HANDWRITEN MATHEMATICAL EXPRESSION RECOGNITION USING GRAPH GRAMMARS

by

Mehmet Çelik

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

Sabancı University

Spring 2010

HANDWRITEN MATHEMATICAL EXPRESSION RECOGNITION USING GRAPH GRAMMARS

APPROVED BY:

Assoc. Prof. Berrin Yanıkoğlu, (Dissertation Supervisor)

.....

Prof. Aytül Erçil

.....

Assis. Prof. Hakan Erdoğan

.....

Assis. Prof. Hüsnü Yenigün

.....

Assis. Prof. Cemal Yılmaz

.....

DATE OF APPROVAL: .....

Abstract

This thesis presents a graph grammar approach for the recognition of handwritten mathematical expressions. Pen based interfaces provide a natural human computer interaction; interfaces for entering mathematical expressions are no exception to that. The problem is challenging, as it includes the sub-problems of character recognition (OCR) and 2-dimensional structure understanding. Thus, on top of the problems of the standard OCR systems, such as high variation in character shapes, the two dimensional nature of a mathematical expression brings further ambiguity.

We use graph grammars for structural understanding of the expressions in order to represent as much information as possible in the parse process. Representing input expression as a graph protects the geometrical relations among the symbols of the input, while alternatives include methods for linearization of the input which may introduce critical errors into the parse process. Also graph grammars have the advantage of flexibility over procedurally coded parse systems. Another important aspect of our system is the fact that all alternative parses are evaluated and the one with maximum likelihood is selected as the intended expression. The likelihoods are estimated according to OCR confidence scores and structural relationships statistics.

The segmentation step precedes the parse process, and segments and groups strokes collected from the Tablet input, according to timestamps and distance in space respectively. Then, the segmented symbols are recognized by the OCR engine which uses offline (image) features to allow for flexibility in time dimension, such as adding extra strokes and symbols anytime during the equation. The extracted features are used in an ANN and SVM combination engine returning top-3 character alternatives and confidence values. The parse process expands the graph by generating new tokens with repeated application of grammar rules. At the end, one or more tokens contain the full expression, along with a confidence value based on the

2-dimensional layout of the symbols in the expression and the associated statistics of geometrical relations between symbols. These and the OCR confidence scores are used in disambiguating alternative parses.

Our approach is more powerful compared to graph re-writing systems in that all alternative parses are evaluated, rather than selecting the most likely rule application at a particular step, in an irreversible fashion. This also eliminates the need for specifying rule precedences, making system development or use of alternate grammars easier. The only limitation of our system is that segmentation errors are irreversible. That is, the parse process does not handle alternate segmentations, in order to keep the complexity of the parse process down. We alleviate this problem by providing feedback to the user as the segmentation proceeds, in real time.

Our user interface gives error correction tools to the user to correct OCR errors and it can generate LaTeX code, and MathML codes and graphical rendering of the input handwritten mathematical expression.

An extensive collection of mathematical expression and isolated symbols are collected from 15 users for 57 different expressions from a 70-character alphabet. There are, in total, 1710 mathematical expressions and 10500 isolated characters. All samples are in the natural writing styles of the users.

Özet

Bu tez matematik ifadelerin tanınması için çizge gramerlerine dayalı bir iskelet sunmaktadır. Kalem temelli arabirimler daha doğal bir insan bilgisayar etkileşimi sunar, matematik ifadelerin kaydedilmesi için kullanılan arabirimler buna bir istisna değildir. Bu problemin zorluğu karakter tanıma ile ilgili problemlerle beraber iki boyutlu yapı tanımayı da içermesindendir. Bu durumda, alışılagelmiş karakter tanıma sistemlerinin çözmesi gereken sorunların üstüne, örneğin değişken karakter şekilleri, matematik ifadelerin iki boyutlu doğası da çözümlenmelidir.

Sistemimizde çözümleme süreci içerisinde girdi tarafından sağlanmış bilgilerin mümkün olduğunca korunabilmesi için çizge gramerleri kullanılmıştır. Matematik ifadenin bir çizge olarak temsil edilmesi ifadenin sembolleri arasındaki geometrik ilişkilerin korunmasını sağlamaktadır. Diğer alternatifler yöntemler girdinin doğrusal hale getirilmesini içermektedir ve bu çözümleme sürecine kritik hataların dahil edilmesine sebep olmaktadır. Ayrıca, gramerler sahip oldukları esneklik ile yordamlara dayanan sistemlere karşı üstünlük göstermektedir. Sistemin bir diğer önemli yönü de çözümleme sırasında gramerin tanıyabileceği tüm alternatif çözümlemelerin korunması ve daha sonra en yüksek olasılıklı olanın seçilmesidir.

Çözümlemenin öncesinde gelen kesimleme işleminde karakterler zaman etiketleri ve uzayda uzaklıklarından yararlanılarak bölünmekte ya da gruplanmaktadır. Daha

sonra, kesimlenmiş semboller zaman boyutunda esnekliği sağlamak için çevrim dışı özellikler kullanılarak karakter tanıma motorunda tanınmaktadır. Sistemde bir yapay sinir ağı ve bir destek vektör makinesi beraber kullanılmakta ve ilk 3 sonuç güvenilirlik değerleri ile döndürülmektedir. çözümleme süreci, çizge grameri kurallarının ard arda uygulanıp ilk oluşturulan çizgenin genişletilmesi şeklinde ilerlemektedir. İşlemin sonunda bir ya da birden fazla sonuç muhtemel çözümleme oluşturulmakta ve bunlar arasında sembollerin 2 boyutlu düzlemde dağılımlarından hesaplanan olasılıklara bakılarak tercih yapılmaktadır.

Yaklaşımımız çizge yeniden yazma yöntemlerine kıyasla alternatif ifade tanımlarının da saklanabilmesi ile daha güçlüdür. Çizge yeniden yazma yöntemlerinde çizge geri döndürülmeyecek şekilde değiştirilmektedir. Çizge grameri kullanımı ayrıca kurallar arasında öncelik belirleme zorunluluğunu ortadan kaldırmakta ve grameri değiştirme ya da geliştirmeyi kolaylaştırmaktadır. Sistemimizin eksikliği karakter tanıma aşama- sından gelen hatalara geri dönülemiyor olmasıdır. Bu sistemin karmaşıklığını düşürmek için yapılıştır. Doğacak problemler de kullanıcıya karakter tanıma hataları ile ilgili gerçek zamanlı geribildirim yapılarak azaltılmıştır.

Kullanıcı arabirimimiz optik karakter tanıma hatalarının düzeltilmesine imkan vermekte, LaTeX kodu MathML kodu ve el yazısı ifadenin makine yazısına çevrilmiş halini üretebilmektedir.

Proje çerçevesinde 15 kullanıcıdan 57 farklı matematik ifade ve 70 farklı sembolden oluşacak şekilde örnekler toplanmıştır. Toplam olarak 1710 matematik ifade 10500 münferit sembol bulunmaktadır. Örneklerin tamamı kullanıcıların doğal el yazılarına sağdık kalınarak toplanmıştır.

Acknowledgements

I would like to thank my advisor Berrin Yanıkoğlu for providing the motivation and the resources for this research to be done.

I am also grateful to Prof. Aytül Erçil, Assis. Prof. Hakan Erdoğan, Assis. Prof. Hüsnü Yenigün, Assis. Prof. Cemal Yılmaz for their participation in my thesis committee.

Especially, I would like to than my parents, Ali Çelik and Gülten Çelik, and my sister Ayşe Çelik for their unwavering support during this work and in all my academic pursuits.

This thesis is supported by TÜBITAK (The Scientific and Technical Research Council of Turkey), under project number 107E271.

TABLE OF CONTENTS

List of Tables

List of Figures

1 INTRODUCTION

Inspite of the ever-growing place of computers and other digital devices in our lives, pen and paper still remains the most convenient way for communicating or recording information or making small calculations. The linear input of a keyboard or a point and click device such as a mouse or a trackball is not very convenient for the preparation of complex documents with graphs, figures, tables and mathematical expressions. However as the price of graphic tablets and touch screens decrease, pen based computing has become more viable, leading to increased research on pen based recognition systems.

Today's recognition systems perform very well with machine print text, but there is much work to be done in handwritten text recognition, especially with complex structures such as sketches, graphs or mathematical expressions. Mathematical expressions have a greater number of symbols to distinguish in comparison to handwritten text, and more importantly, the meaning of symbols in mathematical expression differs according to the spatial relations between them.

Mathematical expression recognition would greatly simplify the task of writing scientific articles that contain mathematical formulas. There are several commercial scientific writing platforms, such as Scientific WorkPlace® or Microsoft®'s Equation Editor. Although they use relatively similar input languages for mathematical expressions, there is still a learning curve for a platform to fully utilize its capabilities and yet they are not as convenient or natural as handwriting. With a pen based input system, anyone who can write a mathematical expression can do so on a pen computing platform without learning a new language or software for each platform.

Furthermore, there are large databases of scientific and technical papers in both online and offline form in archives. Without a mathematical expression recognition tool, processing these documents to machine understandable format would be impossible. Thus, a system that is capable of recognizing mathematical expressions is of great use for both online and offline recognition tasks.

As a problem, mathematical expression recognition is more complicated than it may first seem. An optical character recognition system, which is a problem that have room for improvement even with the state-of-the-art systems, is the first step of a mathematical expression recognition system. Thus, a mathematical expression recognition system inherits all the problems that an OCR system has: base line detection or correction, segmentation, user variations and so on. On top of these, a mathematical expression recognition system has to recognize geometrical relations between input symbols that are governed by the grammar rules of mathematical notations. Furthermore, in text recognition, the use of a lexicon can recover certain errors. For example, a segmentation error of an OCR system may be corrected by checking the words in a given language for matches. To be able to do a similar kind of error correction for mathematical expression recognition, a dictionary of mathematical expression is needed which is not feasible. For example, for a simple summation operation there is unmanageably many possibilities for the two numbers in operation. This is in fact the main reason for low accuracy of mathematical expression recognition systems, as there is no lexicon and no redundancy in the expressions.

1.1 Literature Review

Work on mathematical expression recognition has been conducted especially in the last 10-15 years, with the advances in tablet technologies. Since this is a difficult problem, significant results have been obtained as a result of long term research by

various groups. Below we provide a literature review which is organized by research groups.

DRACULAE (EFES)

One of the active groups in mathematical expression recognition is Zannibi et al. [1, 2]. In one of their later publications [2], mathematical expressions are transformed into a **baseline structure tree**. The overall system works in three stages. The first stage builds the baseline structure tree by analyzing the baselines of the components. In this tree, each node has three children: below, inline, and above. The second stage groups and labels compound groups (e.g. "sin", "123"'), from among the inline components. In the last stage, the expression analyzer analyses the expression syntax and produces an operator tree that describes an ordered application of operators to operands. Zanibbi's implementation is named Diagram Recognition Application for Computer Understanding of Large Algebraic Expressions (DRACULAE) and uses the Freehand Formula Entry System (FFES) as user interface and for symbol recognition. Worst case time complexity of DRACULAE with n symbol input is $O(n^2 logn)$.

In another paper [1], they apply compiler techniques to diagram recognition. The steps are summarized as below:

- Find linear structures in the input: Baselines are detected in the expression.
- Organize these linear structures into a tree: tree structure from baselines are generated for later compiler like processing.
- With a fixed control structure, divide the processing into lexical, syntactic and semantic analysis.
- Analysis are done on attribute trees and tree transformation techniques are used.

