CS490y Final Report
Organizing the Structure of Mathematical Expressions

Clare M. So
Student Number: 250017188
Department of Computer Science
University of Western Ontario
London, Ontario, CANADA, N6A 5B7

clare@scl.csd.uwo.ca

March 27, 2003

[1]This thesis is supervised by Dr. Stephen M. Watt.

Abstract

This document describes the result of this project that concerns with organizing the structure of mathematical expressions. Chapter 1 presents the motivation, the goal of this project, and some previous works in structural analysis of mathematical expressions. Chapter 2 is a brief review of mathematical expressions' properties. Chapter 3 outlines the requirements of the input files and the method to generate input files. Chapter 4 documents the design of the software. Chapter 5 describes the method to generate Presentation MathML. Chapter 6 concludes the report by suggesting some possible future works and presenting the experience gained from this project.

Keywords: MathML, Mathematical Handwriting Recognition

Acknowledgments

First of all, I would like to thank my supervisor, Dr. Stephen Watt, for his guidance and patience. Many thanks go to the system administrators of the ORCCA lab, Mr. Igor Rodionov and Mr. Laurentiu Dragan, for setting up and maintaining the hardware and the software of my computer in the lab. My thanks also go to Ms. Bethany Heinrichs and the members of the ORCCA lab for their support in various forms. Without any of them, this thesis would not be possible.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Inputting mathematics into computers is not a trivial task. First, the structure of mathematical expressions is two-dimensional, but the computer keyboards only support linear inputs. For example, the range of integration is indicated by two numbers in the upper right and lower right corners of the integral sign in $\int_a^b (x+3)^2 dx$. It is impossible to enter such expressions without using special tools. Second, the spatial relationships between different symbols are arbitrary. These relationships are intuitive to mathematicians, but they are not intuitive to computers. Computers' input devices, such as handwriting recognizers and scanners, must be taught to recognize individual symbols as well as the two-dimensional relationships in order for them to understand mathematics. For example, dx in the integral example mentioned previously should not be interpreted as "d times x". Instead, these two letter together should be understood as "integrate with respect to x because the integral sign is present. Third, mathematical expressions often use the symbols that are not available in the ordinary computer keyboards. Without special tools or look-up tables, it is impossible to enter symbols such as the integral sign ("$\int$") and the Greek letters.

There exist methods that allow one to enter mathematics, but these methods are troublesome to use. TEX is the most popular method to typeset mathematics in the scientific community. MathML [1] is an emerging recommendation to put both of the presentation and the content of mathematics on the World Wide Web. Computer Algebra Systems, such as *Maple*, define their own syntax for entering mathematics. Unfortunately, these methods require a considerable amount of learning and practice to use. Templates in Computer Algebra Systems are easier to use than the methods mentioned, but these templates take up screen space and limit the set of mathematics that can be entered (Figure 1.1).

Handwriting is the most natural way to enter mathematics. However, the handwriting recognizers nowadays cannot recognize mathematics because the recognizers are built to treat linear input and pictures only. In the symbol recognition level, mathematics can be treated like text. But in the structural level, mathematics cannot be treated like text because they are two-dimensional. The recognizers often treat mathematics as pictures,

