MathBrush: A System for Doing Math on Pen-Based Devices

George Labahn, Edward Lank, Scott MacLean, Mirette Marzouk, David Tausky

David R. Cheriton School of Computer Science
University of Waterloo, Waterloo, Ontario, Canada
e–mail: {glabahn,lank,smaclean,msmarzouk,datausky}@cs.uwaterloo.ca

Abstract

Many on-line (interactive) mathematics recognition systems allow the creation of typeset equations, normally in LaTeX, but they do not support mathematical problem solving. In this paper, we present MathBrush, a system that allows users to draw math input using a pen-input device on a tablet computer, recognizes the math expression, and then supports mathematical transformation and problem solving using back-end Computer Algebra Systems (CAS). We describe the architecture of the MathBrush system, which includes modules that support symbol recognition, semantic analysis, the transfer of recognized expressions to back-end CAS, and interface techniques for interacting with CAS output. We also identify unique challenges associated with recognition for math problem solving, such as the need for deeper semantic analysis than is required by LaTeX, and the need to deal with ambiguities in user input. Our experiences serve to inform researchers seeking to design interactive mathematics recognition systems geared toward mathematical problem solving.

1. Introduction

Mathematics recognition can be separated into off-line systems, which extract equations from scanned document images, and on-line, or interactive, systems, which accept hand drawn input and parse math expressions. In the off-line domain, many recognition systems have as their goal the archiving of physical manuscripts [18]. The output of these systems is typically a typographic representation of the math expressions, thus allowing the original paper document to be stored digitally and reconstructed with high quality on a computer screen or by a printer.

Here we focus on on-line mathematics recognition, where the goal is to allow a user to enter a math expression into a computer. Users enter mathematics into a computer for a number of reasons. At one extreme, one wishes to generate typeset expressions for a manuscript. At the other, one wishes to solve or manipulate a mathematical expression, perhaps using a Computer Algebra System (CAS) such as Maple or Mathematica. CAS are often viewed as sophisticated and very powerful calculators which support a wide range of mathematical analysis and manipulation tasks. Unfortunately, while linear keyboards are fine for use with numerical calculators, the same cannot be said for CAS. The need to transcribe mathematical expressions, an inherently two-dimensional arrangement of symbols on a page, into a one-dimensional sequential form presents challenges for users of these systems [11].

In our work, we explore the design of a pen-math system, MathBrush. Features of MathBrush include:

- A pen-based interface that allows users to draw handwritten mathematical expressions, to correct recognition results, and to interact with a back-end CAS.
- Recognition algorithms to support the transformation of hand drawn math into MathML [2], a mark-up language that represents the underlying math expression and that can be used to transmit expressions to backend CAS.
- Algorithms that instantiate ambiguous input, such as the use of ellipses to represent a series.
- Trainable recognizers, which can be customized to an individual's handwriting.
- A component architecture to experiment with various math recognition algorithms.

To the best of our knowledge, the MathBrush system is the only pen-based mathematical system that combines recognition of handwritten input and integration with multiple CAS to do significant mathematical computation.

In this paper, we describe the system architecture of the MathBrush system, including its component-based

approach to recognition and to interaction with CAS. We describe the recognition algorithms used in MathBrush to produce a working pen-math problem solving system. We highlight unique recognition challenges that arise due to our goal of back-end CAS integration. Finally, we describe an evaluation study and present ideas for future extensions of our system. Together, these contributions inform those researchers seeking to build effective on-line recognition systems that support interactive mathematical problem solving.

2. Related Work

Whether on-line or off-line, mathematics recognition systems commonly structure the recognition process in three phases [4]. Given that an equation has been extracted from a document image or input by a user, the system first performs *segmentation* to separate individual symbols. Next, *symbol recognition* seeks to identify the specific characters that comprise the math equation. Once symbols have been identified, the system performs *structural analysis*, identifying the spatial relationships between symbols. The final mathematical relationships are commonly represented in a tree or graph, which can then be parsed to produce LaTeXor some other representation of the equation being recognized.