MathPAD(Now in MathPaper)

LaViola's system [3], MathPad, is a user-dependent mathematical illustration application. It utilizes a pairwise recognition method in conjunction with Microsoft's character recognition engine. With the pairwise recognition method instead of one classifier to recognize all symbols, many classifiers are used to determine whether a symbol is one letter or another. For example, a classifier is used to determine a symbol is "a" or "b". The parsing algorithm is based on two methods: a 2D grammar and procedurally coded syntax rules. Coded syntax rules implement a context free grammar [4]. The success rate of symbol recognition system is 95.1% while correct parsing decision rate is % 90.8 for eleven subjects with 36 expressions. With a new version of the project, now the system can identify matrices with % 91.6 accuracy [5].

MathBrush

Similar to [2], the system is developed by Labahn et al[6] and based on tree rewriting. In four steps, they transform a baseline tree into an expression tree. The work is extend Zanibbi's work, in that, system uses a symbol database for storing the structural and semantic type of a symbol. The structural information is used to determine the center of the symbol and refine the baseline finding process; the semantic type is used to determine the grammatical structure of an expression. The structural analysis step of their algorithm repeats itself until the combination of structural confidence value and recognizer confidence value reach a threshold.

MathFoR

In their approach [7], Tapia et al consider an entire mathematical expression as a connected weighted graph with each bounding box center as a node. The algorithm

starts with a minimum spanning tree and modifies this tree in successive iterations, with predefined rules such as dominance, being on the right, and distance between symbols. Grammar rules are implicitly defined in attributes of the symbols. For example, numeric symbols subscript positions are not defined and for operators like "+" no script position is defined. These implicit rules are used to find operator dominance. The system is also able to recognize matrix structures by the reserved symbols "[" and "]".

Infty

The work by Suzuki et al [8] uses a parsing process that finds a spanning tree in a virtual network. In this network, vertices correspond to symbols and edges indicate possible relationships between two symbols. Each symbol has different recognition candidates and there is more than one possible relation between two symbols; thus there is more than one edge between each vertex, defining both parent and child recognition and the spatial relation between them. Each edge also has a cost value calculated from relative position distributions of symbol pairs. Since there is several possible values for each vertices and edges, this network called a virtual network. After the completion of network generation, spanning trees are generated and the one with the lowest cost is selected as the parse tree. They report a 90% percent success rate for recognizing 123 different formulas, with an average of 34.7 characters.

Later with [9], they introduced a formula description grammar to select the correct parse tree among candidate parse trees produced by the previous structural analysis step. The grammar used in this system is one dimensional, so each tree is converted into a string before checking with the grammar rules.

Sexton et al.

In [10], Sexton et al use a database driven character recognizer and two different structural analysis methods. The first one is the projection profile cutting (PPC). This work is based on off line formulas so profile cutting is fairly robust. They use successive horizontal and vertical profile cuts on the formula to generate a parse tree. If a horizontal(vertical) cut separates an individual symbol, that symbol is put in a leaf node; if a cut creates a multi-symbol node, that node is processed again with a vertical(horizontal cut). The resulting tree is a parse tree that specifies the subcomponent relationship of the expression.

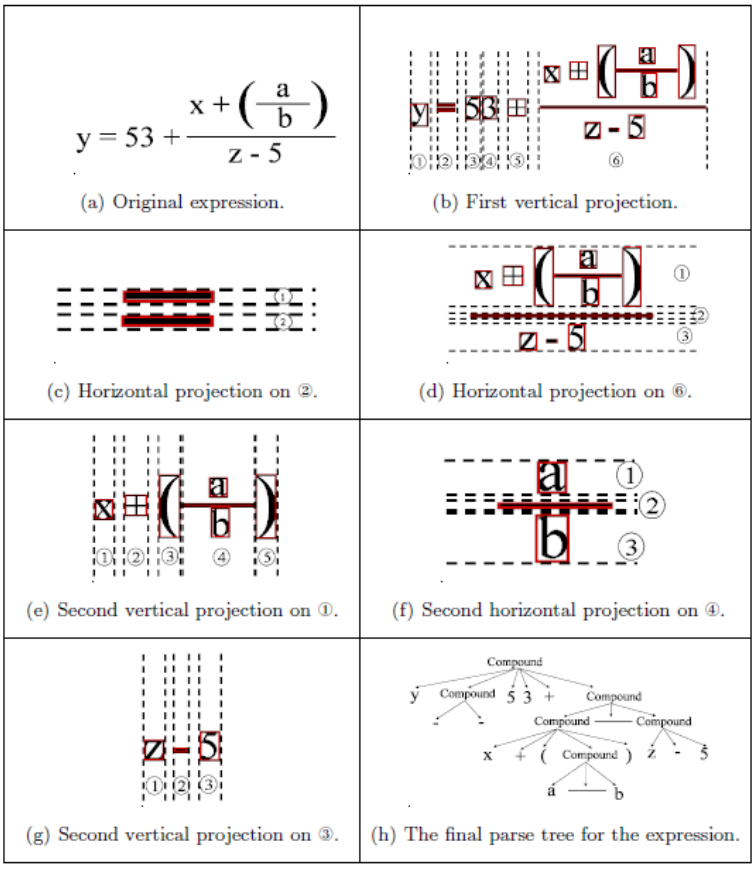

Figure 1.1: Example PPC parse tree

The second method they use is graph rewriting. They generate the initial graph

with edges that are generated on a line of sight basis. The graph rewriting process tries to apply all rules to all symbols and generates non-terminal nodes. This process repeats itself until no rule can be applied. Then a non-conflicting sequence of rewrites is converted into a parse tree.

Garain et al.

In the system of [11], the structural analysis system is divided into on line and off line phases. The online phase generates bounding boxes and center points for the symbols along with a level value which is 0 at the base level and has positive and negative values above and below base level. Also the meaning of ambiguous symbols such as dot or horizontal line and function names such as "sin", "log" are determined in the online phase. A finite automata is maintained to detect the function names. The offline phase utilizes a projection profile cutting technique where the whole expression is divided into vertical and horizontal stripes recursively until no further segmentation is possible. Finally, a context free grammar guides the merging process of segmented symbols. Success rate is 74.92% because system does not have the ability to recover from placement errors.

Grammar rules are given as follows:

```
E ->  ES | S
E ->  S^{S} | S_{S} | \frac{S}{S} | \sqrt{S}
| \stackrel{S}{S} | \overline{S} | \underline{S}
| \overbrace{S} | \ underbrace{S}
| \mathcal{RL} | \ELLIP | \ ACCENT
| \begin{array} MAT \end{array}
| AN | RL | GS | MS | PM | FW | HN | HL |
```

Where AN = Arabic numerals, RL = Roman letters, GS = Greek symbols, MS

= mathematical symbols, PM = punctuation marks, FW is function words, HN is Hindu numerals and HL is Hindu letters.

Yeung et al.

In [12], Yeung et al define precise replacement rules with a definite-clause grammar (DCG). Due to backtracking, DCG parsers are inefficient so some methods (e.g. left factored rules, binding symbol preprocessing and hierarchical decomposition) are developed to speed up the process. Left factoring grammar rules are designed in a way to prevent repetitive searches for the same term. The grammar is defined on strings but they do not clarify how they generate strings from two dimensional expressions. Detecting binding symbols (e.g. =) and decomposing the expression into sub expressions shortens the problem size for each run of the algorithm. For large sized expressions (about 30 symbols), the processing time is reported to be 0.24 seconds, while small expressions are processed under 0.05 seconds.

Others

The system of [13] uses a one-pass dynamic programming based symbol decoding and graph generation algorithm for symbol recognition. Then the system uses an A*-like tree search algorithm to generate N-best hypotheses. the search starts from a terminal node and works backwards (right to left direction).

D. Prusa and V. Hlavac introduced a system[14] where the process runs in two phases: elementary symbol detection phase and structural analysis phase. In the elementary symbol detection phase, strokes are grouped according to some distance constraint and each group is kept under 4 elements. Each candidate is processed by a freely available OCR tool and all candidates are sent to the structural analysis phase with returned labels without any elimination. The second phase is based on

a parsing algorithm that uses a 2D grammar. There is a general definition for the grammar but authors do not mention their production rules. A success rate of 97% is obtained, excluding OCR errors, but no information is given about the nature of the test expressions, e.g. number of symbols, number of mathematical constructs. In general, 2D grammar based algorithms are slower (exponential with respect to number of symbols); they report an average 0.082s for a formula recognition, but without the information about the number of symbols in each tested formula.

Vuong et al.[15] developed a progressive recognition and analysis method. Progressiveness is defined chronologically and based on two assumptions: users do not continue a higher level expression without finishing sub expressions and users do not make any corrections on the previously written expression parts. Strokes are recognized into symbols and symbols are updated into mathematical expression trees in real time as they are written by the user. Character recognition is based on original elastic structural matching [16]. Multi-stroke symbols are recognized with the help of a list of previously written strokes for possible groupings. Without taking into account the errors coming from the symbol recognizer, they have a 100% recognition rate for expressions with less than 10 symbols. However success rate drops to 80% if the expression has more than 30 symbols.

Our Approach - Mathlet

The system developed in this thesis follows a previous Master's Thesis by Büyükbayrak on mathematical expression recognition, called *Mathlet*[17]. This system is a procedurally coded system where symbols where the parse process starts by sorting every symbol according to its x-coordinate; it then goes over each symbol to find structurally significant ones, such as summation sign or fraction sign. Each such significant symbol divides input expression into subexpressions, in a recursive fashion. The user

interface provides a complete environment to write a scientific article with ease, using the Tablet PC API for text recognition, a user interface for separating figures and mathematical expressions; providing feedback and corrections. While the system is quite successful for a careful user, it has certain limitations that constrain the natural writing style: it assumes that symbols consists of single strokes as in the Graffiti system on Palm PDA OS. Furthermore, it requires that structurally significant symbols (e.g. summation sign) comes before the related symbols in the x-coordinate ordering. Finally, the procedural parsing system makes code change difficult.

Putting restrictions on the order of symbols or the number of strokes are ways that are commonly used to reduce the complexity of the problem, but they result in considerable diversion from the natural handwriting of expressions. Based on the experiences obtained with the original *Mathlet*, we aimed to reduce input constraints to minimum to give a more natural interface and flexibility and to be able to modify the system without much complication. As a result, in the current system, the only assumption about the writing is that symbols either do not intersect in space, or that they are well-separated in time. In order to make future improvements manageable, we decided to use use a method based on grammars rather than a procedural approach.

Another main principle set forth for the current work was to select the most likely parse alternative from among the possible alternatives. Handwritten mathematical expressions may be locally ambiguous, but they correspond to an unambiguous global expression. Existing mathematical expression recognition systems typically work by making the best possible decision in a greedy fashion during the parse process. As a result, they choose what seems to be the best parse in parts of the expression and can not recover from early mistakes. Our aim was to eliminate any strict decisions so as to select the statistically most likely parse alternative at the end of the parse process.

While this creates extra overhead, we chose this approach over the sub-optimal greedy approaches.

2 METHODOLOGY

For mathematical expression recognition and especially handwritten expressions there may be ambiguities in the interpretation of the expression. However, this ambiguities stems from the handwriting style of the user or sloppiness of the handwriting, while underlying mathematical expressions are unambiguous. The main idea of our approach is that if input symbols are a mathematical expression then it is unambiguous and has an interpretation. Thus, without relying on what there should be, the system can deduce the whole meaning from geometrical relations between symbols. For example, in the expression $x^y z$, both x^y (y as a superscript) and y_z (z as a subscript) are valid meanings. Then, recognizing $x^y z$ as y as a superscript and x and z in the same level with each other will be backed by the previous hypothesis that y is a superscript, which means that without assuming the general structure of the expression, the system can find the most likely meaning for input expression.

With this approach, there is no emphasis on which order the system process the symbols and there is no assumption about the structure. Furthermore, there is no precedence between rules and the system keeps every possible interpretation or possible subexpression without creating irreveresable mistakes. As in the given example, ambiguity of the geometrical relationship between y and z can be solved later in the process.

