

MATHLET V3:
RECOGNIZING HANDWRITTEN MATHEMATICAL EXPRESSIONS

by
UTKU ÜLKÜ

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

Sabanci University
Spring 2013

MATHLET V3:
RECOGNIZING HANDWRITTEN MATHEMATICAL
EXPRESSIONS

APPROVED BY:

Assoc. Prof. Berrin Yanıkođlu
(Thesis Supervisor)

Assoc. Prof. Yücel Saygın

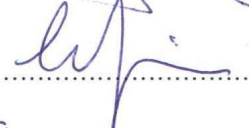
Assoc. Prof. Cem Güneri


Assoc. Prof. Selim Balcısoy

Prof. Muhammet Köksal


.....


.....


.....


.....


.....

DATE OF APPROVAL: 13/08/2013

© Utku Ülkü 2013
All Rights Reserved

MATHLET V3:
RECOGNIZING HANDWRITTEN MATHEMATICAL
EXPRESSIONS

Utku Ülkü

Computer Science and Engineering, M.Sc. Thesis, 2013

Thesis Supervisor: Berrin Yanıkoğlu

Keywords: handwriting, recognition, online, mathematical, expression

Abstract

This thesis presents MathLet v3 which is the third version of a system developed to recognize handwritten mathematical expressions. Previous versions were developed by Hakan Büyükbayrak and Mehmet Çelik.

MathLet v3 implements two steps to recognize handwritten mathematical expressions; symbol recognition and parsing. In the symbol recognition step, two classifiers are combined. One of these classifiers uses online features while the other one uses offline features. Both classifiers return probability distributions over classes.

In the parsing step, probability distributions are used to increase time performance of MathLet v3. Moreover, parallel programming is used in parsing phase. Special handling approach for mistaken symbols is also implemented in the parsing step.

MathLet v3 has four applications and two of them can be accessed through the Web. Users write mathematical expressions or upload existing InkML files which contain mathematical expression and get recognition results for them through the Web by using these applications.

MathLet has been participating in a competition named CROHME since 2011. The evaluation results of MathLet in CROHME show that the accuracy of MathLet has increased from 0.55% to 8.35% starting from 2011, although recognition task becomes more difficult each year. In addition to accuracy improvements, experiments made in order to measure the time performance of MathLet v3 show that MathLet v3 has become faster.

MATHLET V3: ELLE YAZILMIŞ MATEMATİKSEL İFADELERİ TANIMA

Utku Ülkü

Bilgisayar Bilimi ve Mühendisliği, Yüksek Lisans Tezi, 2013

Tez Danışmanı: Berrin Yanıkoğlu

Anahtar Kelimeler: el yazısı, tanıma, çevrimiçi, matematiksel, ifade

Özet

Bu tez elle yazılmış matematiksel ifadeleri tanımak için geliştirilmiş bir sistemin üçüncü versiyonu olan MathLet v3'ü sunar. Önceki versiyonlar Hakan Büyükbayrak ve Mehmet Çelik tarafından geliştirilmiştir.

MathLet v3 elle yazılmış matematiksel ifadeleri tanımak için iki aşama uygular; sembol tanıma ve çözümlenme. Sembol tanıma aşamasında iki sınıflandırıcı birleştirilir. Bu sınıflandırıcılardan biri çevrimiçi özellikleri kullanırken diğeri çevrimdışı özellikleri kullanır. Her iki sınıflandırıcı da sınıflar üzerindeki olasılık dağılımını verir.

Çözümlenme aşamasında, MathLet v3'ün zaman performansını artırmak için olasılık dağılımları kullanılır. Ayrıca paralel programlama da çözümlenme safhasında kullanılır. Çözümlenme aşamasında, yanılığa düşülen karakterler için özel işleme yaklaşımı uygulanır.

MathLet v3 dört uygulamaya sahiptir ve bunlardan ikisine Web üzerinden ulaşılabilir. Kullanıcılar bu uygulamaları kullanarak Web üzerinden matematiksel ifadeler yazar ya da matematiksel ifade içeren InkML dosyalarını yükler ve bunlar için tanıma sonuçları elde eder.

MathLet 2011'den beri CROHME adlı bir yarışmaya katılmaktadır. MathLet'in CROHME'daki değerlendirme sonuçları, tanıma görevinin her yıl daha zor hale gelmesine karşın, MathLet'in doğruluğunun 2011'den başlayarak %0.55'ten %8.35'e yükseldiğini gösterir. Doğruluk geliştirmelerine ek olarak, MathLet v3'ün zaman performansını ölçmek amacıyla yapılan deneyler MathLet v3'ün daha hızlı hale geldiğini gösterir.

ACKNOWLEDGEMENTS

I wish to express my gratitude to,

Assoc. Prof. Berrin Yanıkođlu, for her understanding, academic support and contributions to my personality development,

TÜBİTAK (The Scientific and Technical Research Council of Turkey), for the M.Sc. fellowship supports,

Hakan Büyükbayrak and Mehmet Çelik, for their previous work,

Çağlar Tırkaz, for his resampling code,

Assoc. Prof. Yücel Saygın, Assoc. Prof. Cem Güneri, Assoc. Prof. Selim Balcısoy and Prof. Muhammet Köksal, for their participation in my thesis jury,

Sevcan Alcı, for her heavenly love,

last, but not the least, to my family, for being there when I needed them to be.

TABLE OF CONTENTS

1	Introduction	1
2	Previous Work	4
2.1	MathLet	6
2.1.1	MathLet v1	6
2.1.2	MathLet v2	9
3	MathLet v3	15
3.1	Symbol Recognition	15
3.1.1	Offline Classifier	16
3.1.2	Online Classifier	17
3.1.3	Classifier Combination	18
3.2	Parsing in MathLet v3	25
3.2.1	Tokens	26
3.2.2	Grammar Rules	28
3.2.3	Rule Application	32
3.2.4	Sorting Existing Tokens	34
3.2.5	Parsing MathML Codes	34
3.3	Accessibility	36
4	Accuracy and Time Performance Evaluations	41
4.1	Accuracy Evaluation	41
4.2	Time Performance Evaluations	42
4.2.1	Evaluation Metrics	42
4.2.2	Initial Time Performance Evaluation Results of MathLet v3	43
4.2.3	Time Performance Evaluations of MathLet v3 After Improve- ments	44
4.2.4	Comparison of Measurement Results	45

5	CROHME Competition	46
5.1	Data Format	47
5.2	Task and Evaluation Metrics	50
5.3	Evaluation Results of MathLet	52
5.4	CROHME Evaluation Results	54
6	Conclusions	56
7	References	59

LIST OF FIGURES

1.1	The symbol “5” written in one stroke and two strokes	2
2.1	Single stroke equivalents of the symbols “=” and “+” used by Math-	
	Let v1	7
2.2	Data collection interface of MathLet v1	7
2.3	ME recognition interface of MathLet v1	8
2.4	The article structure recognition interface of MathLet v1	9
2.5	Symbol recognizer in MathLet v2	10
2.6	Interface of CharCollector	11
2.7	An example for mistaken symbol handling in MathLet v2	12
2.8	Initial token list after recognition of the symbol “ x ” in MathLet v2 .	12
2.9	Handwritten ME “ $2d$ ”	13
2.10	Interface of MathLet v2	14
3.1	An example for symbol classification in MathLet v3	16
3.2	Interface of the tool “View Ink Points”	18
3.3	Classifier combination in the symbol recognition in MathLet v3 . . .	19
3.4	The symbol “ a ” written ambiguously	20
3.5	The symbol “2” written in two different ways	24
3.6	A visual representation of the token “ a^3 ”	26
3.7	An example for mistaken symbol handling in MathLet v3	27
3.8	Initial token list after the recognition of the symbol “ x ” in MathLet v3	28
3.9	Recognition result for the ME “ $1 + 2435$ ”	30
3.10	The ME “ 2^x ” written ambiguously	31
3.11	Recognition result for the ME “ $y + 16 = x$ ”	33
3.12	The tokens generated while the ME “ $y + 16 = x$ ” is being parsed . .	33
3.13	Expression tree and the MathML code for the ME “ $a + c = b$ ” pro-	
	duced by MathLet v3 before MathML parsing	35

3.14	InkML upload Web page of MathLet v3	38
3.15	Example result pages for uploaded InkML files	39
3.16	The web interface of MathLet v3	40
5.1	A view of the ME “ $T \int \Delta dl$ ” included in Part-IV training files	47
5.2	A view of the ME “ $a + c = b$ ” included in a Part-I training InkML file	50
5.3	A view of the ME “ $\int (2^x - 3e^x) dx$ ” included in Part-II training files .	52
5.4	A view of the ME “ $[b^x \{(\frac{a}{b})^x + 1\}]^{\frac{1}{x}}$ ” included in Part-III training files	52

LIST OF TABLES

3.1	The accuracy rates of classifiers in MathLet v3	20
3.2	The accuracy rates of different classifier combinations	20
3.3	Symbol based accuracies of classifiers	24
3.4	Mistaken symbols for the symbols classified by combined classifier with the rate less than or equal to 50%	25
4.1	The accuracy evaluation results of MathLet	41
4.2	The initial time performance evaluation results of MathLet v3	43
4.3	Initial stroke number-processing time relationship in MathLet v3	43
4.4	The improved time performance evaluation results of MathLet v3	44
4.5	Improved stroke number-processing time relationship in MathLet v3	44
5.1	An example of an InkML file for the ME “ $a + c = b$ ”	49
5.2	The content MathML code of the ME “ $a + c = b$ ”	51
5.3	The evaluation results of MathLet v2 in CROHME 2011	53
5.4	The expression-level recognition rates of MathLet v3 in CROHME 2012 with the test dataset of CROHME 2011	53
5.5	The evaluation results of MathLet v3 in CROHME 2012	53
5.6	MathLet v3’s expression recognition rates with errors in CROHME 2012	54
5.7	MathLet v3’s expression recognition rates with errors in CROHME 2013	54
5.8	The evaluation results of CROHME 2011	54
5.9	The evaluation results of CROHME 2012	55
5.10	The evaluation results of CROHME 2013	55

LIST OF ABBREVIATIONS

CROHME Competition on Recognition of Online Handwritten Mathematical Expressions

CYK Cocke-Younger-Kasami

GUI Graphical User Interface

HMM Hidden Markov Model

HTML HyperText Markup Language

InkML Ink Markup Language

MathML Mathematical Markup Language

ME Mathematical Expression

MLP Multi-Layer Perceptron

NN Neural Network

PDF Portable Document Format

SCFG Stochastic Context-Free Grammar

SVM Support Vector Machine

XML Extensible Markup Language

1 Introduction

Recognition of handwritten or printed text has become a very important need since many years. This need has increased with the increasing the popularity of smart phones, electronic pads, electronic tablets, tablet computers and other touch-enabled devices. Handwritten mathematical expression (ME) recognition has also emerged as a remarkable specific need among these needs. Today, individuals who especially study on science documents need to write MEs and digitize them.

Handwritten MEs can be written by individuals on their computers by using mouse, electronic tablets, electronic pads, touch pads, touch-enabled screens etc. The recognition of these handwritten MEs can be achieved by the systems which generally have an interface or a Web page which can be accessed through the Web. Today, there are also mobile applications which can be used for the same purpose through the smart phones, tablet computers and other mobile devices.

The task of the recognition of handwritten ME generally consists of two steps which are character or symbol recognition and structural analysis [1]. The task of symbol recognition step is to recognize individual characters which are included in the handwritten ME. For instance, the task of symbol recognition for the ME " $a_n + b_k$ " is to recognize the symbols " a ", " n ", "+", " b " and " k ".

In structural analysis phase, the main task is to identify the relationships between the symbols which are recognized in the first step. Then, the ME is structured among identified relationships. For example, in the ME " $a^2 + b^3$ ", there are two superscript relationships between the symbols " a " and " 2 ", and the symbols " b " and " 3 ". After the identification of these relationships, the system should also consider the plus sign between " a^2 " and " b^3 " and constitute ME at the end.

After the recognition process is finished, users can obtain the digitized ME and use this information easily. For instance, users can use the L^AT_EX code of ME if they write a thesis, paper or article on a L^AT_EX editor. Mathematical Markup Language (MathML) code of ME is another output format that can also be used by users

depending on their needs.

The task of handwritten ME recognition possesses some ambiguities. First one can be called as the symbol segmentation. Especially the segmentation of symbols which are written in more than one stroke such as “ i ”, “ $!$ ”, “ $+$ ”, “ $=$ ” etc. is difficult. In addition to these symbols which naturally consist of more than one stroke, users may write other symbols in more than one stroke too. For example, some individuals write the symbol “5” in two strokes. They generally write the line which is at the top of the symbol “5” in one stroke and the remaining part in another stroke, while some individuals write this symbol in only one stroke. In Figure 1.1, the symbol “5” in the left is written in one stroke, while the one in the right is written in two strokes. Secondly, there are too many possible relationships between recognized symbols. Superscript and subscript relationships are only two of them. For instance, in order to identify the relationship between the symbols of the ME “ a_n ” is not trivial and also depends on the writing habits of users. A system can recognize the ME as “ an ”, “ a^n ” or “ a_n ”.

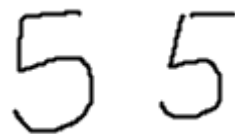


Figure 1.1: The symbol “5” written in one stroke and two strokes

Handwritten ME recognition can be divided into two categories. These categories are online and offline handwritten ME recognition. In online ME recognition, symbols consist of strokes. The number of strokes may be one or more than one. Online ME recognition systems can also use temporal information about input data. Two examples of the systems which implement online handwritten ME recognition can be found in [2] and [3]. These systems are based on academic studies. There is also MyScript Equation recognizer which is the commercial system developed by Vision Objects [4]. MathLet [5] is another example of the systems which implement online ME recognition.

In offline ME recognition, there is no temporal information about input data. The input data is the image of symbols, in other words there is a set of black pixels representing a symbol. There is no certain information about the strokes which a

symbol consists of. One example of these systems is Infty project [6]. A recent system for the recognition of printed MEs is detailed in [7].

With the increasing attention paid to the area of ME recognition, Competition on Recognition of Online Handwritten Mathematical Expressions (CROHME) has been organized since 2011. In order to compare MathLet with other benchmark systems, participation in CROHME was crucial. There was also need to measure and improve the time performance of MathLet. This thesis presents newer version of MathLet, namely MathLet v3, which participated in CROHME and has improved accuracy and time performance. MathLet v3 has also two applications which can be accessed through the Web.

As a contribution of this thesis, MathLet v3 uses both online and offline features in different classifiers and combines these classifiers. In the combination, the classifier which returns greater prediction probability for the most probable symbol that it predicts is chosen. In the parsing phase of MathLet v3, mistaken symbols are specially handled by using prediction probabilities. The functions used in parsing were analyzed and one loop which takes MathLet v3 much time to process was parallelized. This thesis also presents the evaluation of the time performance of MathLet v3. Symbol recognition, parsing step and MathML parsing implemented in MathLet v3 provided an increase in the time performance and expression level recognition rates of MathLet v3. Furthermore, this thesis presents two Web applications of MathLet v3. One of these applications provides that users can upload existing ME included in Ink Markup Language (InkML) file and the other one provides that users can write their own handwritten ME. Users can get top-5 recognition results for the MEs using both applications.

The remainder of the thesis is organized as follows. First, a review of previous work on handwritten ME recognition is given in Section 2. Section 3 presents MathLet v3 which is the system developed to recognize handwritten MEs. Section 4 reports accuracy and time performance of MathLet. Section 5 provides an overview of CROHME and reports MathLet v3's evaluation results obtained in CROHME competitions. In Section 6, contributions and future work are presented.

2 Previous Work

For the purpose of handwritten ME recognition, several approaches are proposed and used by different benchmark systems. In the work described in [2], 2D stochastic context-free grammar (SCFG) is defined. This grammar consists of symbols, grammar rules and probability function. Grammar rules are manually defined in this grammar. Spatial relations are given as a parameter to these grammar rules. These spatial relations are horizontal, vertical, subscript, superscript and inside relations. The system also uses a parser based on Cocke-Younger-Kasami (CYK) based algorithm which is defined for 2D SCFG. The recognition process is started by Hidden Markov Model (HMM) classifier which achieves symbol recognition and segmentation steps. Then, a set of symbol recognition and segmentation hypotheses are obtained. 2D SCFG continually generates the ME from its subexpressions according to these hypotheses and CYK-based parser finds the most probable ME. As a result, HMM-based classifier, 2D SCFG and CYK-based parser jointly achieve the recognition of ME which is given as input. This system participated in CROHME 2011 and took the first place.