In the off-line domain, a number of researchers have studied the various processes involved in the recognition of mathematical expressions. As off-line recognition is not the focus of this paper, the interested reader is referred to [4] for a survey of research on mathematics recognition. As well and more recently, the InftyReader research project has described a number of innovations in mathematics recognition from scanned documents [8, 18].

In the on-line domain, current math recognition systems can be characterized based on two distinct primary goals. A set of systems exist that seek to simplify the process of creating typeset math expressions on computers. Other systems, including our MathBrush system, seek to recognize math expressions for the purposes of performing mathematical operations through a pen-based interface.

Many computer programs, including LaTeX, MS Word, and Internet Explorer, support the display of typeset mathematical expressions. To create typeset expressions, a user typically transforms the desired two-dimensional expression into a one-dimensional text string. Both characters and commands are combined, either through some interface as in Microsoft's Equation Editor or within the text string as in LaTeX, to produce an expression that contains the characters arranged appropriately as specified by the commands. A set of research systems exist that support the creation of these typeset equations using pen-computers, including work by Chan and Yeung that uses structural matching [3], a handwriting interface in the InftyEditor [15], and the Freehand Formula Entry System, FFES [14].

Beyond simply typesetting a math expression, some recent work has explored the possibility of using pen-based computers to support mathematical problem solving. One such system is the MathPad^2 system [6], an application for creating mathematical sketches. MathPad^2 allows the users to create their hand-written mathematical expressions using familiar math notation and free-form diagrams. Users can then create associations between the equations and diagrams. The diagrams are animated, and the mathematical equations specify the behaviour of the animation.

While animating diagrams allows users to see a physical interpretation of abstract mathematical concepts, mathematical manipulations such as evaluating, approximating, expanding, factoring, and other common manipulations represent another area where pen-math systems could aid problem solving. MathJournal [20] appears to be the first commercial system for doing mathematics on Tablet PC. MathJournal recognizes and interprets diagrammatic and graphical representations of some engineering and mathematical problems, but has limited mathematical and problem solving capabilities. More recently, the MathReco [19] system demonstrated support for evaluating and solving mathematical expressions. The system included a rule-based recognizer for mathematical expressions, and typical editing features. It also allows an input expression to be graphed, a single variable within an equation to be evaluated, or a definite integral to be approximated. These operations were accomplished using either a built-in math engine or Mathematica. However, once the math engine has evaluated or approximated the output, no further manipulation of the output is supported.

3. System Architecture

The main system modules that make up MathBrush consist of a user interface, a Character Recognizer (*CR*), a Structural Analyzer (*SA*) that interacts with a Matrix Analyzer (*MA*), a CAS interface tool, and finally a mathematics rendering tool. These modules and their interdependencies are depicted in Figure 1. MathBrush has been designed both to do math and to evaluate different recognition algorithms and semantic analysis approaches. Each of the components has standard input and output APIs to facilitate their replacement and evaluation. For example, the Microsoft Tablet SDK provides a rich library for text recognition and Microsoft provides guidelines [1] for designing recognizers that can extend and be compatible with theirs. Our character recognizer supports the standard

Tablet PC recognizer API which allows its replacement with a more advanced recognizer when available. For input, the *SA* accepts a set of bounding boxes with character candidates for each box, and as output it generates a standard MathML representation of the mathematical expression. In the remainder of this section, we provide a brief overview of the components of MathBrush.

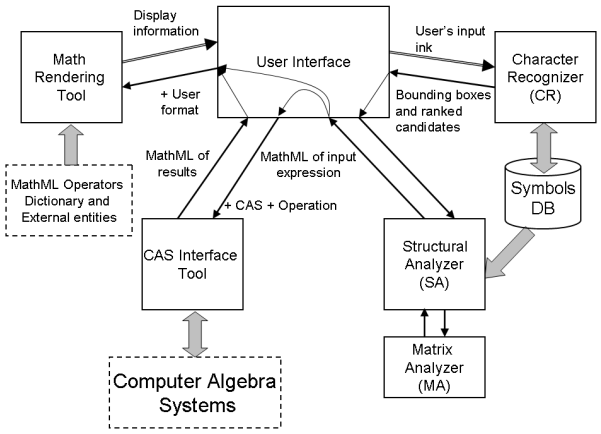

Figure 1. MathBrush System Components

The MathBrush user interface module, shown in Figure 2 receives ink from the user, collects the user's interactions and commands (via context-sensitive menus), and ultimately renders the results back to the user. It also allows the user to interact with the output expression or part of it for further manipulation.