An overview of the system is give in Figure 2.1. Output of the system is a syntax tree that represents the input mathematical expression. OCR system takes a list of strokes and outputs a list of bounding boxes with recognition results. Structure recognition takes a list of bounding boxes with recognition results and a list of gram-

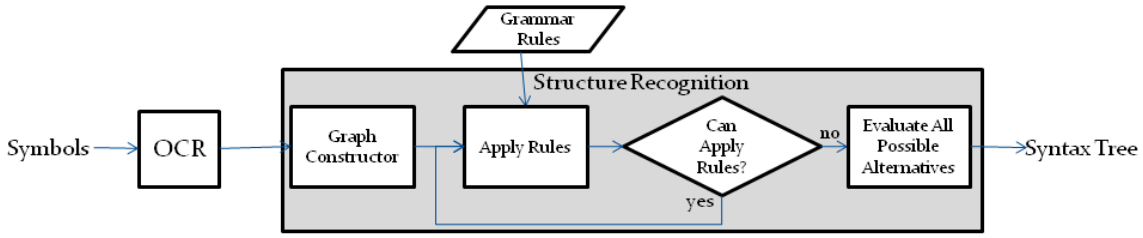

Figure 2.1: System overview

mar rules and outputs a syntax tree. In this section the terms node and token are used interchangeably.

2.1 OCR

First step of the OCR system is segmentation of the strokes to create a set of symbols. The segmentation process uses both time and space information. The second step is creating an image from strokes of a symbol, since our recognizer is based on offline features. Here, every point in the strokes of the symbol is mapped into a 32 by 32 grid that will create the image, then points are connected by lines. Then, extracted features gets fed into the classifiers. Resulting output is a list of bounding boxes with top 3 recognition results.

2.1.1 Character Segmentation

The OCR system recognizes characters as they are written by the user. After each stroke is written down, the new stroke is checked for intersections with previous strokes. If there is an intersection and the difference between time stamps of those strokes are within a certain limit, then those strokes are merged as a symbol. If there is no intersection, the new stroke gets recognized as a single stroke character. For example, in the process of writing the plus sign, after first stroke (either horizontal or vertical one) is written that stroke gets recognized as a single stroke symbol. When

the second stroke is written down, which intersects with the previous one, it gets merged with the first one and plus sign gets recognized as a symbol.

For example, if two strokes intersect by their time stamps are more than two seconds apart then those to strokes processed separately as two different symbols. Two possible intersections for an expression is shown in Figure 2.2. The intersecting strokes that creates the plus sign have to be recognized as one symbol, but the intersection between the letter y and fraction sign has to be ignored by the system. By defining a proximity threshold for time dimension y and fraction sign can be separated and plus sign can be recognized correctly.

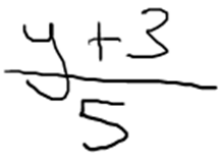

Figure 2.2: Sample intersections

2.1.2 Character Recognizer

The OCR system takes the segmented characters as input and outputs top-3 alternatives with associated confidence scores, for each recognized character. The OCR engine is developed using a Support Vector Machine (SVM) and an Artificial Neural Network (ANN) in conjunction. The SVM used in our systems utilizes a polynomial kernel, and the ANN uses sigmoid threshold functions, 1 hidden layer with 30 neurones.

Prior to recognition, preprocessing is first done to reduce size variations. This is done on the online data to reduce artifacts whereby the coordinate of each point is mapped into a fixed coordinate range. Then a character image is created from these points by connecting them.

Feature extraction takes as input the image of the resized character, ignoring time

dimension. This is done to eliminate temporal variations in the drawing of characters, as well as allowing user corrections of symbols and formula that may be done after the equation is completed. For both classifiers input features are:

- horizontal, vertical and diagonal histograms of the symbol images
- horizontal, vertical and diagonal depths of the first black pixels of the symbol images
- number of black pixels in an 8 by 8 windows over the whole symbol image
- ratio of width to height

Confusion in character recognition systems, without context information, is inevitable. Table 2.1 gives possible characters that may get confused by a recognition system. Actual confusion information depends on the selected features for classification of the characters. With no context information, some confusion between characters is accepted and can be handled by the parse process.

o	ω	1	s	+	g	2
0	w	l	5	t	9	z
		)	$\int$			
		(				
		/				

Table 2.1: The characters that are difficult to discriminate without context information

v	γ
u	v
μ	y

Table 2.2: Confused character with current feature set and classifier

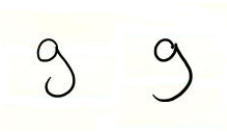

Figure 2.3: Characters 9 and g from different users

Size Normalization

One goal of size normalization is to obtain a fixed size image from which to compute features; while another one is to reduce size variations. Because very long ascenders and descenders seriously degrade recognition performance, we aim to scale ascender/ descender and body parts separately, as done in [18]. This is done by scaling down a long stroke inside the character more so compared to other parts and results in less extreme ascenders/descender.

From every letter's vertical histogram, if there is a long low value section in the histogram it is rescaled differently than the rest of the letters. Low value section of the histogram represents a vertical stroke, in the ascender and the descender letters and if it is proportionally longer than rest of the letter, this means, for example, a letter y has a long tail and it gets scaled down. The normalization process also checks the average histogram value of the rest of the letter where an ascending or descending part is found to protect thin symbols such as l or (.

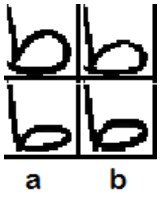

Figure 2.4: Before (a) and after (b) normalization

Ascender-Descender Characters: In typography, an ascender character is a character with a portion of it above the median line and a descender character is a character with a portion of it under the base line. Simply if a letter is taller than

the letter x it is an ascender if it is goes lower than the letter x it is a descender character.

OCR Test Results

The OCR system is designed to recognize lowercase letters, numbers, '+','-','=' signs, summation, integral and square root symbols and parenthesis, from the 46-character LaViola data set consisting of 20 samples of each letter from 11 users [19].

The success rate of the SVM system on this data is 92%. Although there are methods to generate posterior probabilities from multi-class SVM classification [20], we used a ANN to generate classification alternatives and obtain reliable recognition confidence. The ANN classifier we used is a 1-hidden layer feed-forward neural network with 30 hidden neurons. The performance of this classifier is lower compared to the SVM, with top-1 and top-3 recognition rates of 88% and 97%, respectively. Since the SVM is more successful in the top-1 performance, the OCR system uses the SVM output as the top-choice and gets the next two choices and the confidences from the ANN.

2.2 Structure Recognition

Mathematical expressions are, at least in the same field of mathematics, designed to be unambiguous; thus, expressions are highly structured which means they are governed by a grammar. The recognition process of mathematical expressions has to be guided by some kind of rule set, whether it is explicitly defined such as graph rewriting algorithms or implicitly implemented in a procedural parsing process. The first logical step is, designing a grammar for the intended range of mathematical expressions.

Our grammar has been designed with the observation that every grammatical

relation between symbols or symbol groups is defined between neighboring ones. Simply, if there is a relation between two symbols or a symbol group, there are no other symbols between them. So, each grammar rule has one central symbol and a number of neighboring relations. For example the summation symbol has three neighbors: one above, one below and one on the right of the symbol.

2.2.1 Graph Grammars

Graph grammars are first introduced separately by [21] and [22]. They provide formalism for grammatical processing of multi-dimensional data which cannot be achieved by string grammars. Despite the fact that they are introduced to solve picture processing problems, since their introduction, graph grammars have been used in areas such as concurrent systems, databases, programming languages and biology [23].

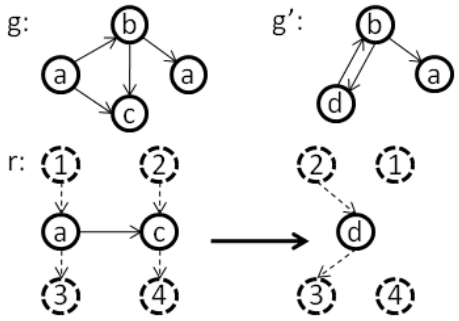

Figure 2.5: Graph g' after rule r applied to graph g. Embedding rule is graphically represented: only edges towards c and edges outgoing a are kept.

A graph grammar $G = (N, T, E, R)$ consists of a set of rules R that are defined on a graph that consists of a set of terminal and non-terminal nodes N and T that are connected with a set of edges E. A rule $r = (g_l, g_r, C, Em)$ consists of a left-hand side graph g_l and a right-hand side graph g_r, an applicability predicate C, and an embedding rule Em. The applicability predicate C (application condition) ,these conditions are constrains on attribute values of nodes and/or edges, and non-existence

of certain edges. With applicability predicates, a production rule can be restricted even if it has a match in the input graph. A production is the application of a rule r to graph g to produce g', which is denoted as $g \Rightarrow_r g'$. With a production $g \Rightarrow_r g'$, an occurrence of a subgraph g_l of g is replaced with a subgraph g_r to produce g', according to embedding rule Em, if the applicability predicate C is satisfied.

In string grammars, the placement of the production is obvious, but in graph grammars, placement of production graph g_r has to be specified. Embedding rule Em describes how to handle dangling edges (edges that lose one of their nodes after the g_l is removed from the graph) and how to connect the produced graph g_r to the existing graph. A derivation from graph g to graph g' of grammar G is a sequence of productions where $g \Rightarrow_r g_1 \Rightarrow_r g_2 ... \Rightarrow_r g'$. A graph $g = (n, e)$ is in graph grammar G iff $n \subseteq N$ and $e \subseteq E$ and there exists a derivation that can generate g with rules R.

In our case, there is only one embedding rule because all rules follow the same embedding. Normally a graph grammar rule indicates that g_l is replaced by g_r; here $g_r \cup g_l$ is added to the graph.

2.2.2 Proposed Grammar

In the proposed graph grammar, nodes represent recognized symbols and edges are spatial relationships between symbols. Node attributes include the bounding box of the symbol, position and size and the center of the symbol. These attributes are used in the applicability predicates of the rules. The parser checks every node against every grammar rule to find a matching rule and applies this rule to the graph. The parse process continues until there is no valid production.

Nodes: A node is a tuple $n = (t, c, i, A)$ where t is the type of the node, c is the identifier of the rule that constructed the node, i is a unique identifier and A is

a set of attribute values. The type of a node t is the lexical type of the symbols, such as number, letter, operator or type of the expression if node is non-terminal. The construction c is used to keep track of the rules that have been used: each node knows which rule constructed itself, so if needed, the whole history can be generated. Each box in figures 2.8 and 2.9 represents a node in the graph.

Edges: An edge is tuple $e = (t, n_1, n_2)$ where t is the type of the edge, n_1 and n_2 are nodes that are connected together by the edge. There are three types of edges:

- Spatial relationship edges: if two nodes are *neighbors*, they are connected with this type of edge(Figure 2.9,a).

- Component edges: edges between a non-terminal node and its components (Figure 2.9,b), used to help generation of syntax trees after the parse process. For example, a node representing a^2 will have component edges to nodes a and 2.

- Production edges: edges to the produced non-terminal node from terminal node (or another non-terminal node) (Figure 2.9,c). Only center node of the rule graph gets the component edge after production. For example, after the generation of the node represents a^2 node representing a will have a production edge to node a^2, since the center node of the rule that matched is base of the superscript relation.

First step of initial graph construction is token generation. Tokens are nodes of the graph that represents symbols of the input expression. It should be noted that later in the recognition process graph nodes represents recognized expressions. Tokens carries the information comes from the OCR, bounding boxes and recognition results. In addition to these, typographical centers and type information, (e.g. number, letter, sign etc.) is added to the tokens.

Second step is generating edges to connect nodes of the graph. In initial graph only neighborhood edges are generated. Edges also contains the distance information between symbols.

