PenCalc : A Novel Application of On-Line Mathematical Expression Recognition Technology

Kam-Fai Chan & Dit-Yan Yeung
Department of Computer Science
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
E-mail: {kchan,dyyeung}@cs.ust.hk

Abstract

Most of the calculator programs found in existing pen-based mobile computing devices, such as personal digital assistants (PDA) and other handheld devices, do not take full advantages of the pen technology offered by these devices. Instead, input of expressions is still done through a virtual keypad shown on the screen, and the stylus (i.e., electronic pen) is simply used as a pointing device. In this paper, we propose an intelligent handwriting-based calculator program with which the user can enter expressions simply by writing them on the screen using a stylus. In addition, variables can be defined to store intermediate results for subsequent calculations, as in ordinary algebraic calculations. The proposed software is the result of a novel application of on-line mathematical expression recognition technology which has mostly been used by others only for some mathematical expression editor programs.

1. Introduction

Nowadays, some pen-based mobile computing devices, such as personal digital assistants (PDA) and other handheld devices, are becoming popular. For example, as of September 2000, the customer base of PalmOS PDAs from 3Com and its strategic partners had already exceeded 8.5 millions. Mobile devices based on Microsoft's Windows CE, Symbian's EPOC, different versions of Linux, and some other platforms are also reaching out to a rapidly expanding market. It is estimated that there will be 33.7 million handheld devices worldwide by 2004.

One of the most distinguishing characteristics of these mobile computing devices is the use of a stylus (i.e., electronic pen). However, almost all the calculator programs available in existing devices do not take full advantages of the pen technology offered by these devices. Instead, input of expressions is still done through a virtual keypad shown on the screen, and the stylus is simply used as a pointing device.

To overcome this shortcoming, we propose to develop an intelligent handwriting-based calculator program that can recognize handwritten mathematical expressions and perform appropriate calculations accordingly. In addition, by defining variables, intermediate results can be stored for subsequent calculations.

In this paper, we will first review some related work. Then, we will describe some characteristics of mathematical expressions. Afterwards, we will give an overview of the recognition process and our particular implementation. Finally, we will present some experimental results followed by concluding remarks.

2. Related work

Mathematical expressions are two-dimensional structures composed of special symbols and Greek letters in addition to English letters and digits. Hence, input of mathematical expressions into computers is often more difficult than input of plain text.

With such a large number of characters and symbols, the commonly used type of keyboard has to be specially modified in order to accommodate all the keys needed, as done in [5]. Another method is to make use of some extra keys in the keyboard (e.g., function keys) along with a set of unique key sequences for representing other special symbols, as in [7]. Yet another method is to simply define a set of keywords to represent special characters and symbols, as in LaTeX [6]. However, working with specially designed keyboards or keywords requires intensive training and practice.

Recently, with the advance of handwritten mathematical expression recognition technology [3], input of expressions can be made easier as we can simply write them on the screen using a stylus and the computer is able to recognize them automatically. Despite this, such technology has only been used in limited applications, mostly in some editor programs for mathematical expressions.

For example, Nakayama [8] developed a pen-input mathematical formula editor to ease the problem of entering expressions into a computer. The system allows the user to enter characters and symbols in any order. It uses a pattern matching algorithm to recognize the handwriting and then all the information is kept in a table. During the display of the expression, objects are then checked from left to right, top to bottom and then translated into character strings.

Once we manage to obtain the internal structure of an expression, in principle we can also perform some other actions besides translating the expressions into different formats. One of the potential actions is to evaluate the expressions and perform some calculations accordingly.

Currently, there exists a calculator program, i.e., PenOffice's RiteCalc [9] from ParaGraph, which also allows the user to perform simple mathematical operations just by writing a formula. However, the operations it can support at the moment are only basic arithmetics. In addition, the input must be in one-dimensional form. As a result, the technology required is basically symbol recognition together with some simple segmentation techniques.