As is typical in math recognition systems, MathBrush explicitly separates symbol recognition and structural analysis. Segmentation and symbol recognition are performed by the character recognizer (CR). The CR returns a set of bounding boxes and an n-best list of symbols. The interface module displays the recognition results to the user and allows for correction of the results, as shown in Figure 3. As well, the *CR* is trainable. The user can provide samples that can improve the recognition of symbols one draws in a special way. Figure 4 shows the interface for training the *CR*.

Corrected character recognition results are then passed to the structural analyzer. The *SA* processes the input and constructs a well formed mathematical expression. If it determines that the input expression is a matrix, it passes the input to the *MA*. The *MA* handles the construction of the matrices using the provided elements. If the user did not provide all the elements (using dots and existing elements to represent the matrix in short form), the *MA* inferrs the missing elements. Presentation MathML corresponding to the expression is generated by the analyzers and is passed back to the interface module.

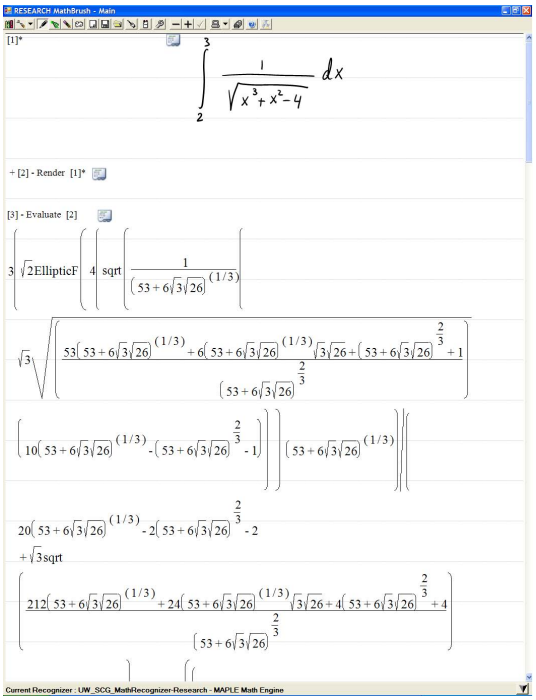

Figure 2. MathBrush User Interface Handling Long Output

Users can invoke operations on the recognized expressions. The interface module sends the MathML representation together with the specified operation to the CAS interface module. This module interacts with the CAS and returns the computed results in presentation MathML. Simple heuristics are used to render the MathML output expressions, as shown in Figure 2. Currently we support Maple and Mathematica.

4. Design and Implementation Challenges

The purpose of the MathBrush system is both to design an effective system for pen-based access to the mathematical capabilities of one or more CAS and to study recognition technology and interface designs that most effectively support a pen-math environment. Building a complete pen-based mathematical system generates many challenges on different levels. For example, on the interface level we wish to know how we can seamlessly access commands from different CAS using the pen, how to display large expressions which often come back from a CAS, how to allow interaction with the results (or parts of the results) in a natural way, and how to use the pen to facilitate operations and manipulation of plots. Independently there are also issues related to the recognition of the handwriting, particularly if the input