When the initial graph is generated from a list of symbols, spatial edges are generated between two nodes if they are neighbors, defined as having a clear line of sight between the center points of their bounding boxes. The initial graph have no production or component edges; those are added to keep track of the past productions as the parse process goes on.

Production edges and component edges are trivial to generate as rules are applied, but spatial relationship edges can not be easily generated. Edges between products of the neighbors of the component nodes and product node are generated if there is no intersection between those nodes.

Spatial relationship edges do not have any attributes. In other words, the existence of a spatial relationship edge between two nodes do not give any information about the nature of the relationship, such as being above or below each other. The advantage of our approach is that by associating spatial relationship attributes with applicability predicates, as opposed to defining global definitions for spatial relationships, each rule can have its own definition for spatial relationships categories. For example the angle to be recognized as a superscript can be selected differently for different rules.

Applicability Predicates: The most important part of the decision when it comes to applying a rule, comes from applicability predicates. For most rules, the angle and/or distance between symbols are checked.

There is two kinds of angle calculation between bounding boxes to solve the following problem. Since our predicates also will be used on produced tokens, we can not guarantee a size or width to height ratio for our token bounding boxes. As

shown in Figure 2.6, this increases the variability of the angles for same grammatical relation. First angle calculation method calculates the angle between the centers of two bounding boxes, as shown in Figure 2.6. Second one calculates the angle between the center of the closest squares that can be fitted into those bounding boxes, as shown in Figure 2.7.

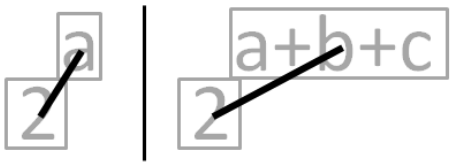

Figure 2.6: Angles between bounding boxes for the first relation

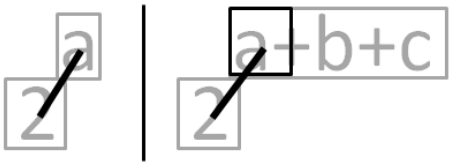

Figure 2.7: Angles between bounding boxes for the second relation

Some rules may have further checks on attribute values. For example, for the rule that checks for fractions, g_l has a central node which represents the horizontal line symbol. To be able to determine the precedence, if one of the matched neighbors comes from another fraction production, its fraction symbol (a horizontal line) has to be smaller than the fraction symbol to be included in the current production. Any constraints used in the applicability predicates are very loose, in order to keep all likely interpretations of the mathematical expression.

Since the matched nodes are kept in the graph, each rule also has a predicate that also checks the non-existence of a production edge that connects to a node which is same as g_r of the rule, to prevent matching same nodes again and generating the same product. This complicates the parse process and increases the complexity but removes the need for defining precedence between rules.

The geometrical relations for x^2,y^2 or b^2 are different, ascender, descender information is also determined and used in the parsing process of the system. Each letter has a center point for geometrical relations. In our work this point is set to the center of the bounding box for normal characters, to one third of the length below the top for descender character and to one third of the length from the bottom for ascender characters.

2.2.3 Parsing

Our parse algorithm is a fairly straight forward bottom-up process. Basically two tasks have to be done by the parser: finding a match for pattern graphs of the rules and embedding the resulting product graph. Since the pattern graph of any rule is a *star graph* (a graph that has a central node and surrounding neighboring nodes connected only to the central node), when processing a node, the parser looks for a matching rule which has the same center node. Then it checks for the neighboring nodes and applicability predicates to finalize the matching process.

Once a match is found, a new node is generated according to the rule, which then gets connected to the existing graph with component and production edges. Spatial relationship edges are generated among newly produced nodes after no possible production is left in the existing graph. Each new node inherits the neighbors of its components and spatial relationship edges between new nodes are generated separately. A parse process is depicted in Figure 2.8 and a state of the graph is given during the parse process in Figure 2.9. Nodes shown in the Figure 2.9 are duplicated to increase readability. In fact there is only one graph which has three kinds of edges.

The output is a graph where all productions are present. If the input can be defined by the grammar, then at least one node which covers all input symbols will be in the output graph. At this point our framework foresees the evaluation of

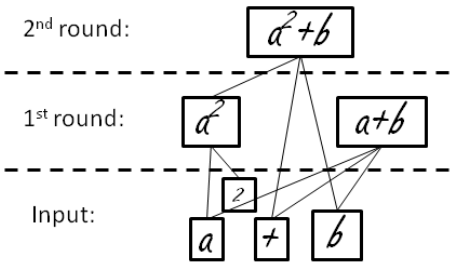

Figure 2.8: Generated nodes in each round, along with their component edges.

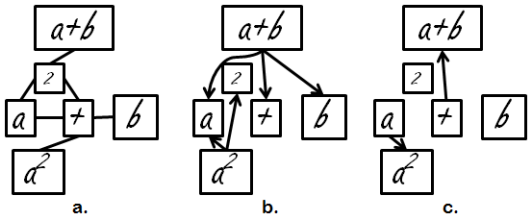

Figure 2.9: Graph after round 1, of Figure 2.8 shown in 3 parts: a) spatial edges, b) component edges, c) production edges.

the possible interpretations (all nodes that cover all the input symbols) in terms of their likelihood according to pre-learned spatial layout and OCR output probability distributions. However this step is currently not completed and we only have an unranked list of alternatives. Since component edges keep the history of productions, an expression tree can be generated if one of the alternative interpretation nodes is selected as the root and component edges are followed until it reaches a terminal node. Even if process cannot produce a node without spatial relationship edges, existing productions can be seen as hypothesis and can be used for further analysis. Like most graph grammar parse algorithms, this has exponential time complexity in the number of symbols.

The first step of the parser is generating the graph representing the input. Nodes come from a class and each carry a bounding box and recognition information, all node attributes and three lists of edges for three different edge type. Then all nodes connected to each other with the spatial relationship edges with the conditions mentioned previously. This generates the input graph, a graph of terminal nodes. At

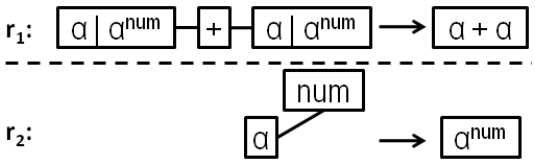

Figure 2.10: Rules used for sample parse. "|" depicts "or"

this the stage parser traverses through all the nodes in the graph and tries to match a rule. During the implementation, each rule is inherited from a base rule class, where all can be called by parser without difference and without a need for modification. When a rule is matched, the product of that rule gets added to the graph with component and production edges, but without spatial relationship edges. After every node is tested against grammar rules, spatial relationship edges are generated for produced nodes. Then the parser starts to traverse the node with the added new nodes. When no rule can be applied after one iteration of the graph parser stops and the nodes covering all the input symbols is returned. Since each node has the record of its components and what production created it, the component edges can be followed from a node to the terminal nodes to generate the syntax tree.

2.2.4 Disambiguating the Parse Alternatives

The parse process may construct more than one node that covers all of the input symbols. In order to be able to chose the best parse alternative, likelihoods of the nodes corresponding to different parses of the equation are calculated while they are constructed in the parse process. These likelihoods are calculated using the spatial distributions of size and distances of the symbols related to the grammar rule. For instance, in Figure 2.11 similar parses are constructed for the two samples. Disambiguation is done by calculating their likelihoods for the considered meanings from the spatial distributions of the components and selecting the parse with higher likelihood. For the right hand side sample, the position of the plus sign makes it

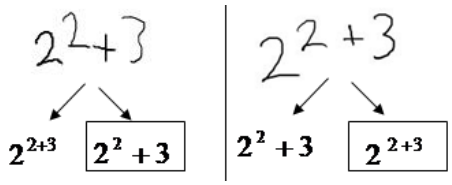

Figure 2.11: Sample disambiguations

more likely to be in the same line with the superscript 2.

In order to extract spatial distribution statistics, data is collected for the spatial distributions between symbols from a set of mathematical expressions written by different users. For example we used differences in x and y coordinates of the closest corners of the bounding boxes of letters for superscript relationship. For each such parameter, histograms are calculated from this data which are then used to approximate the likelihood functions for the spatial relationships described in each grammar rule. With each relation has a likelihood, this values are used to calculate a log likelihood for each rule production.

Bin widths for the histograms that calculated to approximate the likelihoods varies depending on the range and the distribution of the data but for a given histogram bin width is constant. Distribution of the data is taken into account to generate histograms that do not have empty bins.

Apart from the first nodes generated after symbol recognition, likelihood of each node is the average log likelihood of the relations that generated the node and the likelihoods of the components.

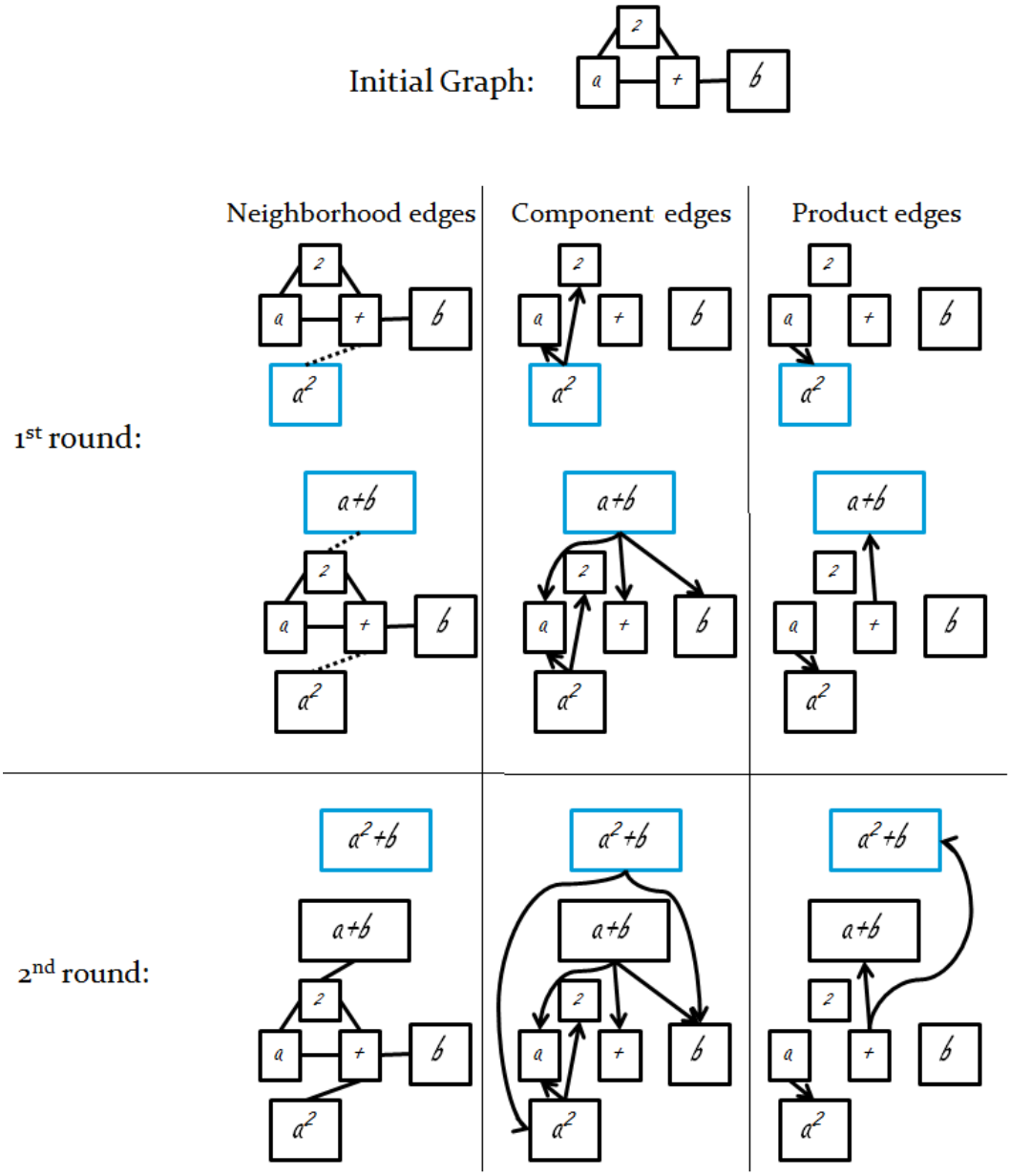

Figure 2.12: States of the graph when parsing the expression $a^2 + b$

3 IMPLEMENTATION

The implementation of the prototype system is done with C# in the .Net framework. With the TabletPC SDK, the Visual Studio development environment provides very useful tools for the user interface, data structures and pen interface. Character recognition methods utilize the LibSVM's [24] .Net[1] implementation as a classifier.