A system which implements a baseline extraction-driven parsing of handwritten MEs is detailed in [8]. The system first identifies the strokes of the leftmost symbol on the main baseline by using a data structure called Left Blocking Tree. Detected symbols are classified by using HMM-based classifier. After the leftmost symbol is detected, the system detects the next baseline symbol. In this step, the system finds the conditional probability which shows that whether candidate for next baseline symbol is placed in the area of superscript, subscript or adjacent at right with respect to current symbol. If the conditional probability of adjacency is greater than other two probabilities a candidate is determined as next baseline symbol, otherwise the region of a symbol according to the current symbol such as subscript, above etc. is found. A Left Blocking Tree is created for each new region and each new region represents a new baseline. This continues until all strokes are processed.

Extracted baselines are then parsed by a modified LL(1) parser which makes lexical analysis. Finally, the system ranks the parses by a scoring function which only considers symbol recognition. This scoring function does not consider the spatial relationships.

An online recognizer, MyScript Equation recognizer, which is developed by Vision Objects [4] handles segmentation, recognition and interpretation steps concurrently. The system has three important entities which are equation recognition engine, grammar and symbol expert. The equation recognition engine first determines the segmentation based on the grammar rules each defining a different spatial relationship such as vertical relationship for fraction symbol, nominator and denominator. Then, symbol expert makes probability estimation based on the segmentation. Symbol expert consists of a set of classifiers which use the combination of the features extracted from online and offline information. These classifiers use neural network (NN) and other pattern recognition techniques. The equation recognition engine uses a statistical language model which uses context information extracted from hundreds of thousands of equations. For the purpose of training the recognizer, a global discriminant training scheme on equation level with automatic learning of required parameters is used. This system participated in CROHME 2012 and it was the winner of it.

Waterloo recognizer [3] is a system developed for MathBrush [9]. Three-step recognition process is used by this system. The system first recognizes the symbols and parsing is performed in the second phase. Third and final step, tree extraction, is for the purpose of ranking the ME. In the symbol recognition stage, strokes are grouped by using proximity of strokes and bounding box alignment. Grouped strokes are then recognized by symbol recognizer which uses feature-based matching and elastic matching distance. A fuzzy relational grammar and a tabular variant of Unger's parsing method are used in parsing step in order to produce parse forest. In the grammar, there are relations for subscript, superscript, horizontal and vertical adjacency and containment such as the relation in the ME " \sqrt{x} ". In tree extraction step, each tree in parse forest is extracted by scoring which is made by considering symbol recognition scores and relation membership grades. This system was one of the CROHME 2012 participants and it took the second place.

As mentioned, the recognition of handwritten ME consists of symbol recognition and structural analysis phases. In the symbol recognition step, most of existing systems use traditional classification techniques. There are also some research, [10], [11], [12], [13], which only concentrate on mathematical symbol recognition without mentioning the problem of structural analysis and the recognition of whole ME. In [2] and [10] HMM is used. The system detailed in [2] uses both online and offline features and combines them. In combination, Naive Bayes Classifier and weighting are used. Support Vector Machines (SVMs) are used in [11] with offline features. In [12], SVM is trained by using online and offline features and taking weighted sum. In [14], multi-layer perceptron (MLP) NN is used. Both online and offline data are used by Neural Network model described in [13].

2.1 MathLet

MathLet is the name of the software which is designed for the recognition of handwritten MEs. It has two previous versions. In this thesis, these previous versions of MathLet are called as MathLet v1 [15] and MathLet v2 [5].

2.1.1 MathLet v1

MathLet v1 [15] is developed by Hakan Büyükbayrak as Master’s thesis under the supervision of Aytül Erçil and Berrin Yanıkoğlu. MathLet v1 uses two-phase process for handwritten MEs; symbol recognition and parsing. The system has the capability of recognizing 66 different mathematical symbols.

In the symbol recognition step, a MLP NN with 40 inputs and 66 outputs is utilized. Data used to train symbol recognizer is collected by using an interface developed for collecting ink data. The interface can be seen from Figure 2.2. Collected data is normalized to 20 equidistant points and x and y coordinates of them form 40 inputs of MLP. MathLet v1 assumes that all symbols are written in only one stroke. By this assumption, the symbol recognition of MathLet v1 turns out to be stroke recognition. In contrast, this provides the easy segmentation of symbols especially intersected symbols. For the symbols which naturally consist of more than one stroke such as “=”, “+”, single stroke equivalents of them are suggested. Figure 2.1 shows the symbols “=” and “+” together with their single stroke equivalents

suggested in MathLet v1.

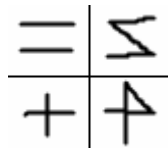


Figure 2.1: Single stroke equivalents of the symbols “=” and “+” used by MathLet v1

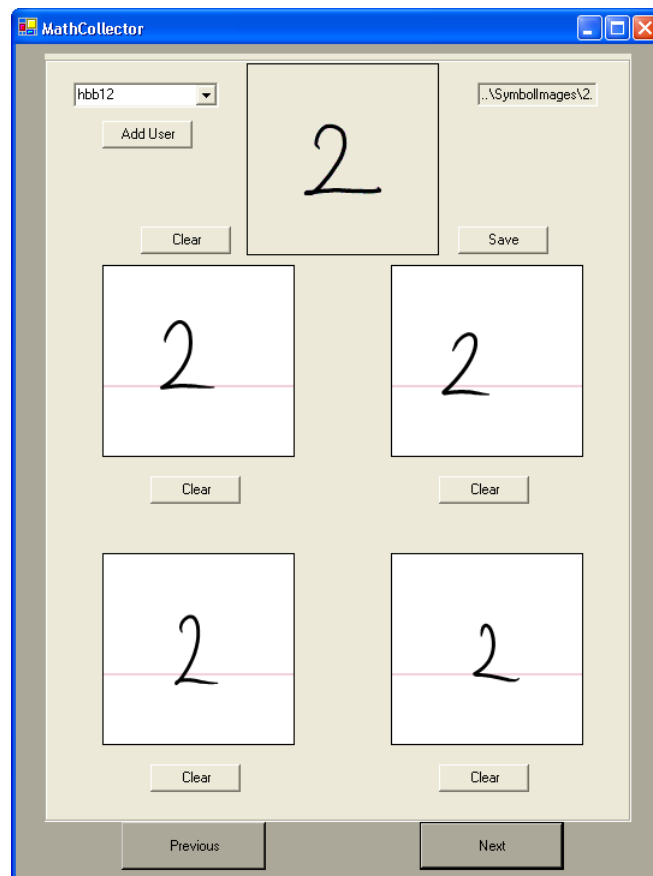


Figure 2.2: Data collection interface of MathLet v1

After symbol recognition, MathLet v1 performs expression parsing step with procedural approach. Fraction, summation, square root, integral, superscript, subscript, logarithm and trigonometric functions are recognized in this step. The system first sorts all symbols from left to right. Expression parsing starts with the leftmost symbol and continues to the right until all symbols are parsed. MathLet v1 uses procedures for each structure in parsing stage and these procedures are applied when a structure is recognized.

Procedures defined consist of simple positioning and size metrics. For instance, when a fraction line is recognized, the system parses upper and lower regions of it. For integral and summation sign, upper, lower and right regions of them are parsed. For subscript and superscript structures, the size of the symbol which is at upper-right (for superscript) and lower-right (for subscript) of base symbol is compared with the size of the base symbol. The size of subscript and superscripts should be smaller than base symbol and their positions should be appropriate. When the structures are combined in a ME, a recursive parsing is performed.

Second interface of MathLet v1 is developed for recognizing MEs. This interface can handle matrices and recursive structures. It can also be used to load and save the ink data. Furthermore, it provides the \LaTeX code of written ME and also can evaluate the result of it. Figure 2.3 shows the sample interface.

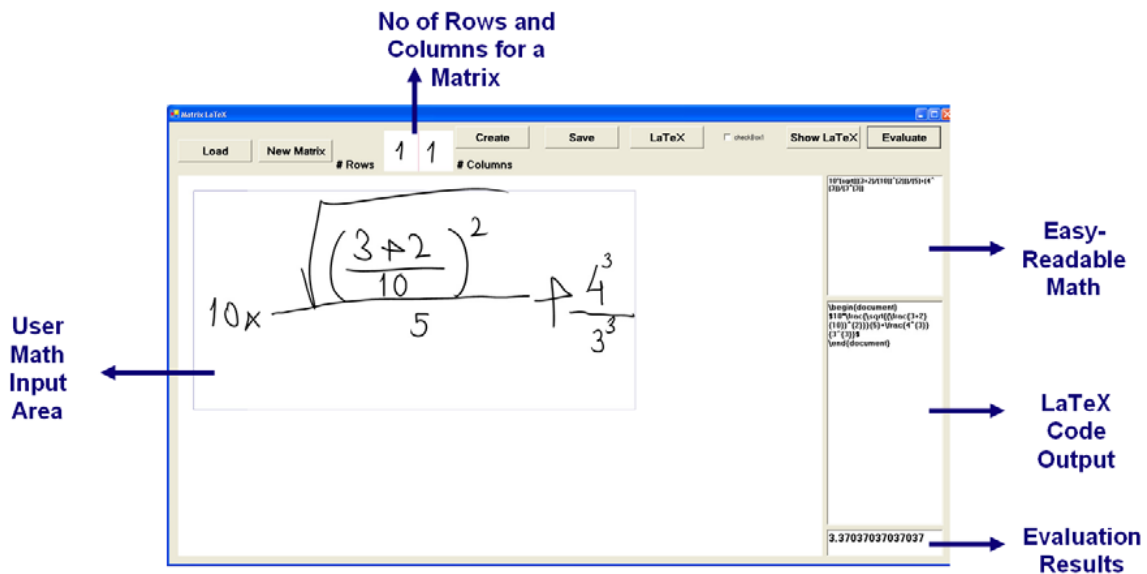


Figure 2.3: ME recognition interface of MathLet v1

It is also possible to recognize articles containing text, MEs and figures in MathLet v1 by using the article structure recognition interface of it. The article structure recognition interface of MathLet v1 is shown in Figure 2.4. This third interface of MathLet v1 provides the segmentation of articles and handling recursive mathematical structures. Moreover, a user can export recognized article in Portable Document Format (PDF) . In the interface, a user should identify the regions of MEs and figures by using different pens.

As a result, the assumption that a symbol is written in one stroke could be ac-

cepted as a weakness of MathLet v1. Also, the system does not have the capability of providing different alternative recognition results. On the other hand, the interfaces of MathLet v1 are the strengths of it. Interfaces provide easy collection of data, writing MEs and recognizing articles containing not only MEs but also text and figures.

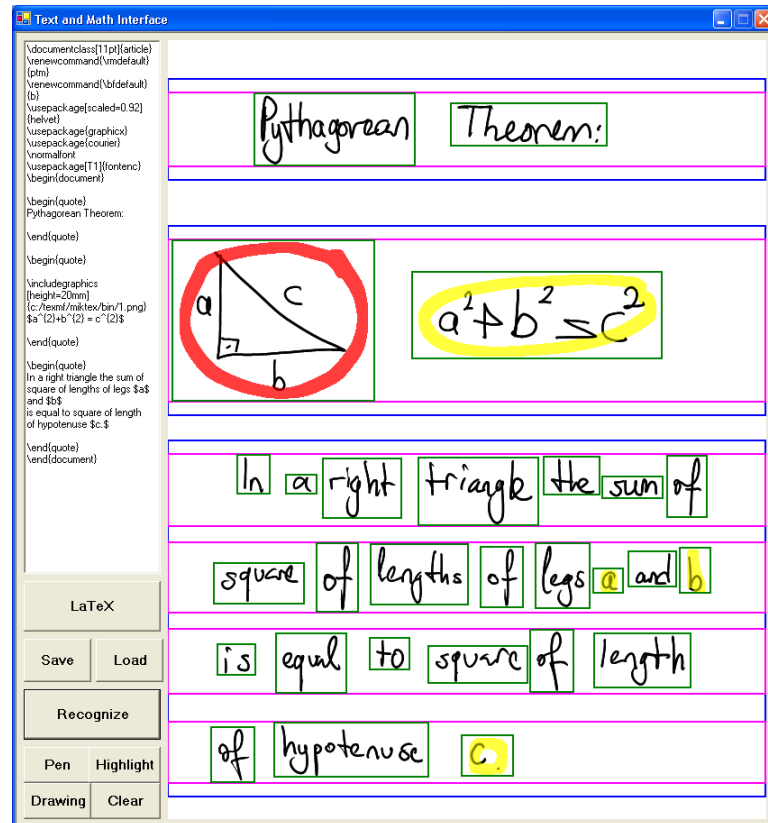


Figure 2.4: The article structure recognition interface of MathLet v1

2.1.2 MathLet v2

MathLet v2 [5] is developed by Mehmet Çelik as a Master's thesis under the supervision of Berrin Yanıkoğlu. It follows the traditional approach which consists of two steps to recognize handwritten MEs. The system first recognizes individual symbols and then the whole ME is recognized by structural analysis. In the structural analysis step, 2D-grammar is used and the system gives more than one result sorted by their statistically calculated likelihood values.

Symbol Recognizer

In the symbol recognition step, MathLet v2 uses a classifier based on SVMs. This classifier is obtained by running a program named CharTrainer. In MathLet v2 [5], symbol or character recognizer is able to return only the label of a predicted class for an input symbol. It does not return the prediction probability of the class. It also does not return any other possible class. Figure 2.5 shows the illustration of this process. SVM kernel of the symbol recognizer is Radial Basis Function. MathLet v2's symbol recognizer is trained by using 288 offline features extracted from the images of symbols. The data collected from students is used for the training of symbol recognizer.

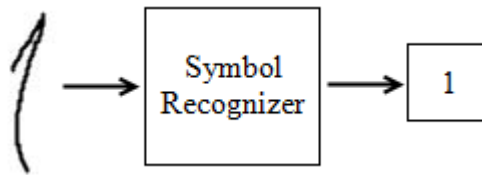


Figure 2.5: Symbol recognizer in MathLet v2

Training data is collected by using a program named CharCollector which is developed in Microsoft .NET environment and C# programming language. The interface of CharCollector can be seen in Figure 2.6. CharCollector produces XML (Extensible Markup Language) files as a training data. These XML files contain the information about the training data. These information are the label of the symbol and x and y coordinates of points which form a symbol included in the training data. CharCollector can generate these two information in two ways. In the first one, a user writes the symbol and it collects the data. In the second one, it takes an InkML file as an input and extract the data from it.

Recognizer is one of the important entities of MathLet v2. It specifies the list of symbols which the system can recognize. Furthermore, it is used to load the classifier and get the results from it. Moreover, symbol recognition results are organized by the recognizer so that the parser can use it.

Token is another important entity in MathLet v2. Initial tokens are generated by using the results returned by symbol recognizer. In other words, initial tokens are the symbols written by a user. Each token stores information about its neighbour

tokens, component tokens and calculated likelihood values together with its 2D position information. Tokens also have their own \LaTeX and MathML codes. Tokens are expanded and new tokens are generated in MathLet v2. For instance, consider that initially there are neighbour tokens “a” and “2”, and the subscript rule is one of the applicable rules for the token “a”. This token can be expanded according to subscript rule after some necessary calculation and the token “ a_2 ” can be generated.

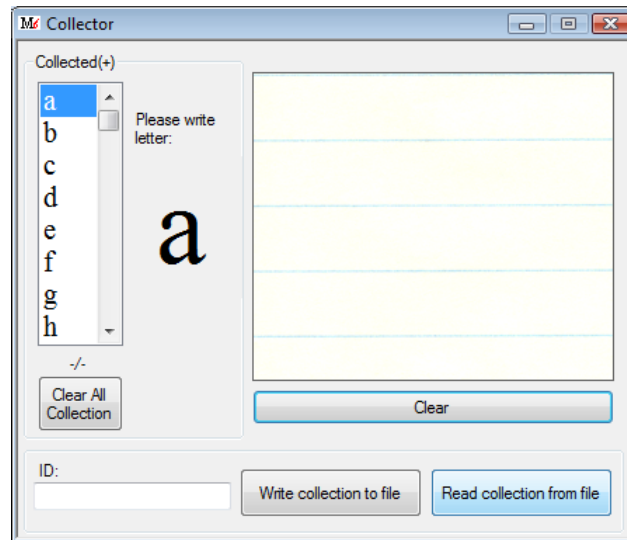


Figure 2.6: Interface of CharCollector

Parser

One of the most important entities of MathLet is parser. Parser creates initial tokens using the results returned by the recognizer and also specifies the neighbourhood relationships between them. To determine the neighbourhood, parser first checks the distance between tokens. If two tokens are close enough and there is not any other token between them, they will be marked as neighbour to each other. Moreover, parser controls the application of grammar rules and the generation of new tokens. If the likelihood of a generated token is less than predetermined threshold value, parser eliminates that token. In addition to these, parser creates neighbourhoods among all tokens and updates the list of existing tokens after each iteration.