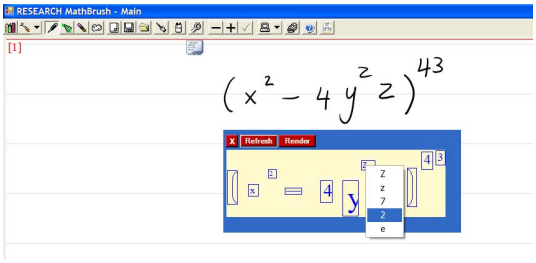

Figure 3. Character Recognition In MathBrush

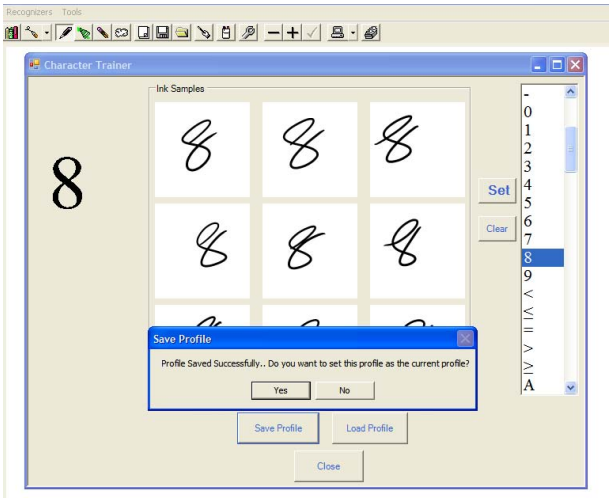

Figure 4. Character Recognizer Training in MathBrush

contains short forms of expressions (for example using $\cdots$ when constructing matrices) and how to resolve semantic ambiguities. In this section, we describe the challenges in building a pen-based system that interacts with the features available in a CAS and the decisions that were made to address these design challenges.

4.1. Input Recognition

4.1.1 Character Recognition

There are two components that perform recognition in MathBrush. In this section, we briefly describe our character recognizer, which shares many features with similar systems reported in the literature. Chan and Yeung detail the various approaches to segmentation and character recognition [4].

The character recognition module combines several existing methods found in the literature. The recognizer involves three phases: stroke preprocessing, segmentation and finally matching. The preprocessing of strokes include:

- Stroke joining - where broken strokes due to hardware or user's hesitation are joined using alignment and timing data
- Re-sampling - where input points are resampled in a way that preserves end- and cusp-points
- Trimming - where end points of a stroke are trimmed if they exhibit high curvature
- Smoothing - to prevent jitter caused by hardware or user hesitation
- Normalization - where we normalize input in order to preserve aspect ratio.

Segmentation is done by first estimating the likely number of strokes which make up the input symbol followed by a process of feature extraction. The estimation of stroke numbers uses proximity and vertical stacking of strokes (groups of strokes that appear to be stacked vertically, such as in +/-, =, or "equivalent to"). A *confusion matrix* is used to eliminate the possibility of recognizing symbols which include strokes also included in other symbols. This helps to prevent the reporting of F,- or L,= instead of a correct E.

Feature extraction uses information such as width, height, angle between end points, and width to height ratio, which are extracted from the input strokes. Every group of strokes is weighed by comparing its features to the features from the database. Both processes together generate a ranking of candidates.

The recognition phase analyzes characters using basic elastic matching [16], deformable template matching [9] and structural chain code matching [3]. Our elastic matching algorithm uses information about both point-to-point and tangent vector comparisons. Following [12], a weighted measure is also included where points which lend a symbol its characteristic shape are weighted higher than points which may be present in any symbol. The deformable template matching algorithm assigns points to a circular Gaussian distribution from which the probability of the model matching the input can be determined. Costs are assigned for moving model points, model points lying on white space, and how well the model matches the input. Finally, the structural chain code matching algorithm is also implemented in the typical fashion, breaking an input into intervals with an assignment of a numeric code to each interval based on stroke direction in that interval. The sequence of these codes is then used to determine which model's chain code most closely matches that of the input. We combine these classifier results using a hand-tuned numerical voting scheme to produce an optimal n-best list.

4.1.2 Structural Analysis

Various techniques have been proposed for structural analysis of mathematical expressions. For a somewhat exhaustive list of previous techniques, the interested reader is referred to [4]. Our approach is based on tree rewriting, as described by Zanibbi et al. [21]. Zanibbi et al.'s approach is a two-pass parser. During the first pass, the algorithm constructs an initial tree representing the math equation, called a baseline structure tree. In its second pass, it then applies a set of transformations to rewrite the tree into an appropriate form and outputs a LaTeXstring. Transformations include aggregating symbol such as "lim" into limits, "sin", "cos", etc. into trigonometric functions, analyzing containment for symbols such as square roots and fractions, and performing other modifications needed to output correct LaTeX.