3. Characteristics of mathematical expressions

Mathematical expressions are two-dimensionally structured patterns composed of different symbol classes. This nature and some other properties make their recognition non-trivial in many ways. Here are some examples:

1. Several letters together may form a unit, like some trigonometric functions (such as `tan`, `sin` and `cos`). Before considering a group of letters as a concatenation of variable names, we should first check whether they together form a function name.

2. Some fence symbols, such as parentheses, group the enclosed units into one single unit. For example, "$(a+b)$" is a unit which holds the sum of a and b.

3. Some binding symbols, like the fraction line, $\sqrt{}$ and $\sum$, dominate their neighboring expressions. For example, in "$\sum_{i=1}^{10} i$", three units, i.e., 10, "$i = 1$", and i are bound to the symbol $\sum$ which gives meaning to the expression as the sum of 1, 2, ..., 10.

4. The relationships among symbols in a mathematical expression sometimes depend on their relative positions. For example, in the expression "a^b", b is the superscript of a representing a raised to the power b. However, in "a_b", b is the subscript of a representing only a variable name. On the other hand, "$a\,b$" is used to represent the implicit multiplication of a and b.

5. The same group of characters may have different meanings in different contexts. For example, "dx" has different meanings in "$\int x^2 dx$" and in "$cy + dx$".

4. Overview of the recognition process and our implementation

After writing a mathematical expression on the screen, what we get is simply a sequence of points expressed in x-y coordinates. In order to extract the final structure of the expression from the point sequence, we have to go through two major stages in mathematical expression recognition, namely, *symbol recognition* and *structural analysis*. Figure 1 illustrates such a process with a simple example.

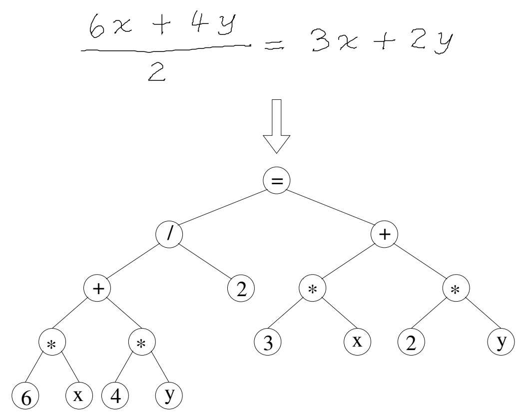


Figure 1. Obtaining the parse tree of an expression from a sequence of points

4.1. Flexible structural matching for symbol recognition

In on-line handwriting, each sequence of points between a pen-down and the next pen-up represents a stroke. Some symbols contain only a single stroke, while others have multiple strokes. It is in general quite trivial to group some strokes together into a symbol when they overlap with each other, e.g., 'f' and 'x'. However, for those symbols which contain separate strokes, such as 'i', 'j' and '=', we may have to find a window of some appropriate size in advance. When certain strokes fall within a single window, we may believe those strokes in fact form a symbol.

With proper segmentation, we can then apply flexible structure matching for recognizing the symbols. First of all, we have to extract the structure of the symbol. As shown in Figure 2, the on-line data of character ‘3’ consists of only a sequence of points. We first convert the point sequence into a chain code and then group it into two primitives. The resulting structure means that the character ‘3’ has only 1 stroke and there are 2 primitives in the stroke. The first primitive (down, 7) represents a *down* curve (going clockwise) with the ending point in the direction 7 (south-east; east is 0, north-east is 1, and so on) of the starting point. The second primitive is also a *down* curve, but with the ending point in the direction 5 (south-west) of its starting point (the ending point of the first primitive).