MathLet v2 mistakes some certain characters for another certain character. For instance, the system cannot distinguish the symbols “1” and “(”, that is to say that users write “1”, but the character recognizer may recognize it as “(” or vice versa. To deal with this problem, when one of mistaken characters is recognized, parser

adds the other one to the initial list of tokens as an alternative without any check. When the character recognizer recognizes the symbol as “1”, the parser adds “(” to the initial list of tokens. One example of this can be seen from Figure 2.7.

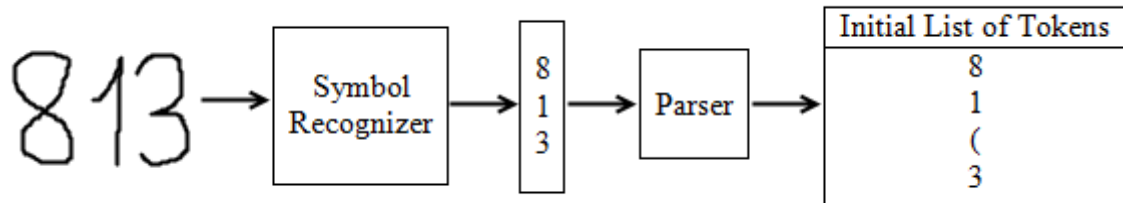


Figure 2.7: An example for mistaken symbol handling in MathLet v2

The parser of MathLet v2 adds the token “\times (\times)” to the initial list of tokens when the token “ x ” is initially recognized. This fact causes an increase in the number of tokens which Mathlet v2 has to consider for the MEs which consist of the symbol “ x ” or “ \times ”. Figure 2.8 shows the illustrative example of this. In this example, the initial token list has 5 tokens. The system needs to process one more token than it has to do.

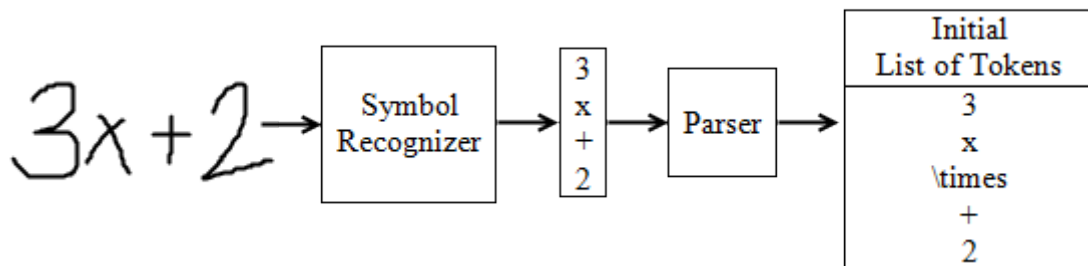


Figure 2.8: Initial token list after recognition of the symbol “ x ” in MathLet v2

Parser uses grammar rules to generate new tokens. Examples of the rules that used in MathLet v2 are the rule for subscript, superscript, square root, 2-stroke symbol generation, operators, multiple numbers, multiple letters and others. Each rule checks an appropriate token together with its neighbours and each rule is fired with an associated applicability score indicating how suitable it is to apply that rule in that situation. For instance, in Figure 2.9 a ME “ $2d$ ” is shown. For this example, subscript rule and alphanumeric rule are fired. These rules produce the tokens “ 2_d ” and “ $2d$ ” where the applicability score of subscript rule is greater than

alphanumeric rule. Hence, the likelihood of “ $2d$ ” is greater than the likelihood of “ 2_d ”. Statistical information is used to determine the likelihood of relationship between two neighbour tokens and calculated likelihood is assigned to generated token. For the same pair of tokens, more than one token can be generated with different fitness values using different rules as in this example.

A handwritten mathematical expression consisting of the digit '2' followed by the letter 'd' in a cursive, slightly slanted font.

Figure 2.9: Handwritten ME “ $2d$ ”

As mentioned, likelihood value is calculated for each generated token. Likelihood calculation is based on statistics and there are different statistics for different relationships. Fitness values are also combined with the fitness of component tokens. At the end, resulting fitness is assigned to generated token. For instance, subscript rule finds the fitness values for the nearest x and y positions of neighbour tokens by using appropriate statistics. Also, subscript rule uses different statistics for the comparison of the height and width of the tokens. Not only these fitness values but also individual fitness values of component tokens are used in the calculation of likelihood.

Histograms are used for the statistical representation of information used in likelihood calculation. For each relationship, there is a list of frequency values for histograms together with maximum and minimum values. Each rule first calculates frequency value and gets the likelihood for generated token by using appropriate statistics.

MathLet v2 is developed in Microsoft .NET Framework environment by using C# programming language similar to CharCollector and CharTrainer. It has Graphical User Interface (GUI) and the interface of MathLet v2 can be seen in Figure 2.10.

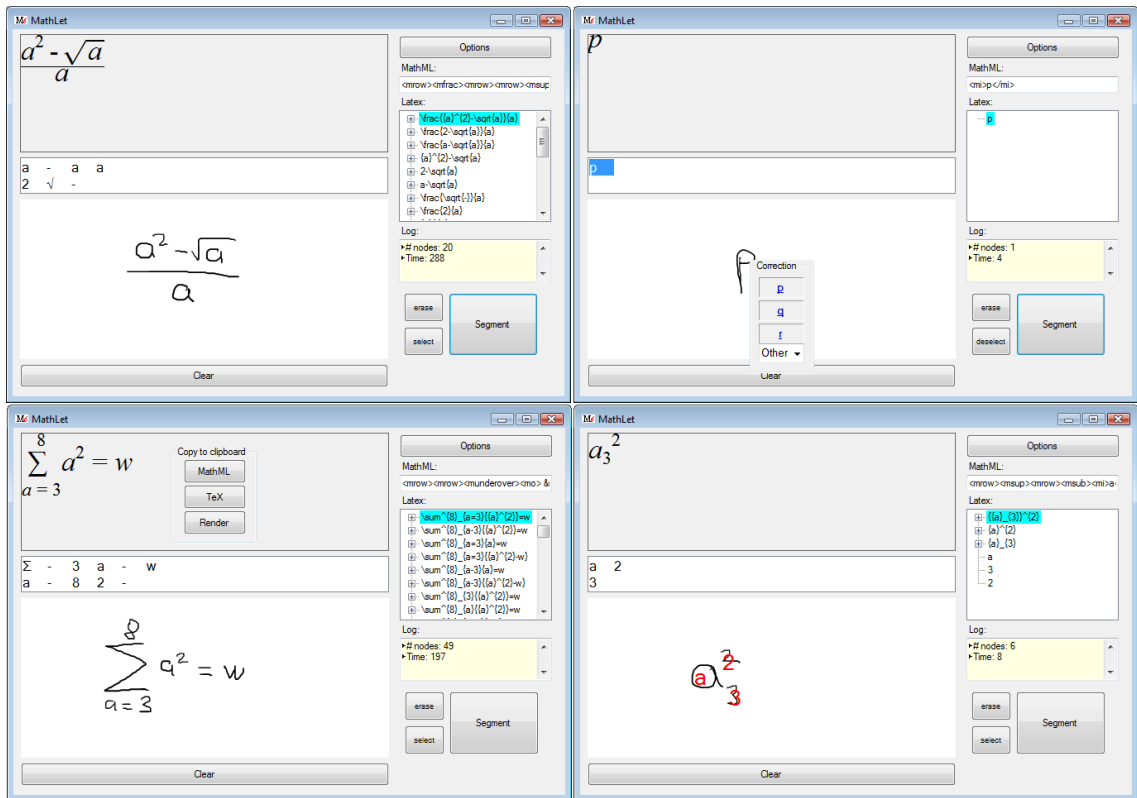


Figure 2.10: Interface of MathLet v2

3 MathLet v3

MathLet v3 is the name of the software developed to recognize handwritten MEs. It is the third version of MathLet as its name applies. MathLet v3 implements two-step process for the purpose of handwritten ME recognition; symbol recognition and parsing. Some of the structures that MathLet v3 uses are also used by MathLet v2. In MathLet v3, these structures are modified and extended in order to obtain better accuracy and time performance results which will be detailed in Section 4.

3.1 Symbol Recognition

MathLet v3 can recognize 102 different mathematical symbols that provides an opportunity for users to write MEs which contain wide symbol range. In the symbol recognition phase of MathLet v3, two classifiers are used. One of these classifiers uses offline features while online features are used by the other one. These two classifiers are combined in MathLet v3. Both classifiers are based on SVM and they are trained by using a program named CharTrainer which was developed in Microsoft .NET Framework using C# programming language and LibSVM [16] library.

As a training data for classifiers, 102474 instances that contain an information about mathematical symbols are used. These training data are collected from the students and extracted by using a program named CharCollector from the data provided by CROHME organizers. This program is an extended version of CharCollector which is also used by MathLet v2. One extension of this version is the ability to deal with 3-dimensional data. These 3-dimensional data include one more information in addition to x - y coordinates of points which form a mathematical symbol. CharCollector is also able to extract information about 102 symbols which may be written in different naming formats i.e., “<” or “\lt” may stand for the symbol “<”. This version can also extract information about the symbols “.” and “,” which were problematic before.

Both classifiers used by MathLet v3 return the prediction probability distribution over all classes as shown in Figure 3.1 rather than just returning the label of the predicted class. In other words, both classifiers give probability estimates. This property provides further information about the accuracy of the result returned by the symbol recognizer. Estimated probability can be defined as follows:

$$p_i = P(y = i | x), i = 1, \dots, k \quad (1)$$

where k is the number of classes, x is the data to be classified and $\sum_{i=1}^k p_i = 1$.

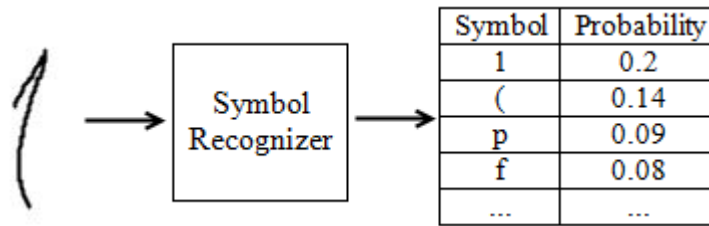


Figure 3.1: An example for symbol classification in MathLet v3

3.1.1 Offline Classifier

In order to extract offline features from training data to train offline classifier, the ink data is transformed into 32×32 bitmap image after scaling operations. Offline features are then extracted from this 32×32 bitmap image. The number of offline features extracted from bitmap image is 288.

32 of 288 offline features are extracted counting the number of black pixels in the half of bitmap image. In feature extraction, an image is first divided into 64 windows with 4×4 size. Then, black pixels are counted in each window and 64 number of black pixels are obtained. First 32 of these 64 number of black pixels are extracted as features. These 32 features can be defined as the number of black pixels in 4×4 windows which are located in the left half of the image.

128 of 288 offline features are extracted from the depth of the first black pixel in each row of image. There are 32 rows in 32×32 bitmap image. First, in the unrotated image, the index of the first black pixel in each row is found. Then, the image is rotated 90° and again the index of the first black pixel in each row is found. This procedure is also applied for 180° and 270° rotated images. At the end, 128

features are extracted. Each 32 features of these represent the depth of the first black pixel in each row in the image with different rotation.

Finally, the remaining 128 features are extracted as the number of black pixels in each row of the image. Firstly, in the unrotated image the number of black pixels in each row is counted and these form 32 of these 128 features. The number of black pixels in each of 32 rows is then counted in the rotated images. The image is rotated -45° , 45° and 90° . As a result, 128 features are formed by the number of black pixels in each row of 32×32 symbol image which is rotated -45° , 0° , 45° and 90° .

3.1.2 Online Classifier

In addition to offline features, online features are also used in the symbol recognition in MathLet v3. Online features are used by a different classifier namely online classifier. Online features are extracted from ink data which consist of strokes forming the symbol. The number of online features used is 38.

In order to extract online features, the following three steps are applied by MathLet v3:

- Resampling distance is calculated from ink data according to the predetermined number of equidistant points which will be included by resampled strokes. In MathLet v3, resampled strokes have 20 equidistant points.
- Strokes are resampled by using resample distance calculated in the first step. Resampled strokes have the predetermined number of equidistant points. For resampling, the codes written by Çağlar Tırkaz are rewritten in C# programming language and used.
- Resampled strokes are scaled to predetermined size and online features are extracted from scaled resampled strokes.

In the first step, the points which form the strokes are used. The distance between each consecutive points is calculated. Then these distances are added and the total distance is found. By using this total distance and the predetermined number of points, resampling distance is calculated. Resampling distance can be defined as a

distance between each consecutive points in the resampled strokes which have the predetermined number of equidistant points.

Secondly, strokes are resampled such that they have the predetermined number of points and the distance between each consecutive points is equal to the resampling distance calculated in the first step.

Thirdly and finally, resampled strokes are scaled to the predetermined size and from these resampled strokes, online features are extracted. Online features are delta features and extracted as a difference between consecutive points in scaled resampled strokes. Starting from the first point, x and y coordinates of each point is subtracted from x and y coordinates of the next point. For 20 points, there are 19 distances between them. Because the difference is calculated for both x and y coordinates, there are 38 delta features.

A tool is developed to view the points of the original and resampled symbols. The name of the tool is “View Ink Points”. This tool takes an input file which contains data about mathematical symbol. Tool first resamples the strokes of the symbol and then scales both the original and resampled symbol to the same size. Finally it shows both the original symbol and the resampled symbol. Figure 3.2 shows the interface of the tool with the example. In the example, the points of the symbol “ e ” is shown.

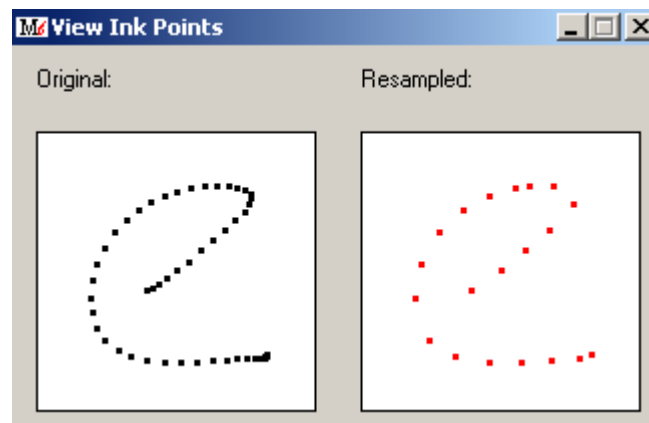


Figure 3.2: Interface of the tool “View Ink Points”

3.1.3 Classifier Combination

Online and offline classifiers are combined in Mathlet v3. In the classifier combination, the prediction probability distributions over classes returned by classifiers are

involved. First, the most probable symbol and its prediction probability which are returned by online and offline classifiers are found and then these probabilities are compared. The result of the classifier which returns greater probability for the most probable symbol is chosen as the result of symbol recognition.

Figure 3.3 illustrates the classifier combination in MathLet v3. In the figure, s_{on} denotes the most probable symbol predicted by the online classifier, s_{off} denotes the most probable symbol predicted by the offline classifier and s_{return} denotes the symbol returned at the end of symbol recognition. $p(s_{on})$ and $p(s_{off})$ denote the probability of most probable symbol predicted by the online and offline classifiers respectively.

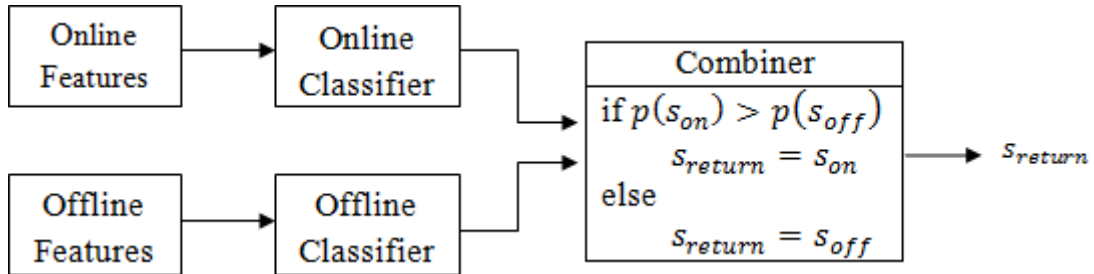


Figure 3.3: Classifier combination in the symbol recognition in MathLet v3

The offline classifier of MathLet v3 uses much more information than online classifier. The information used by offline classifier are extracted from the images of mathematical symbols and do not contain information about the stroke orders of mathematical symbols. On the other hand, online classifier uses 38 features and these features contain information about the order variations occurred while writing a mathematical symbol. For instance, for the symbol “a” written ambiguously in Figure 3.4, offline classifier does not consider the down stroke which is at the right of the symbol and recognizes it as the symbol “0”. In contrast, online classifier considers the down stroke and recognizes it correctly as the symbol “a”. As seen from this example, there is a trade-off between using only online or offline classifier. In order to deal with this trade-off, classifier combination is implemented in MathLet v3.