Our goal, however, is to support mathematical manipulations via back-end CAS. Allowing interaction with CAS to perform meaningful mathematical manipulations of input expressions requires an extension of Zanibbi et al.'s approach. To justify the need for additional recognizer logic, consider the expression $120x^2$, a simple mathematical expression which can easily be rendered in a straightforward fashion in LaTeXas \$120x^2\$. However, to interact with a CAS, our structural analyzer must also recognize that 120 is a term, followed by an implicit multiplication, and then another term, x, which is squared. In contrast, the expression 1204^2 must be parsed as a single term, 1204, which is squared. Our output format is MathML. To correctly interact with a CAS, the MathML output must be mathematically well-formed. In other words, the output must include term grouping, implicit operations, balanced parentheses, integrals with matching dx terms, etc. For example, for correct evaluation of our previous example, $120x^2$, the MathML output must be:

```
<math xmlns='http://www.w3.org/1998/Math/MathML'>
  <mrow>
    <mn>120</mn>
    <mo>&InvisibleTimes;</mo>
    <msup>
      <mrow><mi>x</mi></mrow>
      <mrow><mn>2</mn></mrow>
    </msup>
  </mrow>
</math>
```

Note the presence of the *&InvisibleTimes* operator, the aggregation of 120 into a single term, and the msup item that groups x with its exponent. More complex expressions including integrals or parentheses require checking for additional terms to avoid crashing the backend CAS.

To ensure well-formed MathML, we perform four parsing steps in our structural analyzer. The first two steps, similar to past approaches, construct an initial baseline structure tree using bounding box information and then rewrite that tree based on character identities to produce LaTeXequivalent output. Our third pass then validates the mathematical structure, ensuring that parentheses are balanced, the integral terms have a corresponding variable over which to integrate, that numerators have denominators, etc. Finally, when the expression has been validated, a final parse produces the MathML expression which can then be passed to the CAS. The current version of our *SA* is an extension of work performed by one of the co-authors graduate students [10].

4.2. Input Short Forms

Once a system allows a user to input mathematics in a common form then users also will want to use shortcuts that are common to written mathematics. For example an expression of the form

$$S = 1 + 2 + \cdots + 100$$

is simple and convenient input for the sum of the first 100 integers. Of course one could also write this in a short form via $\sum_{i=1}^{100} i$. However, in the case of matrices, the use of shortcuts is often the only practical way to input large matrices having specific patterns. For example, the expression:

$$A = \begin{bmatrix} 1 & x_0 & \cdots & x_0^9 \\ 1 & x_1 & \cdots & x_1^9 \\ \vdots & \vdots & & \vdots \\ 1 & x_9 & \cdots & x_9^9 \end{bmatrix}$$

or

$$B = \begin{bmatrix} 1 & 0 & \cdots & -8 \\ 2 & 1 & \cdots & -7 \\ \vdots & \vdots & & \vdots \\ 10 & 9 & \cdots & 1 \end{bmatrix}.$$

both represent 10×10 matrices which, in their long form, require 100 entries in order to be fully specified.

An under-specified matrix, such as those above cannot be input into a CAS, which only accepts fully specified matrices. The closest one can come to a short input form in a CAS is to give a functional form for the individual entries (for example, $A = [a_{i,j}]$ where $a_{i,j} = x_{i-1}^{j-1}$, which represents the Vandermonde matrix presented above). Furthermore the problem of recognizing matrices, even when fully specified, involves understanding the implicit structure of the matrix, such as the number of rows and columns, and the nature of the terms. Our system has a separate matrix analyzer (*MA*) module which recognizes matrices and expands any shortcuts which may be present.

The above examples show that matrices present two separate recognition challenges. The first is to recognize under-specified matrices and convert these into fully specified input. The second problem is to recognize the

structure of the matrix itself from the separation of the input entries.

Matrix analysis itself involves four phases: Pre-element processing, element processing, ellipsis processing and matrix expansion. Pre-element processing involves separating ellipsis from the other terms, and generating statistical information regarding characters within the matrix. Element processing involves clustering the individual characters into elements of the matrix, and determining its initial structure. Ellipsis processing groups the ellipsis into straight lines, and associates the ellipsis with elements of the matrix. Finally, matrix expansion uses a variation of an algorithm by Sexton and Sorge [13] to transform the under-specified matrix into a fully specified matrix. The complete details of the matrix recognition module are described in [17].