The system is designed in a modular way to be able to make future improvements easier. Token class implementation is based on the reference type classes of C# language, so throughout the parse process the same token object is not created more than once; instead references to those objects are used. This causes fewer object creation operations. Grammar rules are coded as objects and they are all derived from a base rule class; thus, rules can be implemented without changing anything in the parser structure. Utility functions such as calculating angles between lines or checking intersections are also carried by the base classes.

3.1 Data Collection Interfaces

There are separate software parts that are written to collect isolated characters and mathematical expressions. For both programs, list-box GUI components are used to show the progress of data collection and to navigate through data items. Collection can be done by any pointing device, Microsoft TabletPC SDK supports this, but it is intended to be used on a tablet PC or with a stylus.

Both the character collector and mathematical expression collector uses a similar XML based file format. Each file has user, date and comment fields. In addition

[1] http://www.matthewajohnson.org/software/svm.html

to those, character collector files has a collection with ink and label fields for each character. For expression collector each data item also has segments and relations fields. Segments fields carries bounding boxes and labels of the characters of the expression, relations fields are used to record the relations between characters to later extract statistics from. Stroke information is stored in Microsoft's Ink Serialized Format (ISF), Every parameter coming from the tablet can be saved in ISF format along with application specific information.

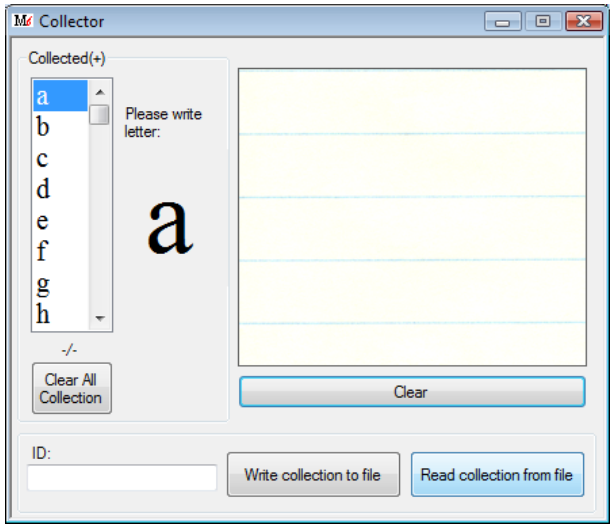

Figure 3.1: Character collection interface

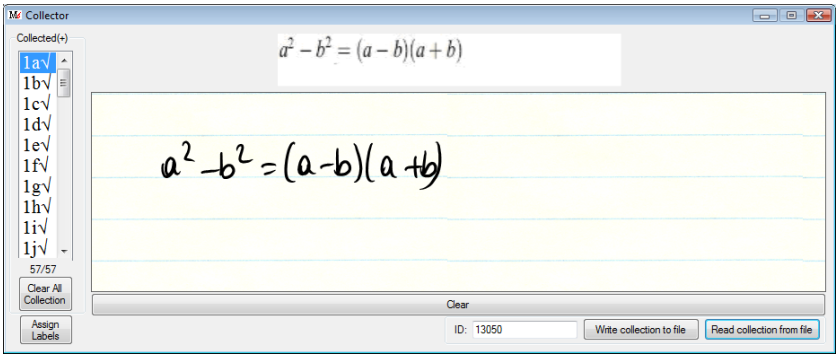

Figure 3.2: Expression collection interface

3.2 Mathlet Interface

The interface has basic capabilities: writing and deleting strokes. Outputs of the interface are LaTeX code and MathML[2] code of the expression and the rendering of the expression. Rendering is done by an open source MathML rendering tool named gNumerator[3].

All generated tokens are shown in a tree list box with theirLaTeX codes. Selecting them updates the MathML code and the rendering with the code of the selected token. Since the control is a tree list component tokens are also listed under each token.

Interface also supports some error correction for OCR. Right clicking a written symbol shows a list of alternatives for the recognition of the symbol. Those alternatives are generated by the ANN. Also, if real time OCR is selected user can see the recognition result as he writes the symbols.

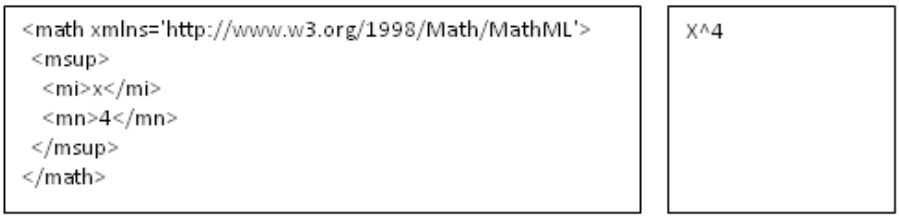

Figure 3.3: MathML and LaTeX codes for expressions x^4

[2]Mathematical Markup Language (MathML) is an application of XML for describing mathematical notations and capturing both its structure and content. It aims at integrating mathematical formulas into World Wide Web documents. It is a recommendation of the W3C math working group.

[3]http://numerator.sourceforge.net

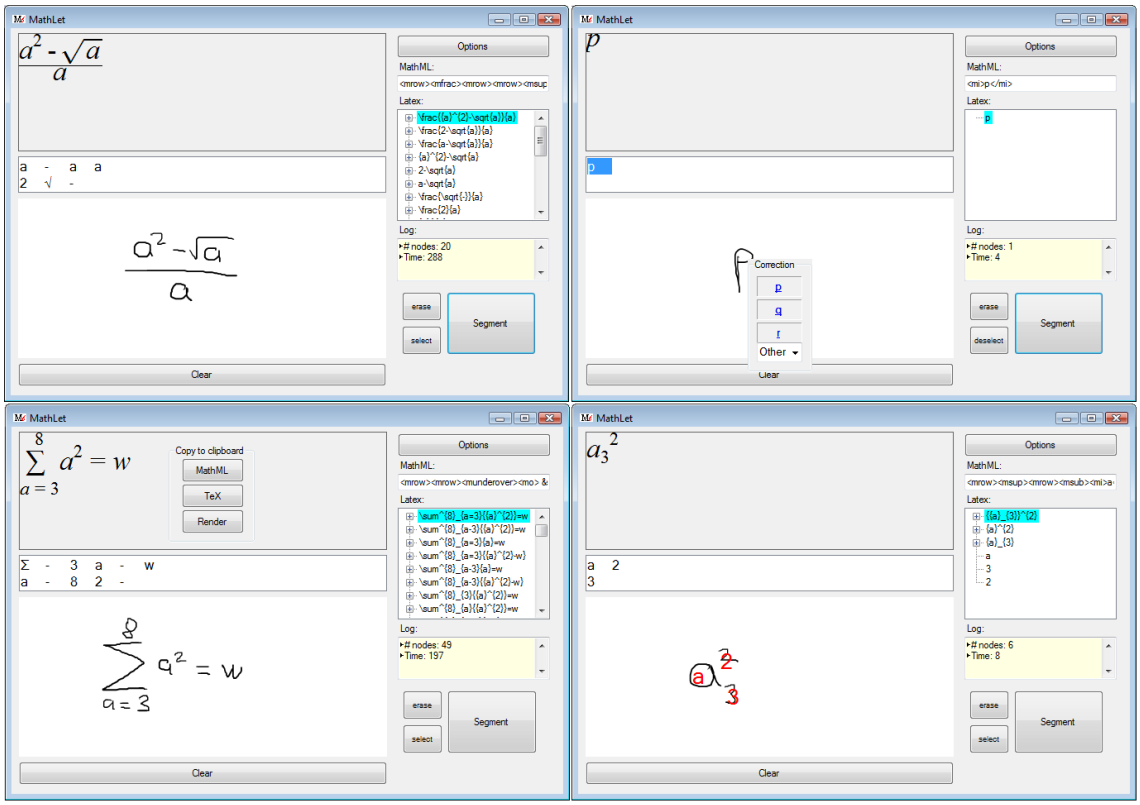

Figure 3.4: Mathlet interface

4 DATA COLLECTION

4.1 Mathematical Expressions

Even for machine printed letters geometrical relations between symbols are varied, this is even more pronounced for hand written expressions. A set of mathematical expressions is collected to be able to generate statistics about the geometrical relations between the symbols of a mathematical expression. There are expressions in this set from different fields of mathematics to be more comprehensive. A list of samples from the collected data is shown in Figures 4.1, 4.2, 4.3 and 4.4.

There are 57 expressions and those expressions are written two times by 15 people. All samples are written in natural hand writing of the users. Base lines are not fixed and there are cursive letters. Most of the expressions are taken from [25] which is a collection of mathematical expressions taken from Standard Mathematical Tables and Formulae[26].

4.2 Isolated Characters

The character set that collected is very similar to the one defined in Kosmala [27]. The set consists of 70 characters, which are lower case letters, digits, the Greek letters and symbols that are shown in Table 4.1. Selected Greek letters and symbols are frequently used in mathematical literature as can be seen in So & Watt [28].

For isolated characters there are two sets of data, one as a training set one for testing. Training set is collected from 15 people with 10 samples from each user. Training set is collected from 6 users and each user has written 4 samples.