The accuracy rates of the online classifier, offline classifier and combined classifier are evaluated. Accuracy of each classifier is calculated as the rate of correctly classified symbols. In the evaluation, each classifier is trained with the same data

which is 80% of all data. The number of training symbols is 82015 for each classifier. The accuracy rate of each classifier is evaluated on the same test dataset which is completely different from the training dataset. Test data is chosen as the remaining 20% of all data. The number of test symbols is 20459. Table 3.1 shows the accuracy rate of each classifier.



Figure 3.4: The symbol “a” written ambiguously

Online Classifier	Offline Classifier	Combined Classifier
77.15%	90.18%	90.45%

Table 3.1: The accuracy rates of classifiers in MathLet v3

Two other combinations are also tested. In both combinations, first the prediction probability of the most probable symbol predicted by online classifier is compared to the predetermined threshold. If it is greater than the threshold, the most probable symbol predicted by the online classifier is chosen as a symbol recognition result, otherwise the most probable symbol returned by the offline classifier is chosen as the result of symbol recognition. The thresholds are 0.85 in one case and 0.9 in the other case. The accuracy results of these combinations are shown in Table 3.2. Notice that, in Combination 1 and Combination 2, the thresholds are 0.85 and 0.9 respectively. Training and test sets are the same as used for the online, offline and combined classifiers.

Online Classifier	Offline Classifier	Combination 1	Combination 2
77.15%	90.18%	90.25%	90.32%

Table 3.2: The accuracy rates of different classifier combinations

In addition to the rate of correctly classified symbols, the accuracy of each classifier on each symbol is also evaluated. Table 3.3 shows these symbol-based evaluation

results. The accuracies are evaluated as the rate of correctly classified instances for each symbol. For instance, online classifier can correctly classify the 82.85% of the instances which are labeled as “*a*”, while offline classifier can correctly classify 90.77% of them.

Symbol	Online Classifier	Offline Classifier	Combined Classifier
<i>a</i>	82.85%	90.77%	91.56%
<i>b</i>	79.80%	92.33%	93.09%
<i>c</i>	85.66%	88.52%	89.75%
<i>d</i>	71.05%	92.48%	94.74%
<i>e</i>	92.72%	91.39%	96.03%
<i>f</i>	64.16%	90.17%	87.86%
<i>g</i>	37.78%	48.89%	50%
<i>h</i>	58.33%	82.14%	80.95%
<i>i</i>	63.86%	81.53%	82.33%
<i>j</i>	60.22%	79.57%	80.65%
<i>k</i>	62.69%	83.42%	83.42%
<i>l</i>	12.96%	29.63%	24.07%
<i>m</i>	64.84%	79.69%	78.91%
<i>n</i>	83.55%	91.13%	92.98%
<i>o</i>	0%	0%	0%
<i>p</i>	81.76%	91.22%	92.57%
<i>q</i>	44.57%	70.65%	68.48%
<i>r</i>	57.25%	73.28%	74.05%
<i>s</i>	64.71%	56.47%	61.18%
<i>t</i>	45.25%	81.01%	79.33%
<i>u</i>	55.79%	80%	77.89%
<i>v</i>	76.14%	81.82%	85.23%
<i>w</i>	86.21%	94.83%	98.28%
<i>x</i>	84.93%	94.95%	96.38%
<i>y</i>	80.89%	91.84%	93.01%
<i>z</i>	51.5%	71.8%	69.17%

0	86.51%	96.98%	96.74%
1	74.93%	92.23%	90.8%
2	92.34%	95.62%	96.51%
3	92.27%	94.5%	97.25%
4	88.64%	93.18%	94.7%
5	64.54%	88.45%	86.85%
6	91.75%	91.26%	94.66%
7	61.42%	91.88%	88.32%
8	73.16%	91.58%	91.58%
9	72.68%	78.35%	81.44%
A	77.42%	77.42%	83.87%
B	79.25%	86.79%	92.45%
C	0%	16.07%	7.14%
E	35.71%	89.29%	89.29%
F	30.56%	86.11%	86.11%
G	47.06%	76.47%	76.47%
H	25%	90%	80%
I	0%	61.54%	53.85%
L	70.37%	96.30%	92.59%
M	21.05%	78.95%	57.89%
N	70.37%	74.07%	77.78%
P	0%	14.29%	14.29%
R	64.86%	75.68%	83.78%
S	4.35%	13.04%	13.04%
T	0%	84.21%	73.68%
V	0%	14.29%	14.29%
X	0%	17.24%	15.52%
Y	4.16%	41.67%	41.67%
–	82.42%	99.14%	99.07%
!	4.17%	75%	68.75%
(91.9%	94.64%	95.79%

)	92.21%	97.26%	98.11%
,	37.72%	61.68%	53.29%
/	1.37%	83.56%	75.34%
[41.86%	86.05%	79.07%
{	57.45%	78.72%	74.47%
}	46.81%	68.09%	59.57%
α	83.33%	76.98%	84.13%
β	69.7%	92.93%	90.91%
cos	79.41%	94.85%	94.85%
Δ	54.05%	94.59%	94.59%
\exists	0%	66.67%	33.33%
\forall	11.11%	55.56%	44.44%
γ	52%	60%	56%
\geq	66.15%	90.77%	90.77%
$>$	60%	84.44%	80%
\in	60%	90%	70%
∞	66.67%	90.35%	90.35%
\int	70.44%	86.68%	88.68%
λ	44.44%	94.44%	94.44%
\leq	72.84%	88.89%	90.12%
lim	63.01%	89.04%	95.89%
log	71.88%	92.19%	92.19%
$<$	40.74%	85.19%	79.63%
μ	48.72%	82.05%	82.05%
\neq	62.5%	89.29%	89.29%
ϕ	42.86%	83.67%	83.67%
π	67.13%	88.81%	88.11%
\pm	50%	79.41%	85.29%
\prime	0%	0%	0%
\rightarrow	59.05%	96.19%	94.29%
σ	0%	80%	70%

sin	82.22%	94.44%	96.67%
$\sqrt{\quad}$	72.45%	98.81%	99.05%
\sum	76.33%	96.45%	96.45%
tan	82.46%	89.47%	92.98%
θ	81.94%	88.19%	89.58%
]	72.09%	90.7%	93.02%
	1.12%	26.97%	21.35%
+	88.9%	97.81%	97.97%
=	93.01%	96.56%	97.39%

Table 3.3: Symbol based accuracies of classifiers

From the results, it is obtained that the accuracy of online classifier is very low compared to the accuracy of offline and combined classifiers. One reason of this fact is that when individuals write the mathematical symbols in a different way, online classifier cannot recognize it. For instance, the symbol “2” is generally written by starting from left center point as shown in the left of Figure 3.5. If the symbol “2” is written in a reverse way as indicated in the right of Figure 3.5, online classifier cannot recognize it. Online classifier recognizes the symbol “2” which is written in a reverse way as the symbol “0”, while offline classifier recognizes it as the symbol “2”.



Figure 3.5: The symbol “2” written in two different ways

Another reason of the low accuracy of online classifier is that some capital letter symbols such as *C*, *S*, *X*, *V* and *P* are generally written in the same pattern as the lower case letter symbols of these. When one of these symbols is written, online classifier mostly recognizes them as lower case letters.

For the symbols which are classified by combined classifier with the accuracy rate less than or equal to 50%, the symbols which are mostly mistaken for these

symbols are investigated. Table 3.4 shows these symbols. For instance, from Table 3.4 it is seen that the misclassified instances of the symbol “ g ” are mostly classified as the symbol “9” by combined classifier .

Symbol	Mistaken Symbol
g	9
l	1
o	0
C	c
P	p
S	s
V	v
X	x
Y	y
\exists	3
\forall	x
l	1
	1

Table 3.4: Mistaken symbols for the symbols classified by combined classifier with the rate less than or equal to 50%

Furthermore, it is also obtained that the misclassified instances of the symbols “ z ”, “ m ”, “ c ” and “ $>$ ” are mostly classified by combined classifier as the symbols “2”, “ n ”, “(” and “)” respectively.

3.2 Parsing in MathLet v3

Symbol recognition step is followed by parsing in MathLet v3. In parsing phase, grammar rules which define relationships between tokens are used. Initial tokens are created from symbols and these tokens are expanded during parsing phase by parser in order to obtain whole ME at the end.

3.2.1 Tokens

Symbols and MEs are represented by a structure called “token” in MathLet v3. Each token stores its own L^AT_EX and MathML codes of mathematical symbol or ME which it represents. Each token has also a likelihood value which defines its fitness. Components of the token are also stored by the token. Each token also stores its bounding box and some 2D information about its position such as top right point of it. For instance, a token representing the ME “ a^3 ” has a L^AT_EX code “ $\{a\}^{\{3\}}$ ”, a MathML code “ $\langle\text{msup}\rangle\langle\text{mi}\rangle a \langle\text{mi}\rangle\langle\text{mn}\rangle 3 \langle\text{mn}\rangle\langle\text{msup}\rangle$ ” and the component tokens representing “ a ” and “ 3 ” together with likelihood value, bounding box and 2D information. A visual representation of the token “ a^3 ” is shown in Figure 3.6.



Figure 3.6: A visual representation of the token “ a^3 ”

Parsing step in MathLet v3 starts with creating initial tokens from the symbols recognized in the symbol recognition step. Here, the probability distribution over classes returned by the symbol recognizer is used. According to the most probable symbol, parser creates the initial tokens. For instance, if the symbol “ θ ” is the most probable symbol according to the probability distribution, a token representing the symbol “ θ ” is created by parser. If the most probable symbol is “ x ”, then two tokens representing the symbols “ x ” and “ \times (times)” are created.

Parser in MathLet v3 applies different procedures while creating the initial token for the symbols which are generally mistaken for another symbol. According to the prediction probability of the most probable symbol, more than one token may be created by the parser. For example, MathLet v3 mistakes the symbol “1” for the symbols “(” and “|”. When the symbol “1” is the most probable symbol, parser checks prediction probability of it. If this probability is greater than or equal to 0.8 only a token representing the symbol “1”, else if this probability is less than 0.6 three tokens representing the symbols “1”, “|” and “(”, otherwise two tokens representing the symbols “1” and “(” are created. Similar procedure is also applied for the symbol

“t” which is mistaken for “+”, the symbol “z” which is mistaken for the symbol “2” etc. The goal of this procedure is to decrease errors due to the misrecognition of symbols in symbol recognition step. An example list of initial tokens for the ME “813” is shown in Figure 3.7. It should be noted that the prediction probability of the symbol “1” is greater than or equal to 0.8 and the symbols “8”, “1” and “3” are the most probable symbols in this example.

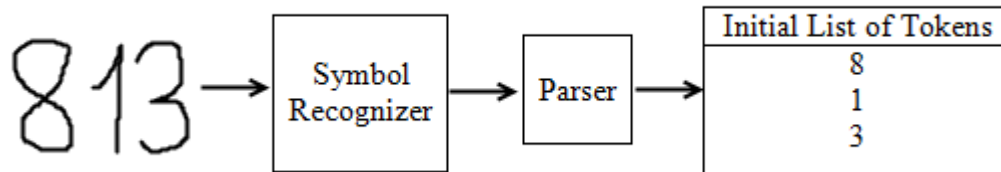


Figure 3.7: An example for mistaken symbol handling in MathLet v3

After the initial token list is created, parser creates the initial neighbourhood between initial tokens. In order to do this, some checks are made among each pair of tokens. First, the distance between tokens are calculated. If they are close enough and there is no third token between them, tokens are marked as neighbour by the parser.

After the creation of initial neighbourhoods, parser makes special checks to distinguish the symbols “ x ” and “ \times ” when the initial token list has one of these tokens. These checks are experimental and depend on the content of ME. After checks, if parser determines that the mistaken token is “likely x ” then the token representing “ \times ” is removed or vice versa. If parser cannot make such decision, two tokens remain in the initial token list. Parser makes different checks to decide that mistaken token is “likely x ” or “likely \times ”. If the left and right neighbours of the mistaken token is a number and the structure “number \times number” is very likely for these tokens, then the mistaken token is marked as “likely \times ”. If there is no neighbour in the left or right of the mistaken token, then it is marked as “likely x ”. Furthermore, if the right neighbour of the mistaken token is plus or minus and the positions of the mistaken token and its right neighbour is appropriate for being a horizontally neighbour, then the mistaken token is marked as “likely x ”. This procedure provides a decrease in the number of initial tokens for some of the MEs which contain

“ x ” or “ \times ”. The time performance of MathLet v3 increases while processing these MEs, because the number of tokens which MathLet v3 has to process decreases. An example is shown in Figure 3.8. In this example, parser eliminates the token “ \times ” after making checks. It should also be noted that, the symbols “3”, “ x ”, “+” and “2” are the most probable symbols according to the probability distributions returned by the symbol recognizer for each symbol in this example.

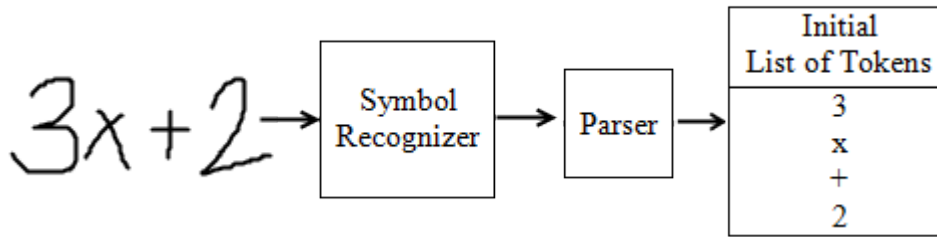


Figure 3.8: Initial token list after the recognition of the symbol “ x ” in MathLet v3

3.2.2 Grammar Rules

As the next task, parser expands existing tokens and generates new tokens by applying grammar rules. There are many grammar rules in MathLet v3. These rules are considered in four different groups. This grouping is done for the purpose of application order which will be detailed in Section 3.2.3. Four groups can be defined as follows: rules defining the conditions to generate tokens representing symbols written in more than one stroke such as the symbol “=”, operator rules defining the conditions to generate tokens representing expressions such as “ $3 + 4$ ”, equality operator rules defining conditions to generate tokens such as “ $x = y$ ”, “ $2 \leq 3$ ” and others i.e., a rule defining conditions to generate multi-number terms such as “123”.

The rules in the first group define the conditions to generate a token representing a symbol written in more than one stroke such as “=”, “ x (may be written like a concatenation of the symbols “)”) and “(”, “ \leq ”, “...”, “ \div ”, “tan”, “cos”. Each rule in this group takes one candidate token which may be recognized as a separate symbol while it is written as a stroke of multi-stroke symbol. Then, the positions of neighbour tokens of the candidate token which represents appropriate symbol is checked by the rule. If such a token exists and that token satisfies further conditions, a token representing multi-stroke symbol can be created. Each rule checks different conditions for generating tokens representing different symbols.

For instance, in MathLet v3 there is a rule to define the conditions for generating a token representing the symbol “=”. A rule checks each of the tokens “-” separately as a candidate token. Then, rule checks whether there is another token representing “-” at the top of the candidate token. If such a token exists, then the widths of two tokens are compared. If the difference of the widths of two tokens are less than 0.75 of width of each token, a token representing the symbol “=” can be generated.

A rule in the second group is the operator rule which defines the conditions for generating MEs containing operator and its operands. The rule takes a token representing an operator symbol which can be “+”, “-”, “×”, “÷” or “±”. Then, the neighbour tokens of that token are checked. If the type (variable, number etc.), height and baseline of neighbour tokens are appropriate for being an operand, new token can be generated. Likelihood calculation is also defined by the rule. In the calculation, the widths and heights of components are considered together with distance between them.

One more condition is also checked while selecting a neighbour token in the right of the operator token. The rule checks whether a neighbour token in the right is included as a component in another neighbour token which is in the right of the operator token and has likelihood value greater than the threshold. If this condition is satisfied, that neighbour token is not expanded by the operator rule. This condition check provides a decrease in the number of tokens that MathLet v3 has to process and this fact provides an increase in the time performance of MathLet v3. As an example, consider the ME “1 + 2435”. In the left of the token “+” there can only be one token representing “1”, while in the right there can be tokens representing “2”, “24”, “243”, “2435”. If the likelihood of the token “2435” is greater than the threshold value, other subexpressions are not generated. An illustrative example can be seen from Figure 3.9.