4.3. Expression Manipulation

Users doing mathematics with pen-math systems not only expect to use natural notations but also expect expect to do actions similar to those now used when doing mathematics with pen and paper. In particular this includes editing the input and manipulating the output. Thus, for input we allow users to handwrite equations and support natural editing gestures such as scratch-out and the back of the pen for erasing ink input. A user is also allowed to select input ink and move it around, again an expected feature of any pen system.

Solving mathematical problems typically involves multiple steps working from one output expression to the next. It is also very common, even for a high-school student, to do actions such as: to isolate and replace specific instances of a subexpression with a new variable; to simplify only a subset of the terms in an expression; or to combine a specific set of terms together in an expression. CAS have many different commands which allow a user to do the above actions. However, the commands to manipulate or even isolate subexpressions are often clumsy at best. If the original expression is complex or long, then these operations are also tedious and error-prone [11].

For example in the simple expression:

$$\sin(x)^2 + \sqrt{\sin(x)^2 + (x-1)^{20} + \cos(x)^2}$$

if one wants to simplify only the

$$\sin(x)^2 + \cos(x)^2$$

under the square root using Maple, then one could not do this by the $simplify$ command, as it expands the polynomial. Instead (assuming that a user did not realize in advance that the sum of squares of sine and cosine was 1) to perform this task the following commands are needed:

1. assign the expression to a variable, i.e. use:
$$V := \sin(x)^2 + \sqrt{\sin(x)^2 + (x-1)^{20} + \cos(x)^2}$$
2. get the $\sin(x)^2$ term using $op(1, op(1, op(2, V)))$; which denotes the first operand inside the square root operator of the second operand of the original expression.
3. similarly get the $\cos(x)^2$ term using $op(3, op(1, op(2, V)))$;
4. simplify those terms and reconstruct V (again from its operand structure).

Doing the above with Mathematica would require a different set of commands. However, in either case the natural operation would be to highlight the subexpression, select the simplify operation, and then replace the highlighted terms by the result of the simplification (which in this case is 1).

In MathBrush, these editing operations are done by using a circle gesture or a highlighter to interact with typed content. The selection is then analyzed with respect to subexpressions on the canvas. This allows the user to interact with only a subset of the entire expression. Figure 5 illustrates how one can do the above substitution task in MathBrush.

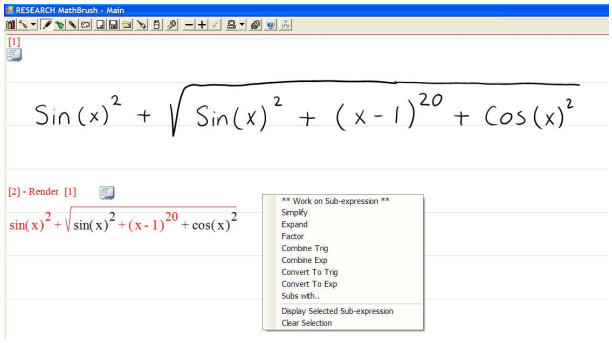

Figure 5. Expression Manipulation In MathBrush. The darker part of the typeset expression $sin(x)^2$ and $+cos(x)^2$ is the subexpression selected by the user using the highlighter from the tool box.

4.4. Other Features

MathBrush contains several other features of value during mathematical problem solving. These include plotting, logging features, and the ability to swap CAS.

Plotting is an important part of any math system. MathBrush allows the user to generate 2D or 3D plots for

its expressions. Users can draw on the plot to alter limits, as shown in Figure 6.