α	β	γ	λ	μ	π	ϵ	τ	ϕ
θ	Δ	Π	Σ	∞	$\int$	$=$	$\approx$	$\neq$
$\geq$	$\leq$	$<$	$>$	$\rightarrow$	$\leftrightarrow$	$\{\ \}$	$()$	,
$+$	-	:	$/$	$\surd$				

Table 4.1: Greek letters and symbols included in the character set

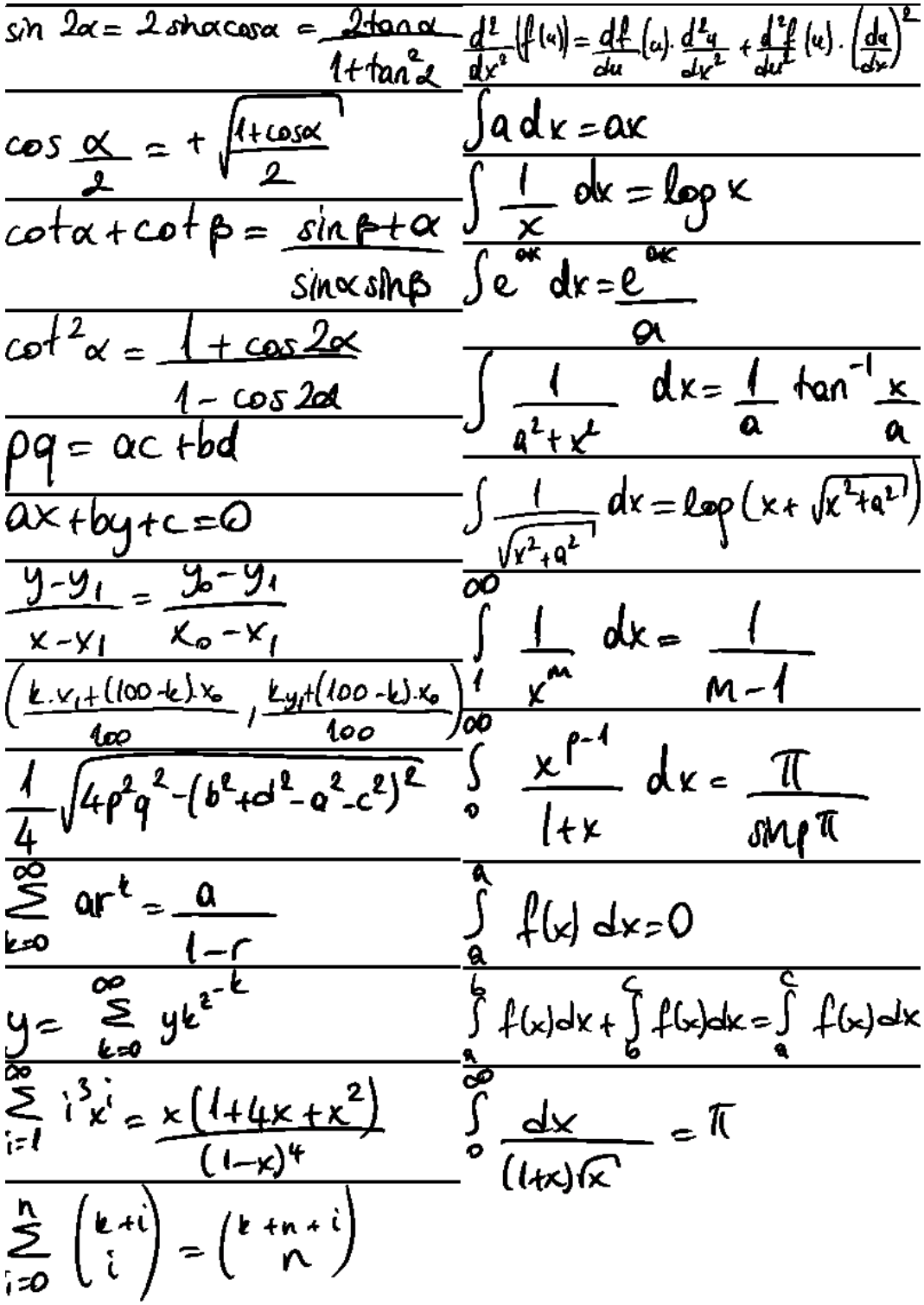

Figure 4.1: Samples from collected expressions (user 1)

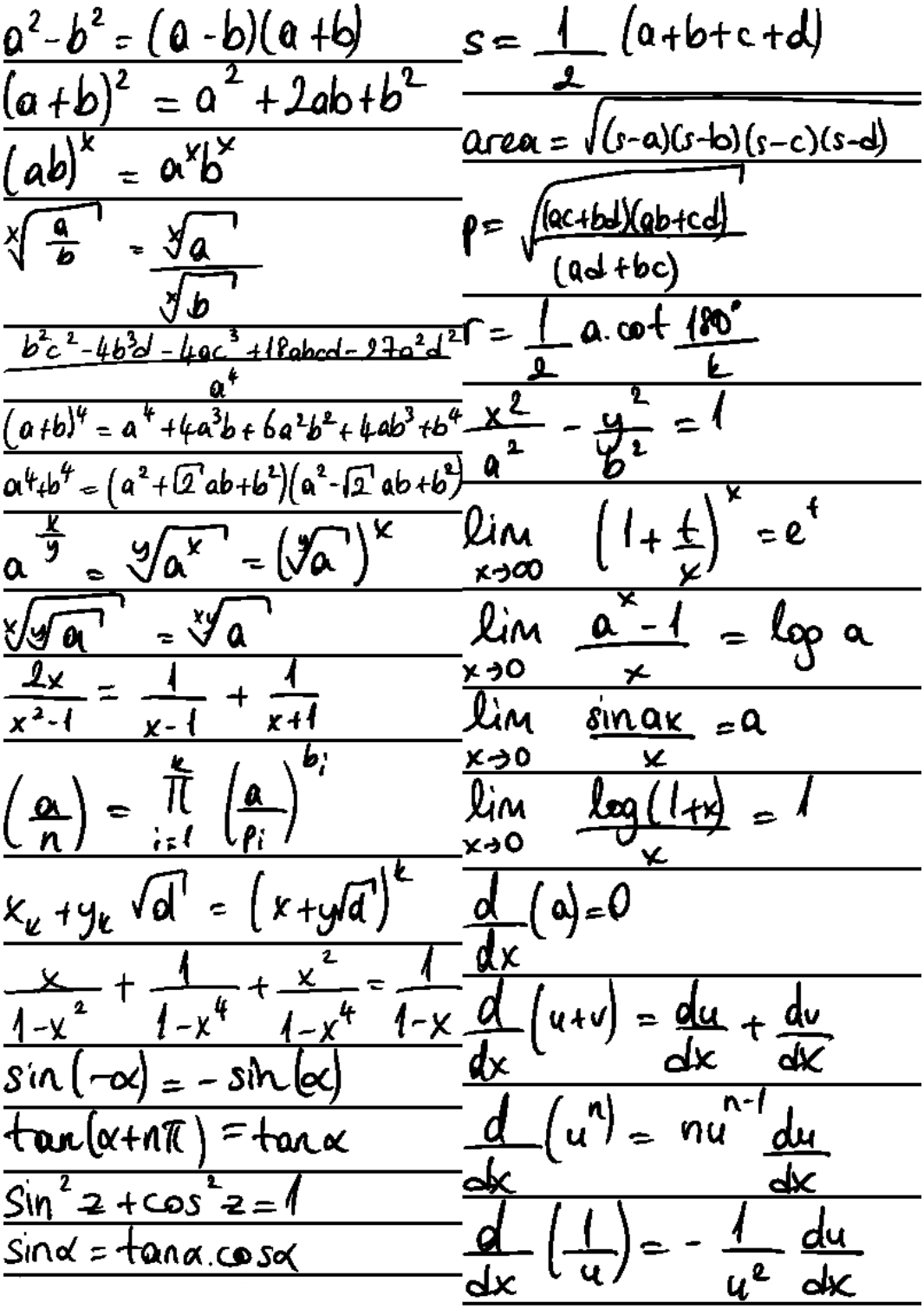

Figure 4.2: Samples from collected expressions (user 1)

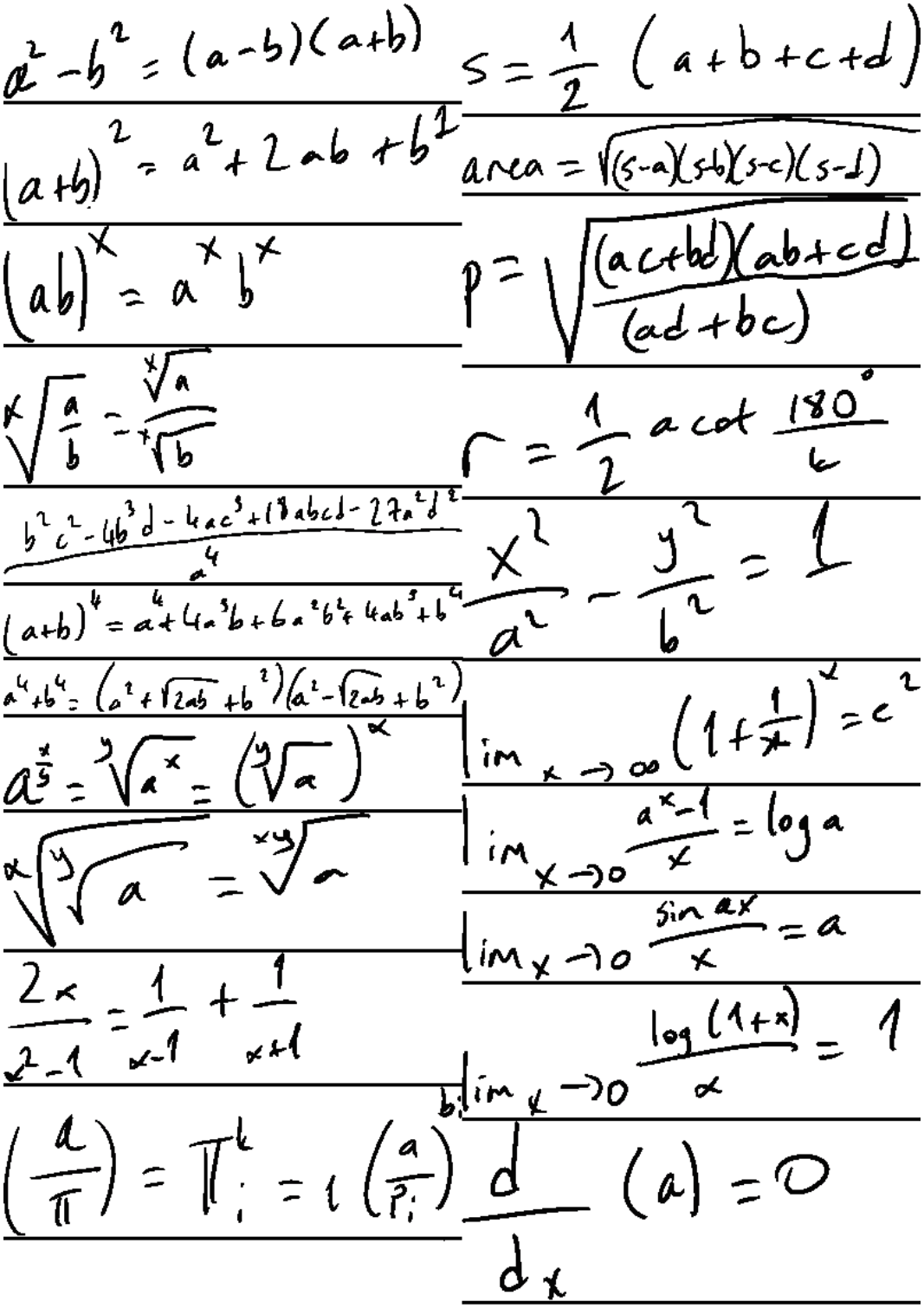

Figure 4.3: Samples from collected expressions (user 2)