The rule in the third group is a rule that defines conditions to generate tokens containing equality operators which are “=”, “≠”, “≤”, “<”, “>” and “≥”. A rule takes a candidate token representing one of the equality operator symbols. A rule then checks the neighbour tokens in the left and right of candidate token. Contextual information is also used by the rule. If the neighbour token represents one of the operators, new token is not generated according to the assumption that

a ME does not contain a subexpression like “+ =”, “≤ −” etc. The calculation of likelihood value of new token is also defined by the rule. In the calculation, baselines of component tokens are considered together with distances between them.

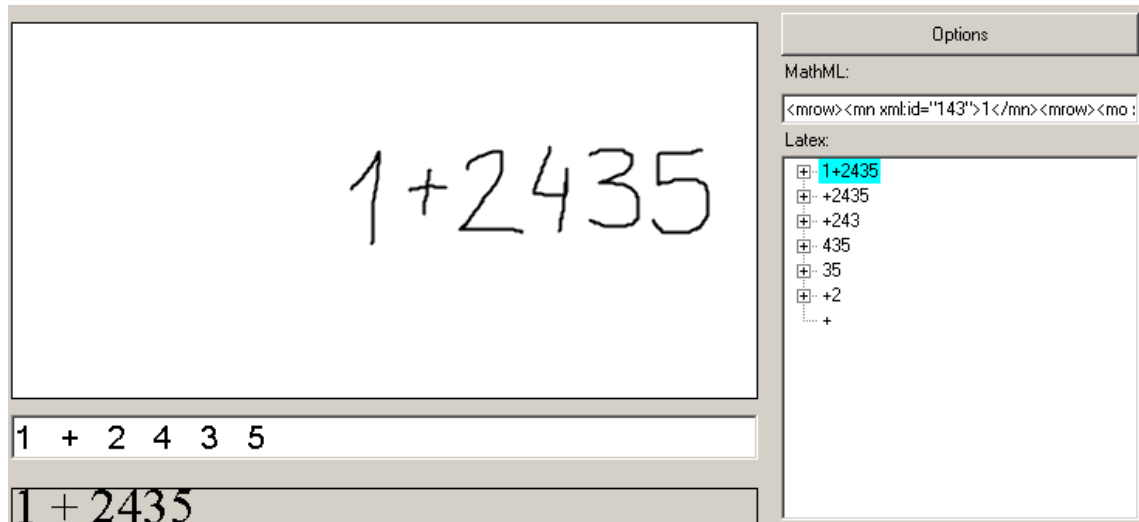


Figure 3.9: Recognition result for the ME “1 + 2435”

In the fourth group, there are many rules defining conditions for different neighbourhood relationships between tokens. There are rules for subscript and superscript relationships which takes a base token as a candidate and checks appropriate position for subscript and superscript relationships. For instance, to generate the token representing the ME “ a_1 ”, the rule takes the token “ a ” as a candidate token. Then the bottom-right of it is checked whether there is a token or not and the rule finds a token representing “1”. The rule also makes checks based on contextual information. Then the token “ a_1 ” can be generated with its likelihood value. The likelihood value is calculated comparing the nearest x and y points of components, the widths and heights of them. For subscript rule, the size of the token in the subscript should be smaller than the size of base token. In this group there is a fraction rule defining the conditions to generate fractions such as “ $\frac{1}{2x}$ ”. There are rules defining conditions to generate numeric terms such as “241”, alpha terms such as “ xy ”, alphanumeric terms such as “ $2a$ ”, multiple terms such as “ a^2b^2 ”, subexpressions containing “lim” such as “ $\lim_{x \rightarrow \infty} x^2$ ”, square roots such as “ $\sqrt{x_n}$ ”, functions such as “ $\sin x$ ”, “ $\tan y$ ”, “ $\log b$ ”, summation and integrals such as “ $\sum a$ ”, “ $\int x^2 dx$ ”, parenthesis and absolute values such as “ $(2 + 3y)$ ”, “ $|-2|$ ”. Each rule defines appropriate checks based on

spatial relationships and contextual information. Likelihood calculation specific to relationship is also defined by the rules.

Parser in MathLet v3 applies the applicable grammar rules to existing tokens and generates new tokens. If a rule is applicable, then parser generates new token with likelihood value. For instance, consider the ME “ 2^x ” which is written ambiguously as shown in Figure 3.10. This ME may be recognized as “ 2^x ” or “ $2x$ ”. The rule for superscript relation and the rule to generate alphanumeric terms check the relative positions of the tokens “2” and “ x ”. Both rules find the relative position of these tokens appropriate to generate the tokens “ 2^x ” and “ $2x$ ”. In other words, both rules are applicable to the tokens “2” and “ x ”.



Figure 3.10: The ME “ 2^x ” written ambiguously

Parser in MathLet v3 calculates the likelihood value and assigns it to the generated token. The calculation of likelihood value is done according to the grammar rule which is applied by parser to generate the token. For instance, the parser assigns likelihood values to the tokens “ 2^x ” and “ $2x$ ” according to the superscript rule and the rule to generate alphanumeric terms for the ME shown in Figure 3.10. The parser calculates the likelihood value for the token “ 2^x ” according to the superscript rule by comparing the x and y position and the width and height of the component tokens “2” and “ x ”. According to the superscript rule, y position of the token “ x ” should be greater and the height and width of the token “ x ” should be less compared to the same properties of the token “2”. The likelihood of the token “ $2x$ ” is calculated according to the rule to generate alphanumeric terms by comparing the distance between component tokens “2” and “ x ” and the baselines of them. The tokens “2” and “ x ” should be close to each other and y positions of their baselines should be comparable. Consequently, the calculated likelihood value of the token “ 2^x ” is greater than the likelihood value of the token “ $2x$ ”.

3.2.3 Rule Application

Parser in MathLet v3 manages the application of grammar rules on existing tokens. If any new token is generated after rule application, the likelihood value of new token is checked by the parser. If this value is greater than the predetermined threshold, new token is added to the list of existing tokens, otherwise it is eliminated by the parser.

Rule application is made in the predetermined order by the parser and after each iteration of rule application, the parser updates the existing neighbourhood relationships between existing tokens or creates new relationships. Parallel programming is involved in this neighbourhood creation step and this provides an increase in the time performance of MathLet v3.

As the first step of rule application, generation rules are applied continuously on appropriate existing tokens until no new token is generated. For instance, a user wants MathLet v3 to recognize the ME “ $y+16 = x$ ”. Consider that, a user naturally writes the symbol “=” in two strokes. Parser in MathLet v3 first generates the token representing the symbol “=”.

As the second step, parser applies the rules in the fourth group continuously until no new token is generated. For the ME “ $y + 16 = x$ ”, the parser creates tokens representing subexpressions “+1” and “16” according to the rule for alphanumeric terms and numeric terms respectively. In the third step, the operator rule which is given in the second group is applied by the parser. The application of this rule is done again until no new token can be generated. At this time, parser in MathLet v3 creates a token representing the subexpression “ $y + 16$ ” according to the operator rule.

After applying the operator rule, as the fourth step, parser applies the equality operator rule until it is not possible to generate new tokens. For the ME “ $y+16 = x$ ”, there is one token representing the equality operator “=”. According to the rule, parser will create three tokens representing the subexpressions “ $6 = x$ ”, “ $16 = x$ ” and “ $y + 16 = x$ ”.

Parser then repeats the second step to see whether any new tokens can be generated from existing tokens. If any new tokens can be generated, then it repeats second, third and fourth step using existing tokens until no new token can be gen-

erated. For our example, no new token can be generated after repeating the second step and parsing step is done. Figure 3.11 shows the input ME and the recognition result of it which is returned by MathLet v3. From the Figure 3.11, subexpressions can be seen in the list of recognition results presented in the right. At the bottom, the readable view of top-ranked recognition result is shown.

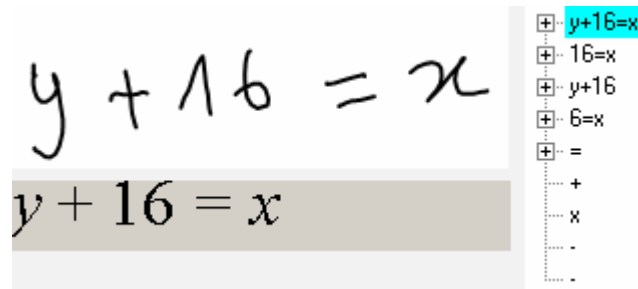


Figure 3.11: Recognition result for the ME “ $y + 16 = x$ ”

Figure 3.12 shows the tokens generated after each rule application step while the ME “ $y + 16 = x$ ” is being parsed. The component tokens of generated tokens are also shown. It should be noted that after the application of equality operator rule, three tokens are generated. For the sake of simplicity, only one of these three tokens is shown in Figure 3.12.

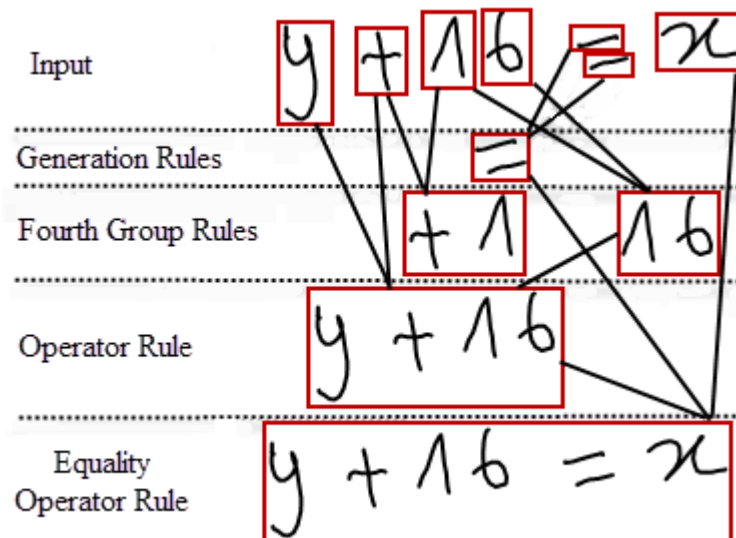


Figure 3.12: The tokens generated while the ME “ $y + 16 = x$ ” is being parsed

Parsing process is manually stopped if it takes MathLet v3 to finish it more than 5 minutes. The parsing process in MathLet v3 is generally stopped, when MEs with

too many number of symbols are recognized.

3.2.4 Sorting Existing Tokens

After parsing process is finished, MathLet v3 sorts the existing tokens. First, extra check based on contextual information is made for tokens which contain the symbols paranthesis or absolute value. The number of left and right parantheses (“(” and “)”) and the number of the symbol absolute value (“|”) are counted. If the number of the symbols left and right paranthesis is not equal to each other or the number of the symbol absolute value is not even, the likelihood value of the token is manually decreased.

Then, the existing tokens are sorted. In this sorting, the number of components and the likelihood values of two tokens are compared. A token which has more components has precedence. If more than one token have the same number of components, then the likelihood value of them are compared. A token having greater likelihood has precedence.

MathLet v3 uses this sorting for the presentation purpose of recognition results. The recognition results are ordered according to this sorting. Top-ranked token is presented at the top, second one is presented below of it and so on. An example can be seen in the right of Figure 3.11.

3.2.5 Parsing MathML Codes

The process of sorting existing tokens is followed by parsing the MathML code of top-ranked token. The need of this process emerged with the low recognition results obtained in CROHME 2011 (see Section 5). The system which participated in CROHME 2011 was MathLet v2. The major source of errors which caused low recognition results in CROHME 2011 was MathML problems. MathLet v2 generated wrong MathML codes for the most of the MEs that are correctly recognized. These MathML problems of MathLet v2 were difficult to fix within the existing parsing algorithm.

The problem was that MathLet produces MathML codes in which there are misplaced “mrow” elements. In the correct format, symbols have to be grouped in “mrow” elements iteratively starting from the right. The rightmost two symbols are

grouped in “mrow” and then each symbol in the left are grouped in another “mrow” element. As an example, consider the ME “ $a + c = b$ ”. The MathML code of this expression can be seen in Table 5.1.

In MathLet v3, the MathML codes of each subexpression is created by the parser based on the rule which generates that subexpression. Hence, grouping the tokens in “mrow” elements is also achieved according to these rules. Each rule specifies grouping tokens in “mrow” element by specifying criteria based on component tokens. Consider the parsing of the ME “ $a + c = b$ ” by the parser in MathLet v3. The parser first creates the token “ $a + c$ ” according to the operator rule. The MathML code of the token “ $a + c$ ” is created by grouping the symbols “+” and “ c ” in one “mrow”, and then grouping all of three symbols in outer “mrow”. Then the token “ $a + c = b$ ” is created according to the equality operator rule by the parser. The MathML code of the ME “ $a + c = b$ ” is created by grouping the symbols “=” and “ b ” in one “mrow”, and the remaining part in one outer “mrow”. As a result, according to the parsing process of MathLet v3, expression tree and MathML code for this ME will be as in Figure 3.13.

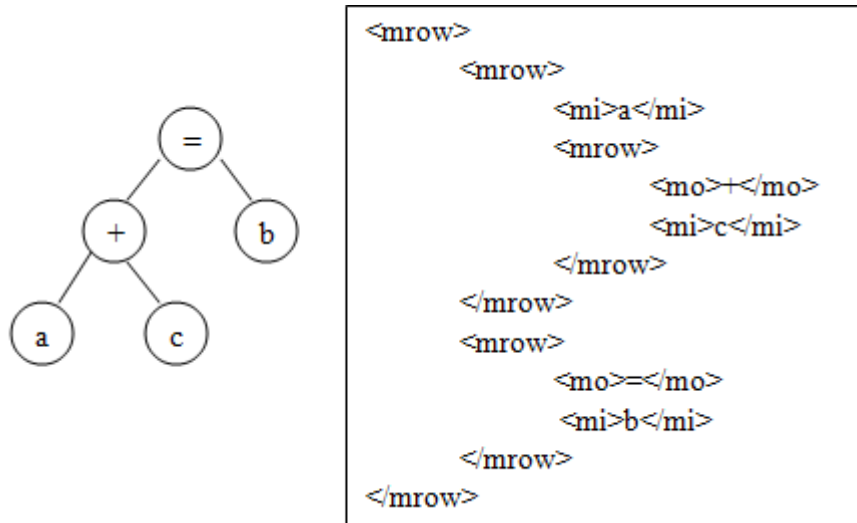


Figure 3.13: Expression tree and the MathML code for the ME “ $a + c = b$ ” produced by MathLet v3 before MathML parsing

Because the console application of MathLet v3 gives an InkML file containing top-ranked recognition result as an output, MathML code of top-ranked token is parsed at the end of parsing process. In order to do this, first each “mrow” element in MathML code is removed. Then, the suffix of the remaining MathML code which

contains two symbols is detected. Finally, grouping the symbols in “mrow” elements is done by inserting “mrow” elements into the correct places.

MathML codes included in output InkML file can still be problematic for some cases in MathLet v3. For instance, the MathML codes of nested fractions and nested square roots such as $\frac{x}{\frac{x}{y}}$ and $\sqrt{1 + \sqrt{2}}$ cannot be constructed correctly in MathLet v3.

3.3 Accessibility

MathLet v3 has four applications each can be accessed by different ways. All applications of MathLet v3 are developed in Microsoft .NET Framework environment using C# programming language. Two of these applications can be accessed through the Web, while the others are Windows and console application.

The first application of MathLet v3 is a Windows application and has GUI to facilitate human-computer interaction. Users write their own MEs and get the recognition results for them. This application also provides a functionality for users to upload InkML files. The MEs included by these files can be viewed and recognized by running this application.

The second application of MathLet v3 is a console application which takes one input file and generates one output file. The input of this application is an InkML file which contains the MathML code of the expression to be recognized together with stroke-level information such as points of strokes and segmentation of them. The output is also an InkML file which contains the MathML code and stroke segmentation information of the best recognition result. Users have to run the application by calling the executable file created by Microsoft Visual Studio automatically after building the solution. Users must invoke the executable file from Windows command prompt together with two parameters which are the paths of input and output InkML files. Some details of InkML and MathML will be given in Section 5.1.

MathLet v3’s third application is used to upload InkML files through the Web. It returns recognition results for the uploaded InkML file. Users can choose the InkML file which they want and see the recognition results for it. The system returns top-5 recognition results together with their \LaTeX codes.