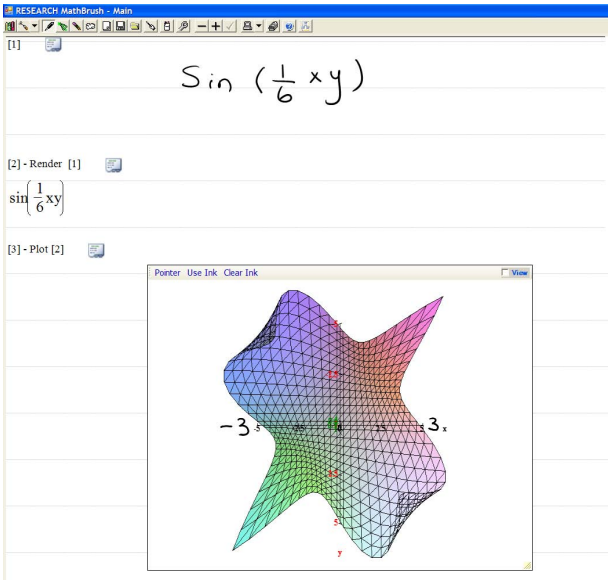

Figure 6. Plotting In MathBrush. The user is adjusting the limits of the x-axis from [-5,5] to [-3,3], by writing the new limits on the plot.

We also provide a logging mechanism that keeps track of all the user's actions in the session. The user can view the log any time during the session to help track sources of inconsistency. We have also found the log valuable in generating test data for our recognition engines.

Using multiple CAS is common during mathematical problem solving. While most CAS generate identical results for the same input, in some cases an operation may fail in one CAS but succeed in another, or the two CAS may produce answers that appear different (are semiotically distinct) yet are mathematically equivalent. This behaviour is a result of the mathematical algorithms used. Presently, using the CAS interface, to deal with failure cases for a given CAS, a user must transcribe the equation manually into a second CAS. The syntax of each CAS is unique, so transcription also involves translation of the input string, rather than simple copy and paste operations. To aid users with this occasional task, MathBrush supports interactive CAS selection which can be done anytime during the session.

5. System Evaluation

Evaluation of software systems is typically done for two reasons: to study performance and/or to determine usability. One challenge associated with performance studies for complete systems is finding some effective error metric. Should a system not recognize a given expression, a mechanism for quantifying the error is necessary to allow comparison to other systems. One alternative is to use some measure of the number of edits needed to correct results, but this value is dependent on either the expertise of the user or sophistication of automatic edit distance analysis (depending on whether the a human observer classifies edit distance or it is automatically extracted). Another option is to use whitebox testing of each recognizer component on a publicly available corpus. However, we found no standard corpus of pen-math data.

We did perform a study to verify the usability of MathBrush and to validate the decisions which had been made during the design and implementation phases. The study also helped in finalizing our ideas for future extensions and plans for a more usable interactive pen-math system. Complete details of the study can be found in [5].

The evaluation was done using a think-aloud and a semi-structured interview. Participants were undergraduate students from computer science, engineering and mathematics in our institution. As is typical in think-aloud style descriptive evaluation, we focussed on a relatively small sample subject population, and explored users' attitudes in depth [7].

During our evaluation, the participants commented that they can use a system like MathBrush to do their problem solving. They liked the ability to manipulate hand drawn and typeset math expressions, and several commented on the benefits of having it available during first year math courses. With respect to the system itself, the students all disliked two-step recognition (first correcting character recognition then verifying structural analysis), preferring a single recognition process. In addition, cases where character recognition was correct but the generation to a mathematical expression failed due to parsing problems were a source of confusion.

6. Future Plans

In this section we summarize our future plans with emphasis on improving the recognition phase and extending the editing facilities on both input and output expressions.

6.1. Input Recognition

At present our input recognition first recognizes symbols with bounding boxes and then organizes these into a mathematical expression. We are in the process of constructing a single new module that merges the *CR* and *SA* modules in our design. Our new approach will use a formal grammar to capture the semantics of the handwriting in conjunction with a symbol recognizer.

We anticipate a number of benefits. Semantic analysis will be integrated with symbol recognition so that only symbols that make sense with respect to grammar semantics will be included in the results. As well, multiple parses can be considered so that users can correct subexpressions in the context of an expression tree that has the correct general form. Finally, our intent is to store the grammar in a simple format, making it easily extensible. This opens up the possibility of domain-specific semantic interpretations by swapping grammar rules in and out.

6.2. Expression Manipulation

Expression manipulation is a difficult and tedious task when done by hand and not much simpler when done in a CAS environment. This is particularly true when it comes to working on parts of large expressions. Currently we support selection of subexpressions and allow a user to manipulate the selected expression.

