TXL specifications are directly and efficiently executable. The amount of code needed to describe a tree transformation is orders of magnitude smaller in TXL than in a language such as C. Sample TXL code is shown in Figure 2.

Tree transformation is well-suited to the math recognition domain, because math expressions have a recursive structure which is naturally described by a tree. Further research is required to determine the extent to which our ideas can be applied in recognizing diagrams from domains other than mathematics. We believe that the separation of layout, syntax and semantics can be used as a structuring principle in designing diagram recognizers from many domains.

In earlier work, we used graph transformation for mathematics recognition [17]. Graph transformation is an attractive and versatile style of computation, but easily runs into efficiency problems. Some work has been done to increase the efficiency of graph grammar parsing in the context of recognizing mathematical expressions [22,23], but we have found a tree transformation-based approach to be adequate. Careful application of search functions and dominance analysis allows an initial tree to be constructed by DRACULAE's Layout pass. Separating Layout, Syntax and Semantics has also made DRACULAE easier to extend than our former graph transformation system [17], where transformation rules dealt with layout, syntax and semantics all at once.


```

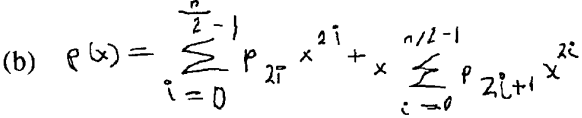
rule convertAdditionsToOperatorTrees
  replace [expression]
    LeftSubexpression[expression] + RightSubexpression[term]
  by
    "Integer Add" { LeftSubexpression } { RightSubexpression }
end rule

```

Figure 2. A Tree Transformation Rule Written in TXL.

This rule from the Semantics Pass replaces the parse subtree for each subexpression parsed as a binary + operation with an operator subtree for the corresponding integer addition.

(a)  $\bar{a} \vee \overline{b \vee c}$

(b)  $p(x) = \sum_{i=0}^{\frac{n}{2}-1} p_{2i} x^{2i} + x \sum_{i=0}^{\frac{n}{2}-1} p_{2i+1} x^{2i}$

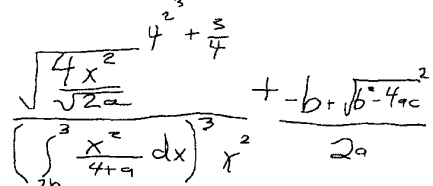
(c)  $\frac{\left(\int_{2b}^3 \frac{x^2}{4+a} dx\right)^3 x^2}{2a} + \frac{-b + \sqrt{b^2 - 4ac}}{2a}$

Figure 3. Examples of Expressions used in Testing.

On the left are hand-drawn expressions created in FFES [26]. The images on the right are generated from the corresponding LaTeX strings created by DRACULAE. The LaTeX strings are derived from the parse trees. (The processing in the Semantics Pass is not needed for LaTeX.) Running on a 500 MHz Pentium III under Linux, each of these expressions is processed in less than one second.

6. Implementation and Testing

DRACULAE is implemented in TXL [8,9]. The current version of the system recognizes single-line expressions which do not include matrices. We have tested DRACULAE on hundreds of hand-drawn expressions, a few of which are illustrated in Figure 3. This testing is made possible by connecting DRACULAE to the user interface and character recognition software from the Freehand Formula Entry System (FFES) [28].

DRACULAE's multi-pass design makes it easy to adapt the system to recognize new constructs. For instance, the following additions allow DRACULAE to process Boolean negation, notated by an overbar. No change is made to the Layout pass. In the syntax pass, add a tree transformation rule to find any horizontal line that has symbols below it and no symbols above; change the label of this line to "OVERBAR". In the code that generates LaTeX from the parse tree, replace this label with "\overline{ }". These types of alterations are easy to perform on the compact, abstract TXL specifications of the tree transformations.

DRACULAE recognizes the layout of a wide variety of handwritten mathematics expressions efficiently, using the reading order of mathematics notation to reduce the amount of search needed. A baseline structure tree and LaTeX string are produced for all input expressions, including syntactically invalid expressions with errors such as unbalanced parentheses. This is because the Layout pass does not enforce syntax or semantics, and the Syntax pass only rewrites tree structures, leaving any unsyntactic symbol layouts in the parse tree. The parse tree is translated to LaTeX regardless of whether the tree represents a valid mathematical expression.