As the implementation of MathLet v3's third application, a Web page in HyperText Markup Language (HTML) is created. In this web page, there is an introductory explanation about the page and the list of supported symbols. There are also appropriate titles and two buttons for browsing the files and recognizing ME. Clicking the browsing button provides that a user browses the files stored in his/her computer and chooses an InkML file to be recognized. Other button is used to get recognition results for chosen InkML file. A web page which is displayed with Mozilla Firefox browser can be seen in Figure 3.14. Web pages are also available online at <http://ferrari.sabanciuniv.edu/MathLetInkml/> and <http://ferrari.sabanciuniv.edu/MathLetInkml2/>. The difference between these two web pages is that they run different web applications which are able to recognize different list of symbols.

As the second step of the implementation of third application of MathLet v3, ASP.NET MVC 2 Web Application is developed. A user runs this application by clicking recognize button in the Web page and the Web application returns the result page. In this Web application, first the Windows application of MathLet v3 is modified and used as the models of an application. Secondly, a result page is developed as the view of the application. This page is shown after recognize button is clicked on the Web page. Thirdly, a controller for the Web application is developed. Sample result pages which are displayed with Microsoft Internet Explorer can be seen in Figure 3.15.

The fourth application of MathLet v3 provides that users can write their own MEs through the Web page. Users can get top-5 recognition results for the ME which they write as similar to the third application. Users are also able to clear the ME which they write by clicking an appropriate button.

As the implementation of the fourth application, first an ink-enabled user control which provides that users can write their own MEs through the Web is created. Users can also clear the MEs which they write and write new ME from scratch. This user control is created by using [17] and included in ASP.NET MVC 2 Web Application.

Similar to the third application, entities of MathLet v3's Windows application are modified and used as models. New views and controller are also developed. In this web application, there are two views. These views can be considered as the

Web pages which have different operational tasks. One of these views is for writing a ME and clearing it, while the other one provides the recognition results for a ME.

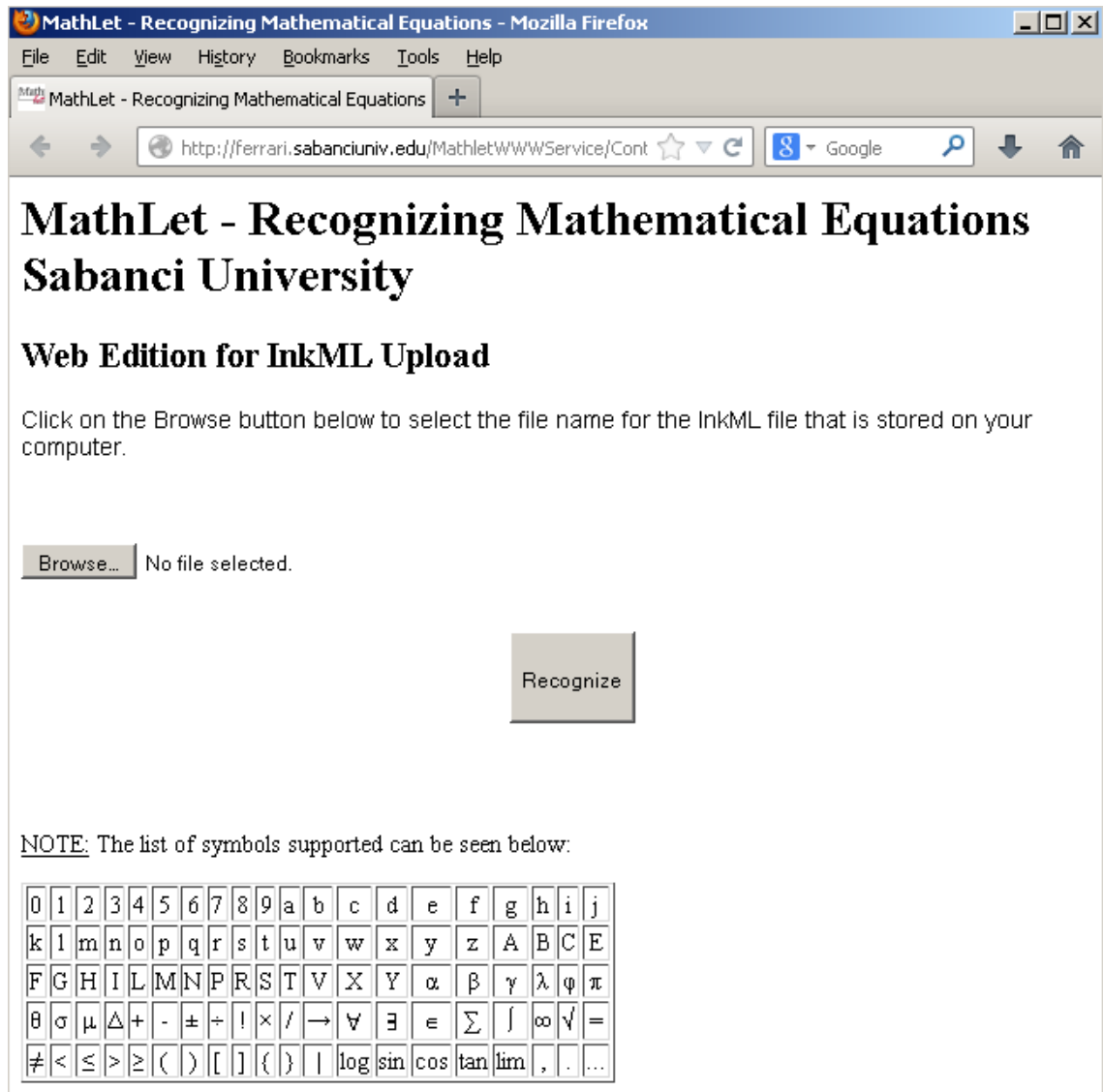


Figure 3.14: InkML upload Web page of MathLet v3

In contrast to the third application, fourth one does not have any Web page written in HTML. Instead views are used. Figure 3.16 shows the sample Web interface pages and their corresponding result pages. In this figure, there are two different MEs written on two Web pages. Below the Web pages, there are result pages which contain recognition results for these MEs. In addition to this figure, fourth application can also be seen online from <http://ferrari.sabanciuniv.edu/MathLet/>. To view the Web page correctly, users need to use Microsoft Internet Explorer and follow the instructions given on the Web page.

For the Windows application of MathLet v3, a setup file is created in order to improve the accessibility of it. A user simply installs the MathLet v3's Windows application by running setup file and uses it. Similarly, a user is able to remove or repair MathLet v3 by running the file which is also installed after the setup.

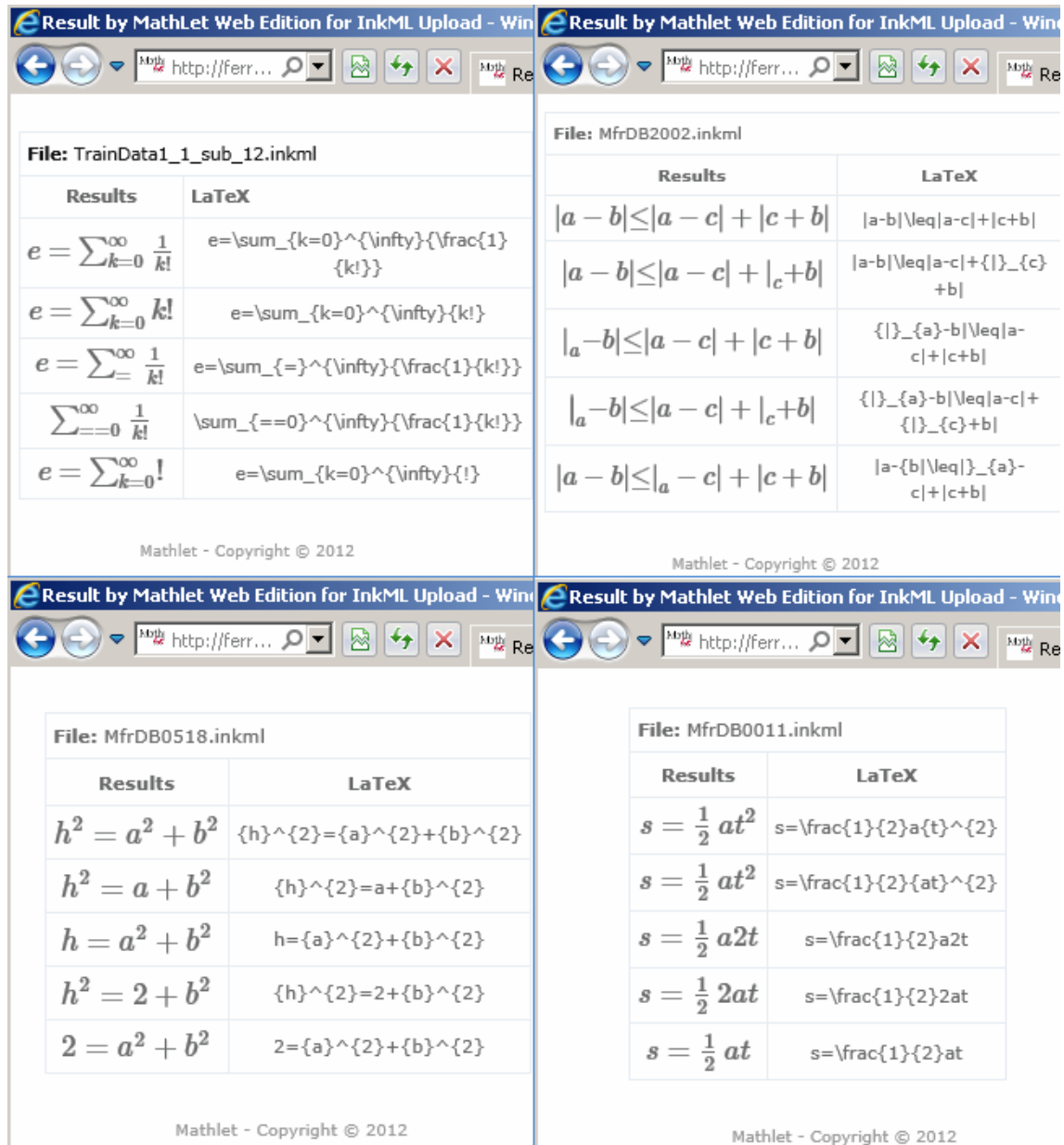


Figure 3.15: Example result pages for uploaded InkML files

Before creating the setup file for MathLet v3, first the file to remove or repair it is created. To do this, the codes provided in MSDN forums [18] are used. A program is developed using Microsoft .NET Framework and C# programming language according to these codes. The file created is included in the installation folder of the

application.

In order to create the installation file for MathLet v3, a “Setup Project” [19] in Microsoft .NET Framework is developed. In this project, the files which are needed to run Windows application of MathLet v3 are included. These files are included in the application folder. Moreover, the file for removing and repairing MathLet v3 is included. An icon is created for this file and this file is included together with existing icon file of MathLet v3 in the project.

“Setup Project” gives two output files. One of them is “MathLet.msi” and the other one is “setup.exe”. Instead of giving the files of Release or Debug folder of an application directory, now these 2 files are given to the users.

Users install the Windows application of MathLet v3 by running the file named “setup.exe”. After installation is completed, users will have shortcut on their desks-tops and MathLet v3 is appeared under Start menu in Windows operating systems. In the same path, they will see the file to repair or remove MathLet v3.

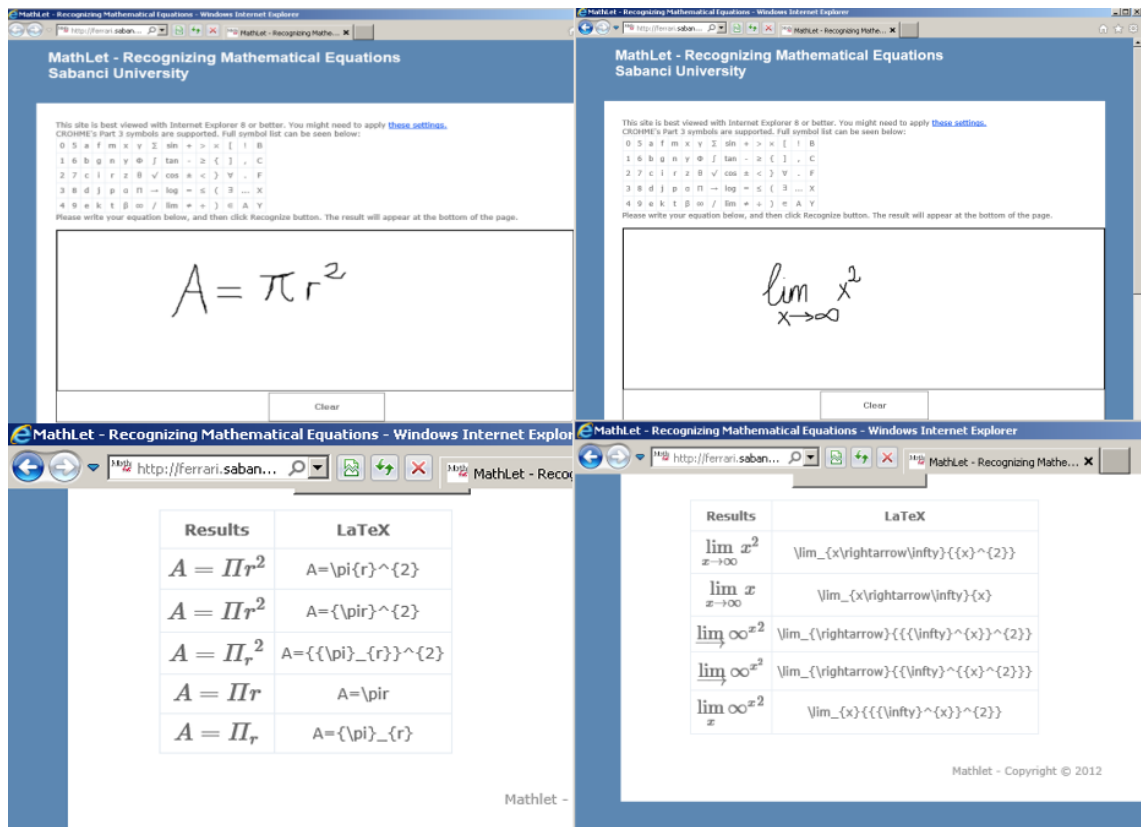


Figure 3.16: The web interface of MathLet v3

4 Accuracy and Time Performance Evaluations

In this thesis, MathLet is evaluated according to the accuracy and time performance of it. Accuracy evaluation results are given based on CROHME results which show the progress of MathLet clearly. Time performance evaluation results of MathLet v3 which includes improvements are given together with the initial results of MathLet v3 which does not have improvements on time performance where both systems are evaluated based on the same metric which is also defined further in this section.

4.1 Accuracy Evaluation

To evaluate the accuracy of MathLet, the evaluation results that MathLet has obtained in CROHME competitions are used. MathLet participated in CROHME in 2011, 2012 and 2013. For different recognition tasks, the accuracy evaluation results that MathLet obtained are reported in Table 4.1. It should be noted that given evaluation results show the rate of fully recognized MEs by MathLet. The details of these evaluation results, evaluation metrics and recognition tasks will be given in Section 5.

Task	CROHME 2011	CROHME 2012	CROHME 2013
Part-I	0.55%	22.22%	N/A
Part-II	0.29%	7.97%	N/A
Part-III	N/A	4.92%	N/A
Part-IV	N/A	N/A	8.35%

Table 4.1: The accuracy evaluation results of MathLet

As seen, the accuracy evaluation results of MathLet v2 were very low in CROHME 2011. For both Part-I and Part-II tasks, the evaluation results were lower than 1%. In addition to improvements on the symbol recognizer and parsing, the efforts made

to correct MathML errors made MathLet v3 more successful in CROHME 2012. For both Part-I and Part-II tasks, a huge gain was obtained in CROHME 2012 in which MathLet v3 did not have special system for Part-III task. Despite of the fact that Part-IV task is more difficult than others, CROHME 2013 results of MathLet v3 are better than Part-II and Part-III results obtained by MathLet v3 in CROHME 2012. Training the symbol recognizer with a huge data and extensions in parsing play key role in obtaining these results.

As a result, starting from 2011, accuracy evaluation results of MathLet have shown an increasing trend with the improvements made on the symbol recognition, parsing and MathML parsing. In short, although the recognition task becomes more difficult, the accuracy of MathLet has been increased from 0.55% to 8.35% within two years.

4.2 Time Performance Evaluations

In order to evaluate the time performance of MathLet v3, first an evaluation metric is decided. Then, initial measurements are made on MathLet v3 which does not have any improvements. The same measurements are made again on MathLet v3 after the improvements. Expression-level recognition rates are also found in both measurements.