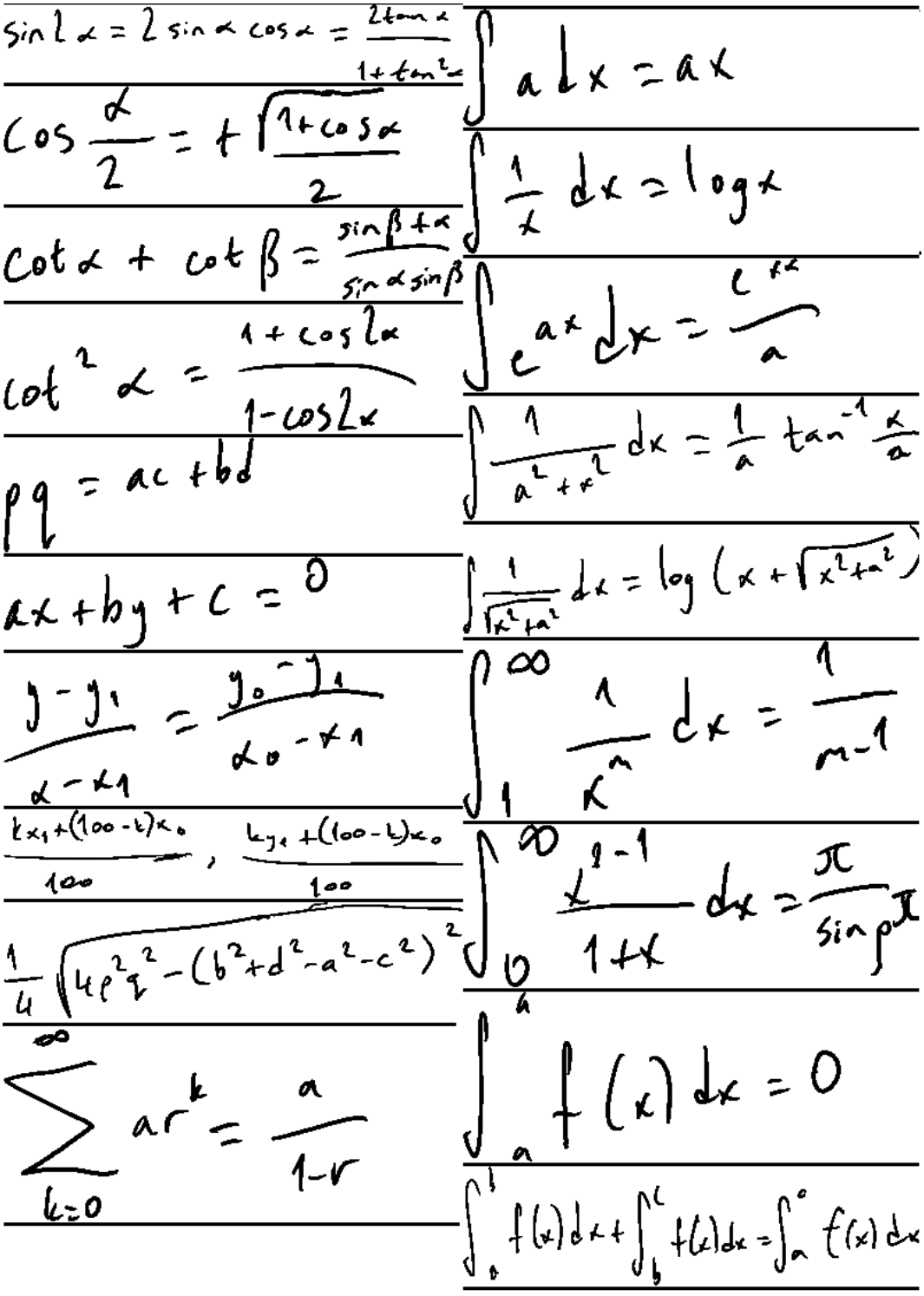

Figure 4.4: Samples from collected expressions (user 2)

5 EVALUATION

5.1 Evaluation

There is no standard way of evaluation for mathematical expression recognition, and evaluations have issues similar to graphic recognition [29]. Most used performance measure is number of expressions correctly recognized from a set of expressions. Those expression sets lack standardization and mostly not publicly available. This mostly makes comparing performance of the systems very difficult. Evaluating user experience is an option but very few system is publicly available and information about coverages, in terms of recognizable symbols and mathematical expression domains, are limited and even not reported [30].

The collected set of mathematical expressions is aimed at extracting statistics from natural handwriting of the users. Thus the set does not meet the input requirements of our recognizer. We selected few shorter expressions from the set to evaluate the system. The limitations of the system mostly comes from the simplistic nature of segmentation and recognition processes, since the focus of the study is the recognition of the expressions. We also tested with manually labeled expressions. Table 5.1 shows the results. Recognition times are ranging from less than a second to 10 minutes. Number of symbols is most influential for the time complexity of the system, also number of neighbors for symbols effects the time spent.

As shown on Table 5.1, our structure recognition rate is %32.5, where OCR errors are present but not effected the parse process to produce a wrong recognition. Effect of OCR results on parse process can be put into two categories, first and most impor-

tant is recognition errors of critical symbols, such as fraction symbol or parathesis. If a recognition error occurs for these symbols, correct structural analysis is not possible. Second effect is on the determination of the baselines of the symbols. Baseline of a symbol will be different if it is recognized as an ascender or a descender symbol, and this information is used in our graph grammar rules, such as rules for subscripts and superscripts.

To observe the behavior of our parse algorithm we also tested with ground truthed expression and our correct recognition rate is %85. This shows that with a state-of-the-art OCR system, success of our mathematical expression recognition system can be greatly improved. Errors are caused by slanted expressions and user variations for geometrical relations.

Correct recognition:	1/40	%2.5
Correct structure recognition:	13/40	%32.5
Segmentation error:	16/40	%40
OCR error in expression:	40/40	%100
Correct structure recognition on manually labeled input:	34/40	**%85**

Table 5.1: Results from 40 expressions from 10 users

Table 5.2 shows the reported results from the groups we mentioned in the literature review. For those results, expression sets used in the evaluations are not available. Evaluation strategies are also different among research groups. For example, MathPad systems is tested after users are given a list of sample symbols that can be recognized by the system, to be informed about the correct way to write the symbols (limitation of the recognizer) and given some time to get used to the interface and learn the capabilities of the system by trying out some simpler expressions[4]. Differences in the interface also effects the evaluation. Interface of the Infty systems gives the recognition result after each stroke is written, thus user gates feedback as he writes the expressions[8]. Correct structure recognition results given in Table 5.1

source	success rate	explanation	# diff. expr.	# samples	# user
Zanibbi et. al. [2]	% 27	Correct recognition rate	73	1	-
LaViola et. al.[4]	%90.8	Correct parsing decision rate	36	1	11
Labahn et. al. [6]	-	interview with users	-	-	-
Tapia et. al. [7]	-	-	-	-	-
Suzuki et. al. [8]	%99	Correct structure recognition	4	-	24
Garain et. al. [11]	% 74.92	Correct recognition rate	-	5500	-
Yeung et. al. [12]	% 97.99	Correct structure recognition	15	4	10
Prusa et. al. [14]	% 97	-	-	-	-

Table 5.2: Reported success rates from the literature

of our system is most comparable with those results. But it should be noted that our test set is limited because our grammar is simple compared to our collected data.

Expression samples shown in Figure 5.1 are correctly recognized expressions by the Mathlet interface. There is also error correction capabilities which further increase the usability of the system. Current implementation supports super-script and sub-scripts, fractions, summations, integrals and square root expressions.

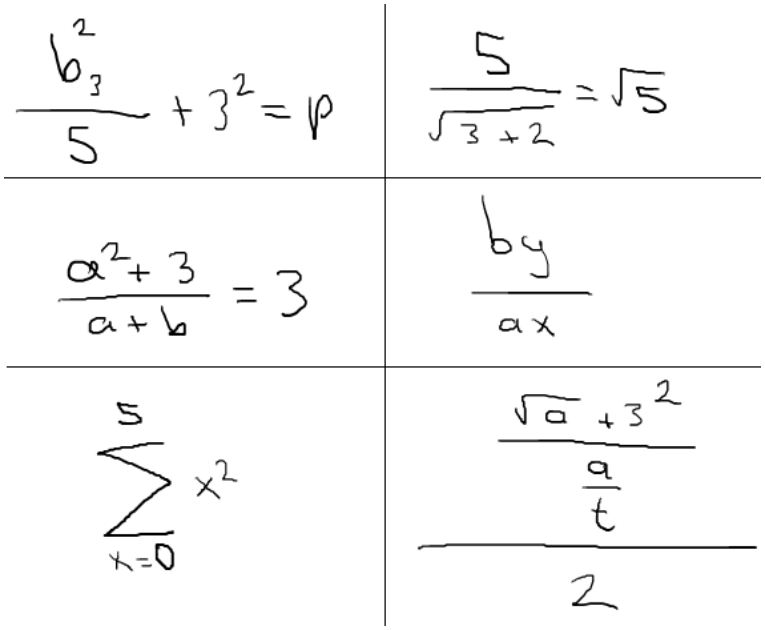

Figure 5.1: Correctly recognized expressions with Mathlet interface

5.2 Future Work

A comprehensive evaluation of the system by direct user interaction is left as a future work. An evaluation strategy similar to LaViola [4] can be followed, where users are given a list of recognizable characters as they should be written and given some time to get use to the system, then they are instructed to write down a list of expressions. Also there can be improvements on OCR system, tools can be implemented to help graph grammar design, improvements can be made on efficiency of the implementation and

system can be integrated with other software.

Improvements can be made on character recognition processes to reduce the character recognition errors. Different feature sets can be tested and online features can be added to the recognition process. Our implementation gives opportunity to separate grammar rule codes and the parser itself. This way grammar rules can be easily tuned to increase the success of parse process.

Graph grammars are inherently hard to design and debug, since they are hard to visualize, due to their multi-dimensional nature. This can be another avenue of future work, developing an interface to design and visualize graph grammar rules. Also, with a graph grammar visualization interface rules can be saved and loaded to programs as external files and this gives the users ability to manipulate the grammar rules.

Most time consuming part of our technique is determining neighborhoods between tokens when initializing the graph and later in the parse process. To increase the speed, a modified kd-tree data structure can be used to be able to utilize sweeping line algorithm which is more time efficient to check intersection between lines.

Furthermore, current interface of Mathlet can be turned into a Microsoft Office add-on. Pen interface can be used to enter mathematical expressions in existing Office applications. New hardware and computer designs gives opportunity to use standard keyboard, touch and pen interfaces at the same time, such as HP TouchSmart computers or laptops with touch screens.

A COMPLEXITY ANALYSIS

Complexity analysis for graph parsing algorithms is out of the scope of this study. However we can show the number of possible rule applications in a simplified graph grammar system. Which certainly does not give a tight bound but gives an idea of the general complexity of the problem. If we assume every input symbol is neighbors of each other (connected graph) and we assume all grammar rules have two symbols in their body then:

Where $N = \{n_1, n_2, ...\}$ is input symbols and
$R = \{r_1, r_2, ...\}$ is grammar rules
number of possible token generation operations is $\frac{(1+|R|)^{|N|}-1}{|R|}$

$$\{n_i\} \rightarrow \binom{|N|}{1}$$

$$\{n_i, r_x, n_j\} \rightarrow \binom{|N|}{2} |R|$$

$$\{n_i, r_x, n_j, r_y, n_k\} \rightarrow \binom{|N|}{3} |R|^2$$

$$\vdots$$

$$+ \underline{\qquad\qquad}$$

$$\binom{|N|}{1} + \binom{|N|}{2} |R| + \ldots + \binom{|N|}{|N|} |R|^{|N-1|} = \frac{(1+|R|)^{|N|}-1}{|R|}$$

This shows, for a given grammar (number of rules $|R|$ are constant) number of

possible token generation is exponential in the number of input symbols $|N|$. In this analysis we do not calculate the probability of rule matches and we assume all rules can be applied in all parts of the input graph. This shows that graph grammar itself is the deciding factor for the complexity of the parser. Since we can represent string grammars with graph grammars, there is polynomial time complexity graph grammar parser. However, two dimensional nature of mathematical expressions guarantees complexity is higher than polynomial for our grammar.

B GRAMMAR RULES

The grammar rules used in our system is given in this section. Rules are grouped according to their center symbols which is show in the left top corner of each grouping. In addition to the grammar rules, the accepted angle ranges are given for different rules. Those values are based on the Cartesian space used by .Net framework which is rotated in contrast to generally used plane.