7. Conclusion

Separating recognition of mathematics notation into Layout, Syntax and Semantic analysis passes is a powerful and useful technique. The separation of structure from semantics is common practice in compiler designs, but has been rarely used in graphics recognition systems. The multi-pass design used in DRACULAE allows robust handling of unexpected input, and makes it easier to adapt the system to recognize new constructs.

In DRACULAE we have exploited the left-to-right reading direction of mathematics notation in an algorithm and search functions that analyze the symbol layout of poorly formatted handwritten expressions in an efficient and general way. We describe the recognized symbol layout of mathematical expressions using baseline structure trees: these are a concise, readable, and dialect-independent representation of the hierarchy of baselines present in a mathematical expression.

In future work we will refine our layout analysis algorithm and search functions, add the use of whitespace information, define a number of mathematical dialects and provide translation to computer algebra system formats (e.g. Maple, Mathematica). We also hope to explore the use of direction to restrict searching while recognizing other diagrammatic notations.

Acknowledgements.

We thank Steve Smithies, Kevin Novins, and Jim Arvo for use of the Freehand Formula Entry System. Genarro Costagliola, Edward Lank, Nick Willan and George Weigt contributed through helpful discussion and assistance with the implementation. This research is supported by the Natural Sciences and Engineering Research Council of Canada.

References.