4.2.1 Evaluation Metrics

It takes MathLet v3 long time to process long MEs. When a ME contains many symbols, MathLet v3 might not give recognition result for it or might recognize it after long processing time. Before making any improvements, some analysis is made on MathLet v3.

In MathLet v3, parsing phase is managed by a function named “recognize”. This function is invoked after a ME is written. Thus, the time which MathLet v3 takes to process this function is decided to measure. In addition to the time measurement, the recognition results of MathLet v3 are also measured.

For CROHME 2011, competition organizers provided 921 training files which include Part-I and Part-II symbols. In time measurements, these training files are used and the processing time of “recognize” function for these training files is mea-

sured. In these measurements, a version of console application of MathLet v3 which is specially prepared for Part-II is used. In other words, the systems used in measurements can only recognize the symbols which are included in Part-II recognition task of CROHME (see Section 5).

4.2.2 Initial Time Performance Evaluation Results of MathLet v3

Time performance measurements provide the processing time for each file. From this information, some statistics are extracted. First, the number of files which requires specific interval of processing time is extracted. This information is shown in Table 4.2.

Processing Time	Number of Files	Proportion
Less than 1 second	415	45.06%
Less than 10 seconds	524	56.89%
Less than 30 seconds	566	61.45%
Between 30 seconds and 5 minutes	73	7.93%
More than 5 minutes	282	30.62%

Table 4.2: The initial time performance evaluation results of MathLet v3

As seen from Table 4.2, there are 282 files which require more than 5 minutes to be processed by MathLet v3. The number of strokes included by the MEs in these files are also analyzed. The number of strokes in these MEs varies between 10 and 54.

Moreover, some relationship between the number of strokes and processing time are detected. Table 4.3 shows this relationship. This relationship shows that when a ME has many strokes, it takes MathLet v3 to process more than one minute.

Number of Strokes	Processing Time
Less than 9	Less than 1.1 seconds
Less than 10	Less than 45 seconds
More than 31	More than 1 minute

Table 4.3: Initial stroke number-processing time relationship in MathLet v3

In addition to these measurements, the initial accuracy of MathLet v3 on these 921 trainig files is also evaluated. MathLet v3 can recognize 108 of 921 training files. In other words, 11.73% of the training files can be fully recognized by MathLet v3. This information will be compared to the evaluation results of MathLet v3 which has improvements in order to make sure that improvements made to increase time performance does not cause any loss in the accuracy of MathLet v3.

4.2.3 Time Performance Evaluations of MathLet v3 After Improvements

After parsing step is extended as detailed in Section 3.2, the same measurements are made on MathLet v3. The measurement results are given in Table 4.4.

Processing Time	Number of Files	Proportion
Less than 1 second	518	56.24%
Less than 10 seconds	628	68.19%
Less than 30 seconds	680	73.83%
Between 30 seconds and 5 minutes	76	8.25%
More than 5 minutes	165	17.92%

Table 4.4: The improved time performance evaluation results of MathLet v3

There are 165 files which require more than 5 minutes to be processed by MathLet v3. The number of strokes included by the MEs in these files is also analyzed. The number of strokes varies between 13 and 54. Table 4.5 shows the relationship between the number of strokes included in MEs and the processing time of them obtained on the improved MathLet v3.

Number of Strokes	Processing Time
Less than 9	Less than 0.8 seconds
Less than 10	Less than 7.9 seconds
More than 40	More than 5 minutes

Table 4.5: Improved stroke number-processing time relationship in MathLet v3

Finally, the accuracy of MathLet v3 is evaluated. There are 142 exact matches

among 921 training files. That is to say that, MathLet v3 can fully recognize 15.42% of the training files after improvements.

4.2.4 Comparison of Measurement Results

Table 4.2 and Table 4.4 show that the time performance of MathLet v3 has been improved. The proportion of the files which take more than 5 minutes to process is decreased from 30.62% to 17.92%. The proportion of the files which take less than 1 second to process is increased from 45.06% to 56.24%. MathLet v3 can process 73.83% of the files within less than 30 seconds after improvements while initially it can process 61.45% of them.

A ME with 10 strokes may be processed within more than 5 minutes in initial measurements, while MathLet v3 can process it within less time after improvements. The minimum number of strokes for the files which take more than 5 minutes to process is increased from 10 to 13.

As seen from Table 4.3 and Table 4.5, the maximum processing time for MEs having less than 9 strokes is decreased from 1.1 seconds to 0.8 seconds after improvements. Furthermore, improvements provide saving 37 seconds' time for the MEs containing less than 10 strokes. Due to some idiosyncracies of the data, the MEs with 10 strokes seem to take much more time to be processed by both initial and improved MathLet v3 compared to the MEs with 9 strokes, even though such a discontinuity between these MEs would not be expected.

Finally, the expression-level recognition rate of MathLet v3 which is improved is greater than the recognition rate obtained in initial measurements. MathLet v3 can fully recognize 15.42% of 921 training files which are the MEs containing Part-II symbols and provided by CROHME organizers, while it can fully recognize 11.73% of them in the initial measurements. This fact provides that improvements on MathLet v3 do not have any negative effects on the expression-level recognition rate of it.

5 CROHME Competition

CROHME has been organized since 2011 with the goal of bringing the researchers who study on the area of the recognition of handwritten MEs under a common platform. In this platform, there is an opportunity for researchers to share the same dataset. Furthermore, researchers can report their performance on common test data. Researchers also find an opportunity to compare their work with other works and to identify the strengths and weaknesses of their systems compared to other benchmark systems. Competition also provides the documentation of challenges and advancements in the area of handwritten ME recognition. Each CROHME has been organized along with an international conference. CROHME 2011 [20], CROHME 2012 [21] and CROHME 2013 were held at ICDAR 2011 [22], ICFHR 2012 [23] and ICDAR 2013 respectively.

The organizers of CROHME 2011 and 2012 were from three different universities in three different countries. First group of organizers was from University of Nantes, France. The name of organizers from this university are Harold Mouchère and Christian Viard-Gaudin. Second group consisted of Dae Hwan Kim and Jin Hyung Kim from KAIST, Republic of Korea. The third organizer was Utpal Garain from Indian Statistical Institute, India. In addition to these organizers, CROHME 2013 has had one more organizer who has been Richard Zanibbi from Rochester Institute of Technology, NY, USA.

The number of participants in CROHME 2011 was four. In addition to four systems which were developed by participants, there was also one more system developed by one of the organizing groups. Participants were from universities in USA, Turkey, Spain and Greece. The system which was developed by organizer was from University of Nantes, France. The number of participants was increased in CROHME 2012. There were six participant systems in CROHME 2012. Similar to CROHME 2011, there was a system from one of the organizers in CROHME

2012. Research groups which had participated in CROHME 2011 also participated in CROHME 2012. In other words, in CROHME 2012 there were two newcomer systems. One of the newcomer systems was from University of Waterloo, Canada, while the other one was the commercial system developed by Vision Objects.

CROHME organizers provided a package for participants. Training data which are InkML files were provided for CROHME 2011 and 2012. Training data consists of MEs which contain symbols and relationships which are appropriate for defined grammar, i.e., Part-I. Training and test datasets were different from each other in each CROHME contest. Organizers also provided evaluation tool for participants in order to enable the participants for evaluating their systems before submission. For CROHME 2013, organizers have provided a wider package. The package consists of papers which are written about previous CROHME contests, training and test data of CROHME 2011 and CROHME 2012. Organizers have provided a tool which can be used to view MEs included by InkML files. A view of one example ME “ $T \int \Delta dl$ ” which is included in an InkML file provided by CROHME organizers as a training data is shown in 5.1. In addition to these, they have provided output result files of all participants in CROHME 2011 and 2012.



Figure 5.1: A view of the ME “ $T \int \Delta dl$ ” included in Part-IV training files

5.1 Data Format

In CROHME, participating systems take an input file which contains information on handwritten ME to be recognized. Then, the recognition result of the participating system for the input file is returned in an output file. In CROHME, both files are in InkML [24] format which is the data format designed to represent ink data. InkML also provides the interchange of ink data between different applications which run across different devices and platforms.

An InkML file consists of three information about ink data. First of these information are the coordinates of points that form a trace or in other words stroke. A set of strokes then forms the symbol. Hence, first information included in InkML file is stroke-level. Second information included in InkML file is symbol-level. It provides information on the segmentation and the label of the symbols included in handwritten ME. The third information provided by InkML file is the MathML code of ME and this information is expression-level.

Some further information may also be annotated in InkML files such as the gender, age and handedness of the writer, the channels such as X , Y , T , the ground truth \LaTeX code of ME. An example of an InkML file for the ME " $a+c = b$ " is shown in Table 5.1. In the InkML file shown in Table 5.1, there are some information about writer. The writer of that ME is 26 years old and right-handed male. Identification codes for the ME and writer are also given. Channels are identified as X and Y together with their types which are decimal.

In the example InkML file, there are 7 strokes for 5 symbols. The symbols "+" and "=" have two strokes while the others have one stroke. This information can be extracted from segmentation information given at the end of InkML file. Here, the "traceGroup" element with identifier `xml:id="9"` references to two strokes by their ids which are "1" and "2". The same "traceGroup" element has also a reference to the element in MathML which has id "+_1". This reference links those two strokes to the symbol "+". The view of the ME " $a + c = b$ " which is viewed by the tool provided by CROHME organizers is shown in Figure 5.2. In this figure, the strokes, the segmentation of strokes and points which form strokes can be seen clearly. It should be noted that these information are extracted from InkML file given in Table 5.1. This InkML file was also provided by CROHME organizers as a training data.

MathML [25] is the name of the standard developed for better understanding of the representation of MEs and the content of them. MathML is used to encode the structure of MEs. MathML also provides displaying, manipulating and sharing the MEs over the World Wide Web [26]. MathML consists of two markups; presentation markup and content markup. First one deals with the appearance of MEs while the latter deals with the mathematical meaning of them. For instance, content markup gives the mathematical content of the ME " c multiplied by b ", while presentation

markup defines that it is presented as “ cb ”, “ $c \times b$ ” or “ $c \cdot b$ ”.

```

<ink xmlns="http://www.w3.org/2003/InkML" >
  <traceFormat>
    <channel name="X" type="decimal" />
    <channel name="Y" type="decimal" />
  </traceFormat>
  <annotation type="UI" >2011_IVC_DEPART_F004_E023</annotation>
  <annotation type="writer" >depart004</annotation>
  <annotation type="truth" >$a+c=b$</annotation>
  <annotation type="age" >26</annotation>
  <annotation type="gender" >M</annotation>
  <annotation type="hand" >R</annotation>
  <annotationXML type="truth" encoding="Content-MathML" >
    <math xmlns="http://www.w3.org/1998/Math/MathML" >
      <mrow>
        <mi xml:id="a_1" >a</mi>
        <mrow>
          <mo xml:id="+_1" >+</mo>
          <mrow>
            <mi xml:id="c_1" >c</mi>
            <mrow>
              <mo xml:id="=_1" >=</mo>
              <mi xml:id="b_1" >b</mi>
            </mrow>
          </mrow>
        </mrow>
      </math>
    </annotationXML>
  <trace id="0" >9.68215 25.7205, ... , 9.78647 25.9974</trace>
  ...
  <trace id="6" >12.19 25.9934, ... , 12.852 25.7847</trace>
  <traceGroup xml:id="7" >
    <annotation type="truth" >Segmentation</annotation>
    <traceGroup xml:id="8" >...</traceGroup>
    <traceGroup xml:id="9" >
      <annotation type="truth" >+</annotation>
      <traceView traceDataRef="1" />
      <traceView traceDataRef="2" />
      <annotationXML href="+_1" />
    </traceGroup>
    ...
  </traceGroup>
</ink>

```

Table 5.1: An example of an InkML file for the ME “ $a + c = b$ ”

The MathML code of the ME “ $a + c = b$ ” which is included in InkML file shown

in Table 5.1 is an example of presentation markup in MathML. Presentation markup is generally used by web browsers or graphics packages for the purpose of displaying MEs. Presentation markup can also be defined as the XML equivalent of $\text{T}_{\text{E}}\text{X}$ math [27]. Presentation markup consists of elements each representing a syntactic structure in ME. For instance, “mi” element is used to represent variables, function names or symbolic constants and “mo” element is used for representing operator symbols. Other examples for elements used in presentation markup are “mfrac” element which is used for fractions, “msqrt” element which is used for square roots and “mrow” element which is used to horizontally group subexpressions. There are about 30 elements in presentation markup in MathML. In CROHME, MEs included in input and output files are encoded in presentation markup in MathML.

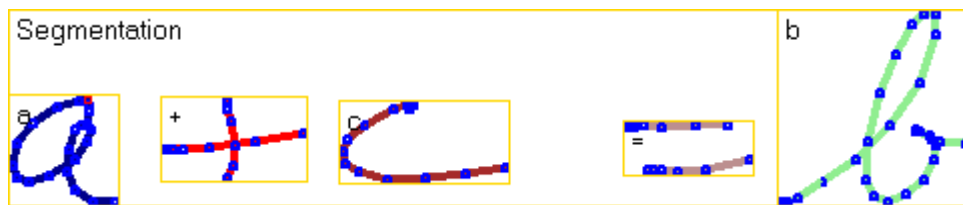


Figure 5.2: A view of the ME “ $a + c = b$ ” included in a Part-I training InkML file

The representation of the ME “ $a + c = b$ ” in content markup in MathML is given in Table 5.2. Mathematical processing packages and documents mostly use content markup. Content markup in MathML generally consists of elements which encode an expression tree. Examples of elements used in content markup in MathML are “ci” element which is used to represent variables and “apply” element which is used to apply operators or functions. Further examples for elements of content markup are “plus”, “minus” and “times” elements which are used for addition, subtraction and multiplication respectively and “root” element which is used to extract roots. There are about 120 elements in content markup in MathML.

5.2 Task and Evaluation Metrics

The task of recognition differs in each CROHME contest. In CROHME 2011, there were two different symbol sets. These sets were named Part-I and Part-II. Part-II had more number of symbols than Part-I. Part-II had 57 symbols while Part-I had 37 symbols. The ME shown in Figure 5.2 is an example of Part-I MEs, while

Figure 5.3 shows the view of ME “ $\int(2^x - 3e^x)dx$ ” which is an example of the MEs included in Part-II. In addition to these symbol sets, in CROHME 2012 there was one more symbol set named Part-III which had 75 symbols. Moreover, there has been fourth symbol set which has been named Part-IV in CROHME 2013. Part-IV has consisted of 102 symbols. Example MEs for Part-III MEs and Part-IV MEs are shown in Figure 5.4 and Figure 5.1 respectively. Each part contains all of the symbols which were included in the previous parts. For instance, 57 of 75 symbols of Part-III were the symbols which had been included in Part-II. In addition to the differences in symbol sets, each part had also some differences according to the logical relationships which they could contain. For instance, in Part-I fraction of fractions such as “ $\frac{1}{2}$ ” is not allowed.

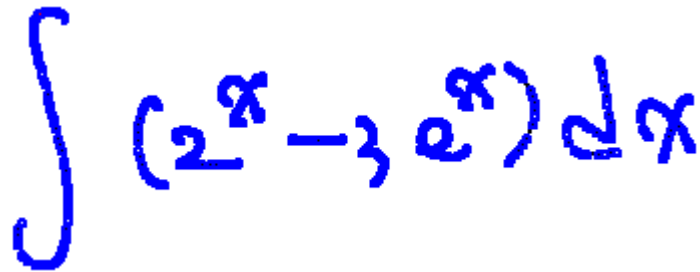
<apply>
<eq/>
<apply>
<plus/>
<ci>a</ci>
<ci>c</ci>
</apply>
<ci>b</ci>
</apply>

Table 5.2: The content MathML code of the ME “ $a + c = b$ ”

In CROHME 2011, organizers evaluated the participating systems in four aspects, stroke-level classification rate, symbol segmentation rate, symbol recognition rate and expression-level recognition rate. Stroke-level classification rate, ST_{rec} , is used to show the percentage of strokes with the correct symbol. Symbol segmentation rate, SYM_{seg} , defines the percentage of symbol which are segmented correctly. The symbol recognition rate, SYM_{rec} , computes the performance of symbol classifier when considering only the correct segmented symbols. Expression-level recognition rate, EXP_{rec} , shows the percentage of MEs which are totally recognized. This measure is very harsh so that the tiniest mistake in ME causes a decrease in the expression-level recognition rate.