We are currently investigating a number of extensions for working with expressions in MathBrush. Simple manipulation using pen actions on equations seems the obvious starting point here. For example, crossing out common terms on both sides of an equation, or dividing out factors on both sides; raising both sides of an equation to a power or more generally applying a function to both sides (so still keeping the equality). The difficulty will be to have a small number of natural actions, consistent over a wide range of mathematics, rather than a large number of specialized actions each clever for specific interactions.

7 Conclusion

In this paper we have described the MathBrush system for doing mathematical problem solving using pen-based devices. The system is intended to be both useful for doing mathematics and providing a platform for research in recognition technolgies for inputting handwritten mathematical text. We have given the component architecture and illustrated a number of the challenges faced when building such a system.

References

[1] Microsoft recognizer guidelines. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/tpcsdk10/lonestar/appendix/tbconcustomrecognizer.asp.

[2] D. Carlisle, P. Ion, N. Poppelier, and R. M. (editors). Mathematical markup language (mathml) version 2.0. 2001. W3C Recommendation, http://www.w3.org/TR/2001/REC-MATHML2-20010221.

[3] K. Chan and D. Yeung. Recognizing on-line handwritten alphanumeric characters through flexible structural matching. *Pattern Recognition*, 32:1099–1114, 1999.

[4] K. Chan and D. Yeung. Mathematical expression recognition: a survey. *International Journal of Document Analysis and Recognition*, 3(1):3 – 15, 2000.

[5] G. Labahn, E. Lank, M. Marzouk, A. Bunt, S. MacLean, and D. Tausky. Mathbrush: A case study for interactive pen-based mathematics. 2008. Submitted to SBIM.

[6] J. LaViola and R. Zeleznik. Mathpad2: a system for the creation and exploration of mathematical sketches. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, page 33, New York, NY, USA, 2006. ACM Press.

[7] C. Lewis. Using the thinking-aloud method in cognitive interface design. Technical report, IBM T. J. Watson Research Center, 1982.

[8] C. Mallon, S. Uchida, and M. Suzuki. A support vector machines for mathematical symbol recognition. Technical Report PRMU-2005-192, IEICE, 2006. pp. 49 - 54.

[9] M. Revow, C. K. I. Williams, and G. E. Hinton. Using generative models for handwritten digit recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(6):592–606, 1996.

[10] I. Rutherford. Structural analysis for pen-based math input systems. Master's thesis, David R. Cheriton School of Computer Science, University of Waterloo, 2005.

[11] K. Ruthven. Instrumenting mathematical activity. *International Journal of Computers for Mathematical Learning*, 7:275–291, 2002.

[12] P. Scattolin. Recognition of handwritten numerals using elastic matching. Master's thesis, Computer Science Department, Concordia University Montreal, 1993.

[13] A. Sexton and V. Sorge. : Abstract matrices in symbolic computation. In *Proc. of Intl. Symposium on Symbolic and Algebraic Computation*, pages 318–325, 2006.

[14] S. Smithies. Freehand formula entry system. Master's thesis, University of Otago, Dunedin, New Zealand, 1999.

[15] M. Suzuki, F. Tamariand, R. Fukuda, S. Uchida, and T. Kanahori. Infty- an integrated ocr system for mathematical documents. In *ACM Symposium on Document Engineering*, pages 95–104, 2003.

[16] C. C. Tappert. Cursive script recognition by elastic matching. *IBM Journal of Research and Development*, 26(6):765–771, 1982.

[17] D. Tausky, G. Labahn, E. Lank, S. MacLean, and M. Marzouk. Ambiguity in matrix recognition. In *Proc. of SBIM*, 2007.

[18] S. Toyota, S. Uchida, and M. Suzuki. A structural analysis of mathematical formulae with verification based on formula description grammar. *Document Analysis Systems VII, Lecture Notes in Computer Sciences*, 3872:153 – 163, 2006.

[19] A. van Dam. Demo. of math recognition at the 2007 workshop on pen-centric computing research, March 2007.

[20] XThink. Mathjournal. www.xthink.com.

[21] R. Zanibbi, D. Blostein, and J. Cordy. Recognizing mathematical expressions using tree transformation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 24(11):1455–1467, 2002.