- [1] R. Anderson, "Two Dimensional Mathematical Notation," in *Syntactic Pattern Recognition, Applications*, Editor K. S. Fu, Springer 1977, pp. 147-177.
- [2] A. Belaid and J. Haton, "A Syntactic Approach for Handwritten Mathematical Formula Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, 6(1), January 1984, pp. 105-111.
- [3] D. Blostein, A. Grbavec, "Recognition of Mathematical Notation," in *Handbook of Character Recognition and Document Image Analysis*, Eds. H. Bunke and P. Wang, World Scientific, 1997, pp. 557-582.
- [4] D. Blostein and A. Schürr, "Computing with Graphs and Graph Transformation," *Software - Practice and Experience*, 29(3), 1999, pp. 197-217.
- [5] K. Chan and D. Yeung, "Mathematics Expression Recognition: a Survey," *Intl. Journal on Document Analysis and Recognition*, 3(1), August 2000, pp. 3-15.
- [6] S. Chang, "A Method for the Structural Analysis of Two-Dimensional Mathematical Expressions," *Information Sciences*, 2(3), 1970, pp. 253-272.
- [7] T. Chaundy, P. Barrett, C. Batey, *The Printing of Mathematics*, Oxford University Press, 1957.
- [8] J. Cordy, I. Carmichael, R. Halliday, *The TXL Programming Language - Version 10*, TXL Software Research Inc., Kingston, Canada, January 2000 (65 pp). www.txl.ca/txldocs.html
- [9] J. Cordy, C. Halpern, E. Promislow, "TXL: A Rapid Prototyping System for Programming Language Dialects," *Computer Languages* 16(1), January 1991, pp. 97-107.
- [10] P. Chou, "Recognition of Equations Using a Two-Dimensional Stochastic Context-Free Grammar," *Proc. SPIE Conf. on Visual Communications and Image Processing IV*, Philadelphia PA, Nov. 1989, pp. 852-863.
- [11] G. Costagliola, A. De Lucia, S. Orefice, G. Tortora, "A Framework of Syntactic Models for the Implementation of Visual Languages," *Proc. 1997 IEEE International Symposium on Visual Languages (VL'97)*, Capri, Italy, Sept. 1997, pp. 58-65.
- [12] Y. Dimitriadis, J. Coronado, C. de la Maza, "A New Interactive Mathematical Editor, Using On-line Handwritten Symbol Recognition and Error Detection-Correction with an Attribute Grammar," *Proc. ICDAR'91 Saint Malo, France*, Sept. 1991, pp. 242-250.
- [13] T. El-Sheikh, "Recognition of Handwritten Arabic Mathematical Formulas," *United Kingdom Information Technology Conference*, March 1990.
- [14] R. Fateman, T. Tokuyasu, B. Berman, and N. Mitchell, "Optical Character Recognition and Parsing of Typeset Mathematics," *J. Visual Communication and Image Representation*, 7(1), March 1996, pp. 2-15.
- [15] C. Faure and Z. Wang, "Automatic Perception of the Structure of Handwritten Mathematical Expressions," in *Computer Processing of Handwriting*, Eds. R. Plamondon and C. Leedham, World Scientific, 1990, pp. 337-361.
- [16] R. Fukuda, S. I. F. Tamari, "A Technique of Mathematical Expression Structure Analysis for the Handwriting Input System," *Proc. ICDAR'99, Bangalore, India*, September 1999, pp. 131-134.
- [17] A. Grbavec and D. Blostein, "Mathematics Recognition Using Graph Rewriting," *Proc. ICDAR'95, Montreal, Canada*, August 1995, pp. 417-421.
- [18] J. Ha, R. Haralick, I. Phillips, "Understanding Mathematical Expressions from Document Images," *Proc. ICDAR'95, Montreal, Canada*, August 1995, pp. 956-959.
- [19] N. Higham, *Handbook of Writing for the Mathematical Sciences*, Siam, Philadelphia, 1993.
- [20] K. Inoue, R. Miyazaki, M. Suzuki, "Optical Recognition of Printed Mathematical Documents," *Proc. Third Asian Technology Conference in Mathematics*, Tsukuba, Japan, 1998, pp. 280-289.
- [21] D. Knuth, "Mathematical Typography," *Bulletin of the American Mathematical Society*, 1(2), March 1979, pp. 337-372.
- [22] A. Kosmala, G. Rigoll, S. Lavirotte, L. Pottier, "On-Line Handwritten Formula Recognition using Hidden Markov Models and Context Dependent Graph Grammars," *Proc. ICDAR'99, Bangalore India*, Sept. 1999, pp. 107-110.
- [23] S. Lavirotte and L. Pottier, "Mathematical Formula Recognition using Graph Grammar," *Document Recognition V, SPIE Proceedings Series, Volume 3305*, 1998, pp. 44-52.
- [24] H. Lee and J. Wang, "Design of a Mathematical Expression Understanding System," *Pattern Recognition Letters*, 18, 1997, pp. 289-298.
- [25] W. Martin, "Computer Input/Output of Mathematical Expressions," *Proc. 2nd Symposium on Symbolic and Algebraic Manipulations*, ACM, New York, 1971, pp. 78-87.
- [26] E. Miller, P. Viola, "Ambiguity and Constraint in Mathematical Expression Recognition," *Proc. AAAI'98, 15th National Conference on Artificial Intelligence*, Madison, Wisconsin, July 1998, pp. 784-791.
- [27] M. Okamoto and A. Miyazawa, "An Experimental Implementation of Document Recognition System for Papers containing Mathematical Expressions," in *Structured Document Image Analysis*, Eds. Baird, Bunke, Yamamoto, Springer 1992, pp. 36-53.
- [28] A. Smithies, K. Novins, J. Arvo, "A Handwriting-Based Equation Editor," *Proc. Graphics Interface '99*, sponsor: Canadian Human-Computer Communications Society, Kingston, Ontario, June. 1999, pp. 84-91.
- [29] H. Twaakyondo and M. Okamoto, "Structure Analysis and Recognition of Mathematical Expressions," *Proc. ICDAR'95, Montreal, Canada*, August 1995, pp. 430-437.
- [30] Z. Wang and C. Faure, "Structural Analysis of Handwritten Mathematical Expressions," *Proc. Ninth Intl. Conf. on Pattern Recognition*, pp. 32-34, Rome, Italy, November 1988.
- [31] H. Winkler, J. Fahrner, M. Lang, "A soft-decision approach for structural analysis of handwritten mathematical expressions," *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing -- ICASSP'95*, pp. 2459-2462.
- [32] R. Zanibbi, "Recognition of Mathematics Notation via Computer Using Baseline Structure," Technical Report ISBN-0836-0227-2000-439, Dept. Computing and Information Science, Queen's University, Kingston, Ontario, August 2000.
- [33] R. Zanibbi, K. Novins, J. Arvo, K. Zanibbi, "Aiding Manipulation of Handwritten Mathematical Expressions through Style-Preserving Morphs," *Proc. Graphics Interface 2001*, Ottawa, Canada, June 2001, pp. 127-134.
- [34] Y. Zhao, T. Sakurai, H. Sugiura, T. Torii, "Formalization and Parsing of Mathematical Expressions for Mathematical Computation," *J. Japan Society for Symbolic and Algebraic Computation*, 6(3), 1998, pp. 2-29.