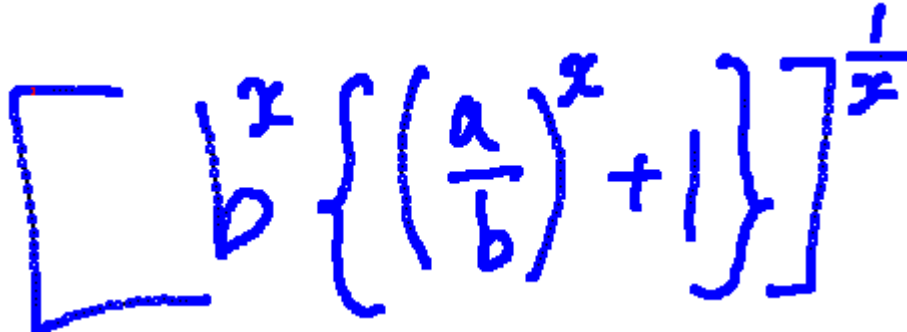
In CROHME 2012, there was one more aspect which was MathML structure recognition rate, $STRUCT$, and shows the percentage of MEs which have correct

MathML structure without considering the label of leaves. For instance, two MEs “ $a_2 + 5$ ” and “ $b_k + c$ ” have the same MathML structure. In CROHME 2012 and CROHME 2013, organizers has also evaluated the expression-level recognition rates of systems with having at most one, two and three errors. These rates have been denoted as EXP_{rec-1} , EXP_{rec-2} and EXP_{rec-3} respectively. These errors may be in symbol label or MathML node tag. In each contest as a final rating of the participating systems expression-level recognition rates are used and the system developed by one of the organizing groups does not compete with other systems.



A handwritten mathematical expression in blue ink, showing the integral of (2^x - 3e^x) with respect to x. The integral symbol is on the left, followed by the expression (2^x - 3e^x) in parentheses, and dx at the end.

Figure 5.3: A view of the ME “ $\int(2^x - 3e^x)dx$ ” included in Part-II training files



A handwritten mathematical expression in blue ink, showing the expression [b^x {(a/b)^x + 1}] raised to the power of 1/x. The expression is enclosed in large square brackets, with a superscript 1/x to the right.

Figure 5.4: A view of the ME “ $[b^x\{(\frac{a}{b})^x + 1\}]^{\frac{1}{x}}$ ” included in Part-III training files

5.3 Evaluation Results of MathLet

MathLet v2 participated in CROHME 2011 with two different versions which were developed for specifically Part-I and Part-II. Table 5.3 shows the evaluation results of MathLet v2 obtained in CROHME 2011.

MathLet v2 took the fourth place among five participants for both Part-I and Part-II grammars in CROHME 2011. The reason of this low evaluation result was the fact that for some MEs, MathLet v2 generated outputs with wrong MathML

structure while it correctly recognized them.

Grammar	ST_{rec}	SYM_{seg}	SYM_{rec}	EXP_{rec}
Part-I	22.39%	27.98%	82.11%	0.55%
Part-II	22.11%	28.25%	83.76%	0.29%

Table 5.3: The evaluation results of MathLet v2 in CROHME 2011

In CROHME 2012, there were three grammars Part-I, Part-II and Part-III. Organizers tested the systems with different datasets from the datasets in CROHME 2011. In addition to the results evaluated with these new datasets, organizers also provided expression-level recognition rate of the systems on the test dataset used in CROHME 2011. Table 5.4 shows the expression-level recognition rate of MathLet v3 in CROHME 2012 on the test dataset of CROHME 2011 together with the results obtained in CROHME 2011. Evaluation results of MathLet v3 on new datasets can be found in Table 5.5. Furthermore, MathLet v3’s expression-level recognition rates with the errors on Part-III dataset can be found in Table 5.6. MathLet v3 took the sixth place among seven participants in CROHME 2012.

Grammar	MathLet v2	MathLet v3
Part-I	0.55%	30.94 %
Part-II	0.29%	18.68 %

Table 5.4: The expression-level recognition rates of MathLet v3 in CROHME 2012 with the test dataset of CROHME 2011

Grammar	ST_{rec}	SYM_{seg}	SYM_{rec}	$STRUCT$	EXP_{rec}
Part-I	61.33%	72.11%	87.76%	37.04%	22.22%
Part-II	49.06%	61.09%	88.36%	17.61%	7.97%
Part-III	45.42%	59.20%	84.27%	14.75%	4.92%

Table 5.5: The evaluation results of MathLet v3 in CROHME 2012

Grammar	EXP_{rec}	EXP_{rec-1}	EXP_{rec-2}	EXP_{rec-3}
Part-III	4.92%	10.66%	14.14%	14.96%

Table 5.6: MathLet v3’s expression recognition rates with errors in CROHME 2012

Expression-level recognition rates of MathLet v3 in CROHME 2013 can be seen from Table 5.7. In CROHME 2013, MathLet v3 has taken the fifth place among six participants.

Grammar	EXP_{rec}	EXP_{rec-1}	EXP_{rec-2}	EXP_{rec-3}
Part-IV	8.35%	19.08%	24.44%	26.23%

Table 5.7: MathLet v3’s expression recognition rates with errors in CROHME 2013

5.4 CROHME Evaluation Results

The evaluation results obtained in CROHME 2011 are given in Table 5.8. System-I was developed at Rochester Institute of Technology in United States, System-II was MathLet v2, System-III was developed at Universitat Politècnica de València in Spain, System-IV was Math-ILSP system and developed at Institute for Language and Speech Processing, Athena Research Center in Greece and the System-V was developed at Université de Nantes in France.

System	EXP_{rec} - Part I	EXP_{rec} - Part II
System-I	4.42%	2.59%
Sytem-II	0.55%	0.29%
System-III	29.28%	19.83%
System-IV	0.00%	0.00%
System-V	40.88%	22.41%

Table 5.8: The evaluation results of CROHME 2011

Table 5.9 shows the evaluation results obtained in CROHME 2012. System-I was developed at Universitat Politècnica de València in Spain, System-II was

Math-ILSP system and developed at Institute for Language and Speech Processing, Athena Research Center in Greece, System-III was developed at Université de Nantes in France, System-IV was developed at Rochester Institute of Technology in United States, System-V was MathLet v3, System-VI was Waterloo recognizer which was developed at University of Waterloo in Canada, System-VII was a commercial system developed by Vision Objects. First five systems had also participated in CROHME 2011.

System	EXP_{rec} - Part I	EXP_{rec} - Part II	EXP_{rec} - Part III
System-I	35.19%	33.89%	22.75%
System-II	8.33%	6.64%	3.69%
System-III	57.41%	38.87%	25.61%
System-IV	28.70%	14.29%	9.43%
System-V	22.22%	7.97%	4.92%
System-VI	51.85%	49.17%	40.16%
System-VII	81.48%	75.08%	62.50%

Table 5.9: The evaluation results of CROHME 2012

The evaluation results obtained in CROHME 2013 are given in Table 5.10.

System	EXP_{rec} - Part IV
System-I	60.36%
System-II	23.40%
System-III	19.97%
System-IV	9.39%
System-V	8.35%
System-VI	2.68%

Table 5.10: The evaluation results of CROHME 2013

6 Conclusions

In this thesis, the improvements on MathLet v2 which is the software developed to recognize handwritten MEs are presented. The improved system presented in this thesis is called as MathLet v3.

The contributions of this thesis are as follows:

- Symbol recognition in MathLet v3 is done by combining online and offline classifiers each returning prediction probability over classes.
- The accuracy of MathLet is increased from 0.55% to 8.35% according to results obtained in CROHME while the recognition task becomes more difficult. In addition to symbol recognition which is introduced above, also parsing process plays a key role in this increase. Parsing process of MathLet v3 has special handling method for mistaken symbols. Moreover, MathML parsing provides a huge gain in the accuracy of MathLet v3.
- The time performance of MathLet v3 is measured and also improved by using new symbol recognition and parsing approaches introduced above. In addition to these, parallel programming which is implemented in the parsing process of MathLet v3 also provides an increase in the time performance of MathLet v3.
- MathLet v3 has four applications and two of them can be accessed through the Web.

MathLet v3 uses two classifiers for the purpose of symbol recognition. One of these two classifiers uses offline features, while the other one uses online features of training symbol. Offline features are extracted from the image of training symbol. Online features are extracted from resampled training symbol after scaling operation. The symbol is resampled such that it contains 20 equidistant points, then it is scaled and features are extracted as the difference between x and y coordinates of

consecutive points of scaled resampled symbol. Hence, 38 delta features are used as online features. MathLet v3 combines online and offline classifiers according to the prediction probability of the most probable symbol returned by these classifiers. These prediction probabilities are compared and the result returned by the classifier which returns higher prediction probability for the most probable symbol is chosen as the result of symbol recognition.

The prediction probability of recognized symbol is used in parsing mistaken symbols such as the symbols “1”-“|” and the symbols “+”-“t”. If one of the mistaken symbols is recognized, the other symbol is added as an alternative if the prediction probability is low. Moreover, in order to distinguish the symbols “ \times ” and “ x ”, contextual checks are involved. Parallel programming is also used in parsing phase. These improvements on parsing step provide an increase in the time performance of MathLet v3. The time performance of MathLet v3 is measured by the time which parsing phase takes Mathlet v3 to process. The initial time performance evaluations of MathLet v3 show that 30.62% of MEs cannot be processed within 5 minutes and 56.89% of MEs can be processed within less than 10 seconds. Improved MathLet v3 can process 68.19% of MEs within less than 10 seconds. Furthermore, the proportion of MEs which MathLet v3 cannot process within 5 minutes is decreased from 30.62% to 17.92%.

MathLet v2 obtained low accuracy rates in CROHME 2011. The reason of this fact was that MathLet v2 generated wrong MathML codes for some MEs included in output InkML files, while it correctly recognized these MEs. In order to generate correct MathML codes for the recognized MEs, additional MathML parsing is implemented in MathLet v3. This additional implementation along with symbol recognition and parsing steps described above provides a huge gain in the accuracy of MathLet v3. MathLet v3 obtains 8.35% in expression level recognition rate in CROHME 2013 while MathLet v2 obtained 0.55% in CROHME 2011. It should also be noted that the recognition task in CROHME 2013 is more difficult than it was in CROHME 2011.

MathLet v3 has four applications. MathLet v3’s console and Windows application also exist in MathLet v2. Other two applications of MathLet v3 are accessed through the Web. One of these applications provides that a user can upload an input

InkML file and get top-5 recognition results as an output. Other application provides that a user writes his/her own ME on the Web page and get top-5 recognition results for it.

Despite of the fact that MathLet v3 has contributions, some future work can still be done. The number of mathematical symbols that can be recognized by MathLet should be increased. The mathematical symbols that belong to set theory such as \subset , \subseteq , \cap , \cup should be recognized in the next versions of MathLet. In addition to these symbols, the symbols which belong to propositional logic such as \wedge , \vee , \oplus may be added to the list of recognized symbols of MathLet's next versions. Moreover, the recognition of matrices should also be provided.

As it can be seen from CROHME results, symbol segmentation in MathLet needs to be improved. The improvements in symbol segmentation will provide an improved symbol recognition. Hence, the expression-level accuracy of MathLet will be improved too.

The interface of Windows application and Web application of MathLet may be modified for better human-computer interaction. Web interface also needs a modification to be viewed in all web browsers. With the increase in the popularity of mobile devices and tablet computers, the development of a mobile application for MathLet may be one of the future work.

7 References

- [1] Kam-Fai Chan and Dit-Yan Yeung. Mathematical expression recognition: a survey. *IJDAR*, 3(1):3–15, 2000.
- [2] Francisco Alvaro, Joan-Andreu Sánchez, and José-Miguel Benedí. Recognition of On-line Handwritten Mathematical Expressions Using 2D Stochastic Context-Free Grammars and Hidden Markov Models. *Pattern Recognition Letters*, 2012. <http://dx.doi.org/10.1016/j.patrec.2012.09.023>.
- [3] Scott MacLean and George Labahn. A new approach for recognizing handwritten mathematics using relational grammars and fuzzy sets. *IJDAR*, 16(2):139–163, 2013.
- [4] Vision Objects. <http://www.visionobjects.com/>.
- [5] Mehmet Çelik. Handwritten mathematical expression recognition using graph grammars. Master’s thesis, Sabanci University, 2010.
- [6] Masakazu Suzuki, Fumikazu Tamari, Ryoji Fukuda, Seiichi Uchida, and Toshihiro Kanahori. INF^{TY}: an integrated OCR system for mathematical documents. In *ACM Symposium on Document Engineering*, pages 95–104. ACM, 2003.
- [7] Francisco Alvaro, Joan-Andreu Sánchez, and José-Miguel Benedí. Recognition of Printed Mathematical Expressions Using Two-Dimensional Stochastic Context-Free Grammars. In *ICDAR* [22], pages 1225–1229.
- [8] Lei Hu, Kevin Hart, Richard Pospesil, and Richard Zanibbi. Baseline extraction-driven Parsing of handwritten mathematical expressions. In *ICPR*, pages 326–330. IEEE, 2012.

- [9] George Labahn, Edward Lank, Scott MacLean, Mirette S. Marzouk, and David Tausky. MathBrush: A System for Doing Math on Pen-Based Devices. In Koichi Kise and Hiroshi Sako, editors, *Document Analysis Systems*, pages 599–606. IEEE Computer Society, 2008.
- [10] Lei Hu and Richard Zanibbi. HMM-Based Recognition of Online Handwritten Mathematical Symbols Using Segmental K-Means Initialization and a Modified Pen-Up/Down Feature. In *ICDAR* [22], pages 457–462.
- [11] Christopher Malon, Seiichi Uchida, and Masakazu Suzuki. Mathematical symbol recognition with support vector machines. *Pattern Recognition Letters*, 29(9):1326–1332, 2008.
- [12] Birendra Keshari and Stephen M. Watt. Hybrid Mathematical Symbol Recognition Using Support Vector Machines. In *ICDAR*, pages 859–863. IEEE Computer Society, 2007.
- [13] Arit Thammano and Sukhumal Rugkunchon. A Neural Network Model for Online Handwritten Mathematical Symbol Recognition. In De-Shuang Huang, Kang Li, and George W. Irwin, editors, *ICIC (1)*, volume 4113 of *Lecture Notes in Computer Science*, pages 292–298. Springer, 2006.
- [14] Ahmad-Montaser Awal, Harold Mouchère, and Christian Viard-Gaudin. Towards Handwritten Mathematical Expression Recognition. In *ICDAR*, pages 1046–1050. IEEE Computer Society, 2009.
- [15] Hakan Büyükbayrak. Online handwritten mathematical expression recognition. Master’s thesis, Sabanci University, 2005.
- [16] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27, 2011. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [17] Julia Lerman. Inking in ASP.NET 2.0, AJAX, and IE7. *CoDe Focus Magazine*, 4(2):57–65, 2007.
- [18] MSDN Forums. Uninstallation file in user’s menu. <http://social.msdn.microsoft.com/Forums/vstudio/en-US/ed4d3444-4634-40c1-bb8a->

- 19437097bb88/uninstallation-file-in-users-menu. Accessed: 2013-07-30.
- [19] MSDN. How to: Create or Add a Setup Project. [http://msdn.microsoft.com/en-us/library/vstudio/19x10e5c\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/vstudio/19x10e5c(v=vs.100).aspx). Accessed: 2013-07-30.
- [20] Harold Mouchère, Christian Viard-Gaudin, Dae Hwan Kim, Jin Hyung Kim, and Utpal Garain. CROHME2011: Competition on Recognition of Online Handwritten Mathematical Expressions. In *ICDAR* [22], pages 1497–1500.
- [21] Harold Mouchère, Christian Viard-Gaudin, Dae Hwan Kim, Jin Hyung Kim, and Utpal Garain. ICFHR 2012 Competition on Recognition of On-Line Mathematical Expressions (CROHME 2012). In *ICFHR*, pages 811–816, 2012.
- [22] *2011 International Conference on Document Analysis and Recognition, ICDAR 2011, Beijing, China, September 18-21, 2011*. IEEE, 2011.
- [23] *2012 International Conference on Frontiers in Handwriting Recognition, ICFHR 2012, Bari, Italy, September 18-20, 2012*. IEEE, 2012.
- [24] Ink Markup Language (InkML). <http://www.w3.org/TR/InkML/>. Accessed: 2013-08-01.
- [25] Mathematical Markup Language (MathML) Version 3.0. <http://www.w3.org/TR/MathML3/>. Accessed: 2013-07-29.
- [26] Robert Miner and Jeff Schaeffer. A Gentle Introduction to MathML. <http://www.dessci.com/en/reference/mathml/>. Accessed: 2013-07-30.
- [27] Hussein Shafie. MathML Presentation Markup for the Impatient. <http://www.xmlmind.com/tutorials/MathML/index.html>, November 11, 2012. Accessed: 2013-07-30.