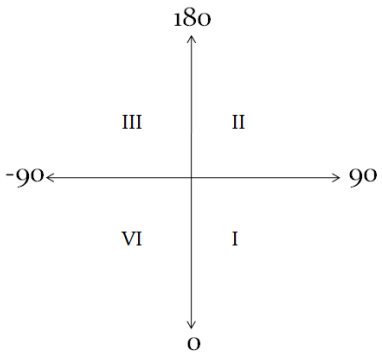

Figure B.1: Cartesian plane given for angle values

List of terminals

op : $*|-|:|.|x|-|/$

eq : $=|<|>|\leq|\geq|\rightarrow$

α : $(a...z)|\alpha|\epsilon|\pi|\gamma|\mu|\tau|\infty$

n : $(0...9)$

f : $sin|cos|tan|cot$

$\mathbf{g_1}$: $\sum|\prod$

$\mathbf{g_2}$: $\int$

$\mathbf{g_3}$: lim

List of rules

op :

$$< \alpha, n, n\alpha, E_t, E_f, E_{fr}, E_{mt}, E_g > \ \mathbf{op} \ < \alpha, n, n\alpha, E_t, E_f, E_{fr}, E_{mt}, E_g, E_{op} > \Rightarrow E_{op}$$

---

eq :

$$< \alpha, n, n\alpha, E_t, E_f, E_{fr}, E_{mt}, E_{op}, E_g > \ \mathbf{eq} \ < \alpha, n, n\alpha, E_t, E_f, E_{fr}, E_{mt}, E_{op}, E_g, E_{eq} > \Rightarrow E_{op}$$

α :

$$\left.\begin{array}{c} \alpha^{<\alpha,n,n\alpha,E_t,E_f,E_{fr},E_{mt},E_{op}>} \\ \\ \alpha_{<\alpha,n,n\alpha>} \end{array}\right\} \Rightarrow E_t$$

$$\alpha \ < \alpha, n, n\alpha, E_t, E_f, E_{fr}, E_{mt}, E_{op} > \Rightarrow n\alpha$$

---

n :

$$\mathbf{n}^{<\alpha,n,n\alpha,E_t,E_f,E_{fr},E_{mt},E_{op}>} \Rightarrow E_t$$

---

f :

$$\mathbf{f}^{<\alpha,n,n\alpha>} < \alpha, n, n\alpha, E_t, E_f, E_{fr}, E_{mt}, E_{op} > \Rightarrow E_f$$

$\mathbf{g_1}$:

$$\begin{array}{c} <\alpha,n,n\alpha,E_t,E_f,E_{fr},E_{mt},E_{op}> \\ \mathbf{g_1} \\ <\alpha,n,n\alpha,E_t,E_f,E_{fr},E_{mt},E_{op}> \end{array} < \alpha, n, n\alpha, E_t, E_f, E_{fr}, E_{mt}, E_{op}, E_g > \ \Rightarrow \ E_g$$

$\mathbf{g_2}$:

$$\left.\begin{array}{c} \begin{array}{c} <\alpha,n,n\alpha,E_t,E_f,E_{fr},E_{mt},E_{op}> \\ \mathbf{g_2} \\ <\alpha,n,n\alpha,E_t,E_f,E_{fr},E_{mt},E_{op}> \end{array} < \alpha, n, n\alpha, E_t, E_f, E_{fr}, E_{mt}, E_{op}, E_g > \\ \\ \mathbf{g_2} < \alpha, n, n\alpha, E_t, E_f, E_{fr}, E_{mt}, E_{op}, E_g > \end{array}\right\} \Rightarrow E_g$$

$\mathbf{g_3}$:

$$\begin{array}{c} \mathbf{g_3} \\ <E_{eq}> \end{array} < \alpha, n, n\alpha, E_t, E_f, E_{fr}, E_{mt}, E_{op}, E_g > \ \Rightarrow \ E_g$$

$\mathbf{E_t}$:

$$\mathbf{E_t} < E_t, E_{mt} > \Rightarrow E_{mt}$$

$$\mathbf{E_t}^{<\alpha,n,n\alpha,E_t,E_f,E_{fr},E_{mt},E_g,E_{op}>} \Rightarrow E_t$$

$\mathbf{E_f}$:

$$\mathbf{E_f} < E_f, E_{mt} > \Rightarrow E_{mt}$$

$\mathbf{E_{fr}}$:

$$\begin{matrix} <\alpha,n,n\alpha,E_t,E_f,E_{fr},E_{mt},E_{op},E_g> \\ \mathbf{fr} \\ <\alpha,n,n\alpha,E_t,E_f,E_{fr},E_{mt},E_{op},E_g> \end{matrix} \Rightarrow \quad E_f r$$

$\mathbf{E_t}, \mathbf{E_f}, \mathbf{E_{fr}}, \mathbf{E_{mt}}, \mathbf{E_g}, \mathbf{E_{op}}, \mathbf{E_g}$:

$$< (> \mathbf{E} <) > \Rightarrow E_t$$

Figure B.2: Example for superscript relation

Following is the list of constraints used as applicability predicates in our graph grammar. Angle values should be viewed with the given Cartesian plane.

- Superscript: slope of the line segment between two token centers is 90 to 170 degrees.
- Subscript: slope of the line segment between two token centers is 10 to 60 degrees.
- Left-Right relation (e.g. +): slope of the line segment between the center token and the token on the right is 60 to 130 degrees and with the one on the left is -60 to -130 on other.
- Above-Below relation (e.g. fraction): slopes of the line segments are -40 to 0 and 0 to 40 degrees for the one below, -140 to -180 and 140 to 180 for the one above of the center token.
- Same baseline relation: slope of the line segment between two token centers is 70 to 110 degrees.

- Summation symbols relations: slopes of the line segments are -40 to 0 and 0 to 40 degrees for the one below, -140 to -180 and 140 to 180 for the one above and 70 to 110 degrees for the one on right of the center token.

- Left-Right relation (for parenthesis): slope of the line segment between the center token and the token on the right is 60 to 120 degrees and with the one on the left is -60 to -120 on other.

- For square root relation applicability predicate do not use angle values. Union of the two bounding boxes have to be smaller than a given threshold. Threshold is a rectangle with the width and height values %50 bigger than the bounding box of the square root symbol.

Bibliography

[1] D. Blostein, J. R. Cordy, and R. Zanibbi, "Applying compiler techniques to diagram recognition," in *ICPR*, 2002, pp. III: 123–126.

[2] R. Zanibbi, D. Blostein, and J. R. Cordy, "Recognizing mathematical expressions using tree transformation," *IEEE Trans. Pattern Anal. Mach. Intell*, vol. 24, no. 11, pp. 1455–1467, 2002.

[3] J. J. L. Jr., "An initial evaluation of a pen-based tool for creating dynamic mathematical illustrations," in *Sketch Based Interfaces and Modeling*, T. Stahovich, M. C. Sousa, and J. A. P. Jorge, Eds. Vienna, Austria: Eurographics Association, 2006, pp. 157–164.

[4] J. LaViolaJr., "Mathematical sketching: A new approach to creating and exploring dynamic," Ph.D. dissertation, Brown University, Providence, RI, USA, 2005.

[5] C. J. Li, R. Zeleznik, T. Miller, and J. J. LaViola, "Online recognition of handwritten mathematical expressions with support for matrices," in *ICPR*, 2008, pp. 1–4.

[6] G. Labahn, S. MacLean, M. Mirette, I. Rutherford, and D. Tausky, "Mathbrush: An experimental pen-based math system," in *Challenges in Symbolic Computation Software*, ser. Dagstuhl Seminar Proceedings, W. Decker, M. Dewar, E. Kaltofen, and S. Watt, Eds., no. 06271. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.

[7] E. Tapia and R. Rojas, "Recognition of on-line handwritten mathematical expressions using a minimum spanning tree construction and symbol dominance,"

in *GREC*, ser. Lecture Notes in Computer Science, J. Lladós and Y.-B. Kwon, Eds., vol. 3088. Springer, 2003, pp. 329–340.

[8] Y. Eto and M. Suzuki, "Mathematical formula recognition using virtual link network," in *ICDAR*, 2001, pp. 762–767.

[9] S. Toyota, S. Uchida, and M. Suzuki, "Structural analysis of mathematical formulae with verification based on formula description grammar," in *DAS*, 2006, pp. 153–163.

[10] A. Raja, M. Rayner, A. P. Sexton, and V. Sorge, "Towards a parser for mathematical formula recognition," in *MKM*, ser. Lecture Notes in Computer Science, J. M. Borwein and W. M. Farmer, Eds., vol. 4108. Springer, 2006, pp. 139–151.

[11] U. Garain and B. B. Chaudhuri, "Recognition of online handwritten mathematical expressions," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 34, no. 6, pp. 2366–2376, 2004.

[12] K. F. Chan and D. Y. Yeung, "An efficient syntactic approach to structural analysis of on-line handwritten mathematical expressions," *Pattern Recognition*, vol. 33, no. 3, pp. 375–384, Mar. 2000.

[13] Y. Shi and F. K. Soong, "A symbol graph based handwritten math expression recognition," in *ICPR*, 2008, pp. 1–4.

[14] D. Prusa and V. Hlavac, "Structural construction for on-line mathematical formulae recognition," in *Iberoamerican Congress on Pattern Recognition*, 2008, pp. 317–324.

[15] B. Q. Vuong, S. C. Hui, and Y. L. He, "Progressive structural analysis for dynamic recognition of on-line handwritten mathematical expressions," *Pattern Recognition Letters*, vol. 29, no. 5, pp. 647–655, Apr. 2008.

[16] K. F. Chan and D. Y. Yeung, "Elastic structural matching for on-line handwritten alphanumeric character recognition," in *ICPR*, 1998, pp. Vol II: 1508–1511.

[17] H. Buyukbayrak, "Online Handwritten Mathematical Expression Recognition," Master's thesis, Sabanc University, Turkey, 2005.

[18] B. Yanıkoğlu, "Segmentation and recognition of offline cursive handwriting," Ph.D. dissertation, Dartmouth College, Department of Computer Science, USA, 1993.

[19] J. J. L. Jr., "Mathematical sketching: A new approach to creating and exploring dynamic illustrations," Ph.D. dissertation, Brown University, Department of Computer Science, 2005.

[20] M. Gönen, A. G. Tanugur, and E. Alpaydin, "Multiclass posterior probability support vector machines," *IEEE Transactions on Neural Networks*, vol. 19, no. 1, pp. 130–139, 2008.

[21] A. Rosenfeld and J. L. Pfaltz, "Web grammars," in *IJCAI*, 1969, pp. 609–619.

[22] H.-J. Schneider, "Chomsky systems for partial orderings, in German," Friedrich-Alexander-Universität Erlangen-Nürnberg, Tech. Rep. 3, 1970.

[23] H. Fahmy and D. Blostein, "A survey of graph grammars: theory and applications," in *ICPR*, 1992, pp. II:294–298.

[24] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001, software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[25] B.-Q. Vuong, S. C. Hui, and Y. He, "Erratum to "progressive structural analysis for dynamic recognition of on-line handwritten mathematical expressions"

[pattern recognition letters 29 (5) (2008) 647-655]," *Pattern Recognition Letters*, vol. 29, no. 9, p. 1454, 2008.

[26] D. Zwillinger and B. Kellogg, "CRC standard mathematical tables and formulae," *SIAM Review*, vol. 38, no. 4, pp. 691–??, Dec. 1996.

[27] A. Kosmala, G. Rigoll, S. Lavirotte, and L. Pottier, "On-line handwritten formula recognition using hidden markov models and context dependent graph grammars," in *In Proceedings of the Fourteenth International Conference on Pattern Recognition*, 1999, pp. 1306–1308.

[28] C. M. So and S. M. Watt, "Determining empirical characteristics of mathematical expression use," in *MKM*, 2005, pp. 361–375.

[29] I. T. Phillips and A. K. Chhabra, "Empirical performance evaluation of graphics recognition systems," *IEEE Trans. Pattern Anal. Mach. Intell*, vol. 21, no. 9, pp. 849–870, 1999.

[30] A. Lapointe and D. Blostein, "Issues in performance evaluation: A case study of math recognition," in *ICDAR*, 2009, pp. 1355–1359.