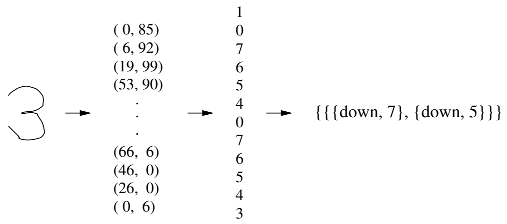


Figure 2. Steps taken to extract the structure

Afterwards, we will match the structure with those in the model set. If no result is found, we may deform the pattern into different ways in order to increase the chance of finding a match. Basically, there are four levels of structural deformation, such as no deformation, primitive type deformations, directional deformations, and simultaneous type and directional deformations. Here we have to emphasize again that the search will stop once a match (or matches after deformation is performed) has been found. As an example, a regularly written ‘T’ is likely to get correct classification during the first level of matching. However, in order to illustrate the generality of our scheme, we will use this simple character anyway as an illustrative example. Figure 3 shows how the character ‘T’ can be deformed to look like other characters by simply deforming its primitive types.

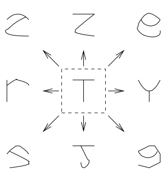


Figure 3. Relaxed versions of ‘T’ as a result of applying type deformations

In some cases, such as the occurrence of ambiguity, we have to do some postprocessing before making the final decision.

4.2. Hierarchical decomposition parsing for structural analysis

It is well known that parsing can be done in polynomial time with Earley’s algorithm while most of the other types of parsers take exponential time. However, exponential parsers can still be fast when the length of the input sentence to parse is short. To bring this idea further, a long sentence can usually be broken up into shorter sentences that can be parsed separately. Afterwards, the final structure will be constructed by combining all the intermediate structures together. Figure 4 illustrates how our proposed hierarchical decomposition parsing works with the conventional parse tree shown in Figure 1.

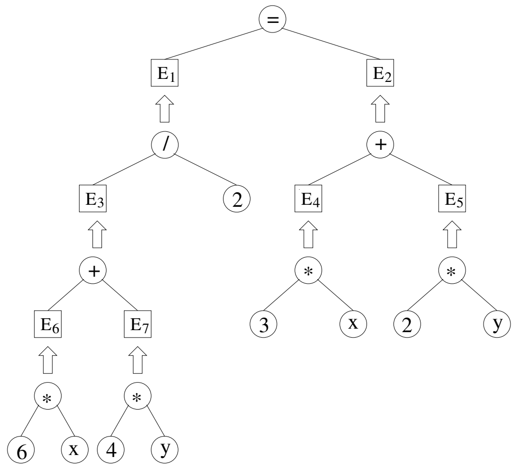


Figure 4. An example showing how hierarchical decomposition parsing works

4.3. Summary

Our current implementation of the symbol recognition module uses a simple yet robust structural approach, called flexible structural matching. This approach is designed to achieve reasonable speed, fairly high accuracy and sufficient tolerance to variations. At the same time, it maintains a high degree of reusability and hence facilitates extensibility. For the structural analysis module, we apply a syntactic approach, called hierarchical decomposition parsing, to obtain the syntactic structures of mathematical expressions. This method uses three main ideas, namely, left-factoring, binding symbol preprocessing, and hierarchical decomposition, to make parsing significantly more efficient. In addition, we incorporate an error detection and correction mechanism into the system to make it more robust. Figure 5 summarizes our implementation for the recognition process.

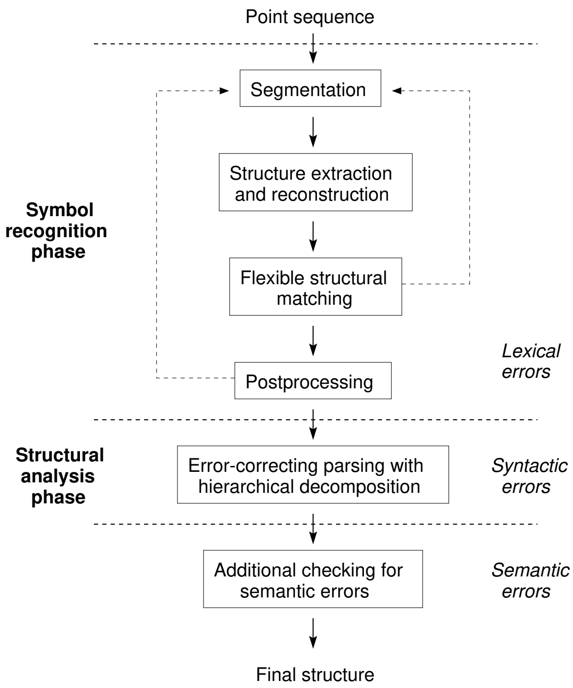


Figure 5. Overview of our implementation for the recognition process

Due to space limitation, we will not explain these techniques in detail in this paper. For those who are interested, details can be found in our previous papers [1, 2, 4].

5. Experimental results

Experiments were performed to test the accuracies of the two major modules. It is shown that the accuracy of our character recognition module is 97.40% over 9300 characters (10 digits, 26 uppercase letters and 26 lowercase letters written by 150 writers) [1]. If the rejected cases are excluded from the calculation, the rate is increased to 98.07%. At the expression level, our system can achieve an accuracy of 97.99% over 600 regularly written expressions (15 different expressions from each of the four domains written by 10 writers. The four domains are elementary algebra, trigonometric functions, analytic geometry, and indefinite integrals) [2]. Figure 6 shows one of the 600 expressions used in the experiment.

In addition, we have built a prototype system (called *PenCalc*). It is an intelligent handwriting-based calculator program currently running on desktop computers. It is being ported to PDAs, with Pocket PC and PalmOS devices being the first ones. As of now, the program can recognize mathematical expressions with the following characteristics:

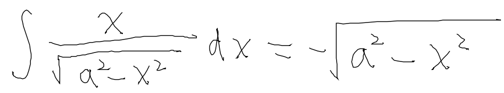


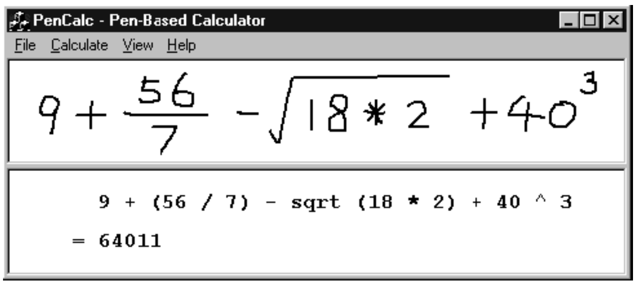
Figure 6. An expression written by one of the 10 writers in the experiment

- basic arithmetics,
- expressions involving real number, fraction and percentage, and
- expressions involving exponent and square root.

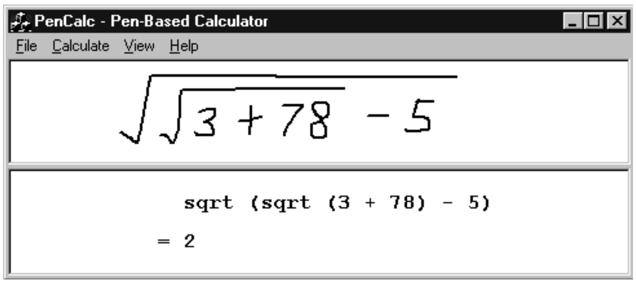
In the following, we will use some illustrative examples to highlight the uniqueness of our proposed application when compared against conventional calculator programs.

5.1. Better user-friendliness

Ordinary calculators become very inconvenient even for slightly more complex mathematical expressions. In contrast, our system provides a more user-friendly and natural interface for entering expressions. Figure 7 shows some such examples.



(a) An expression involving basic arithmetics, fraction, exponent and square root



(b) An expression involving nested square root

Figure 7. Some examples of slightly complex mathematical expression

5.2. Easier memory manipulation

Instead of using such keys as M+, MR, and MC to provide a primitive memory mechanism for manipulating intermediate values, typically just for summing up a sequence of numbers, our system provides convenient ways for defining variables through pen input for more general operations. Figure 8 shows how a variable can be defined in a very straightforward manner.

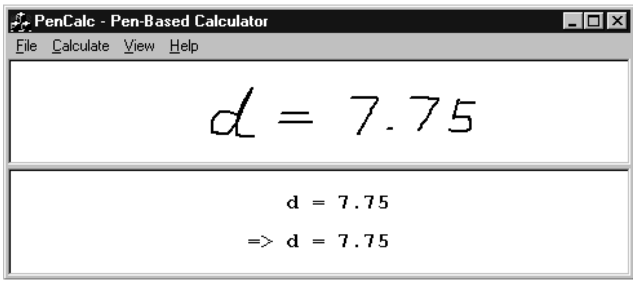


Figure 8. A variable can be defined easily

Afterwards, the variable (with value assigned) can be used to compose other expressions for evaluation. Figure 9 illustrates how the variable is used in a subsequent calculation.

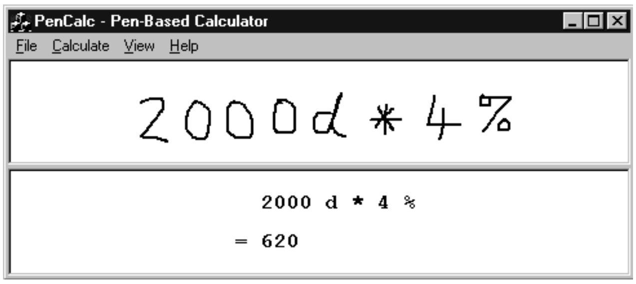


Figure 9. The variable is used in a subsequent calculation

Multiple variables can be defined and used in a similar way. For more examples, please refer to the following URL:

http://www.cs.ust.hk/pencalc

6. Conclusion

Conventional calculator programs usually require the user to enter expressions in certain fixed order through a virtual keypad. Hence, in order to increase efficiency, intensive training is certainly needed. With the handwritten mathematical expression recognition technology, input of expressions becomes easier because it requires no more than what we have already learned in mathematics classes.

Besides the application discussed in this paper, the technology can potentially be used for some other interesting applications as well, including intelligent tutoring systems in mathematics and handwriting-based user interfaces for some existing mathematical systems such as Mathematica, MAPLE and MATLAB.

Acknowledgments

The research described in this paper has been partially supported by the Research Grants Council of Hong Kong, China (Project no. HKUST 6081/97E).

References

[1] K. F. Chan and D. Y. Yeung. Recognizing on-line handwritten alphanumeric characters through flexible structural matching. *Pattern Recognition*, 32(7):1099–1114, July 1999.

[2] K. F. Chan and D. Y. Yeung. An efficient syntactic approach to structural analysis of on-line handwritten mathematical expressions. *Pattern Recognition*, 33(3):375–384, Mar. 2000.

[3] K. F. Chan and D. Y. Yeung. Mathematical expression recognition: A survey. *International Journal on Document Analysis and Recognition*, 3(1):3–15, Aug. 2000.

[4] K. F. Chan and D. Y. Yeung. Error detection, error correction and performance evaluation in on-line mathematical expression recognition. *Pattern Recognition*, 34(8):171–184, Aug. 2001.

[5] F. Grossman, R. J. Klerer, and M. Klerer. A language for high-level programming of mathematical applications. In *Proceedings of the International Conference on Computer Languages*, pages 31–40, Miami Beach, FL, 1988.

[6] L. Lamport. *LaTeX – A Document Preparation System – User's Guide and Reference Manual*. Addison-Wesley, Reading, MA, 1985.

[7] Y. Nakayama. Mathematical formula editor for CAI. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computer Systems*, pages 387–392, Austin, TX, Apr. 1989.

[8] Y. Nakayama. A prototype pen-input mathematical formula editor. In *Proceedings of ED-MEDIA 93 - World Conference on Educational Multimedia and Hypermedia*, pages 400–407, Orlando, FL, 23–26 June 1993.

[9] ParaGraph International Inc. PenOffice, 1999.