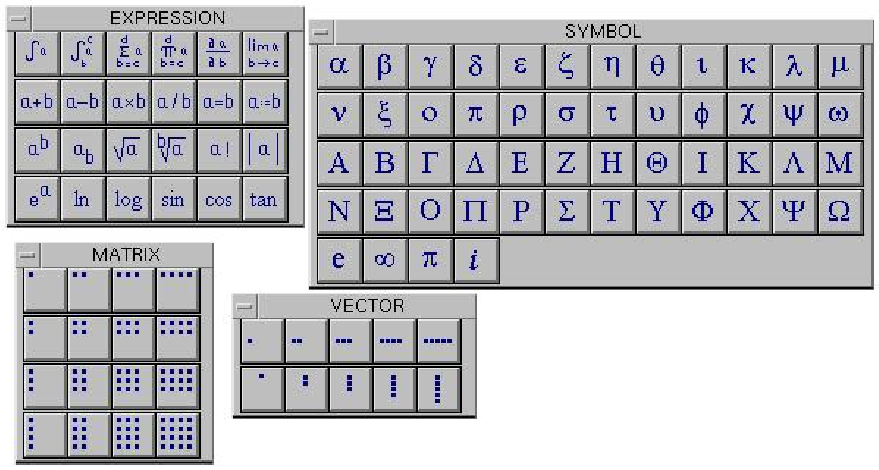

Figure 1.1: *Palettes* in Maple 8 helps users to enter mathematics by providing a predefined set of most frequently used characters and patterns.

therefore the semantics of mathematical expressions are lost. To make the handwriting devices such as the Tablet PCs to be used to enter mathematics, it is necessary to improve the existing handwriting recognizers so that mathematics can be treated properly.

Many mathematical expressions are not typeset properly. These expressions, which are typeset in TEX (and possibly in Presentation MathML in the future), concern with how the expressions look on paper only and do not carry the semantic meaning and symbols' groupings of the expression. For example, the integral example can be typeset as if the closing bracket is being squared like this in TEX:

```
$ \int_{a}^{b}\! {(x+3} {)}^{2} {dx} $
```

Groupings are indicated by "{" and "}" in TEX. "{)}^{2}" indicates that the closing bracket is being squared. Obviously, one cannot square a bracket. Instead, the content of the brackets, $x + 3$, is being squared.

1.2 Goal of the Project

This project's goal is to gain appreciation and experience of the complexities that involve in recognizing the two-dimensional relationships between the symbols and the characters in **single-line** handwritten or typeset mathematical expressions. Symbol recognition problems are not considered in this project.

Upon the completion of this project, rules may be defined for allowed two-dimensional relationships of different symbols and characters. Wan [4] developed a mathematical handwriting recognizer for the Pocket PC. This recognizer recognizes the characters, determines the two-dimensional relationships between the characters, builds an internal syntax tree, and generates Presentation MathML output. It can only handle basic mathematics. With the rules, handwriting recognizers can be extended so that they can handle a larger set of mathematics.

1.3 Previous Works

Lavirotte [3] defined graph grammars in his OFR (Optical Formula Recognition) for recognizing typeset mathematics in documents. This project focuses on getting the semantics of mathematics. The individual characters are nodes of the graph. The links join the nodes that are related to each other. The grammars define the legal two-dimensional relationship between the characters. These relationships between characters are *include*, *superscript*, *right*, *subscript*, *below*, *presubscript*, *left*, *presuperscript*, and *above*. The properties, the size, and the position of the characters are used to determine the relationships. Different kinds of characters do not allow certain kinds of relationships. For example, prescripts are rare so that only a few symbols allow such relationship.

Wan [4] discussed strategies and approaches for building the Mathematical Handwriting Recognizer. The spatial relationships are defined similarly to Lavirotte's work. Unlike Lavirotte's work, Wan's work focuses on the presentation of mathematics as MathML becomes an emerging recommendation. While focusing on the presentation of mathematics, implicit semantics are preserved. Wan's work also extends from recognizing typeset mathematics to recognizing a limited set of handwritten mathematics.

Further discussions of these two works are presented throughout the thesis.

Chapter 2

Properties of Mathematical Expressions

This chapter explains the complexities of treating mathematical expressions in computers. First, the difference between presentation and content of mathematical expressions is explained. By having some knowledge of mathematics, one can see that presentation of mathematical expressions is two-dimensional, arbitrary, semi-standardized, and potentially ambiguous.

2.1 Presentation and Content of Mathematics

Two aspects of mathematical expressions' representation are presentation and content. TeX and Presentation MathML focus on the presentation of mathematics. Input syntax of Computer Algebra Systems, Content MathML, and OpenMath [2] focus on the content of mathematics (figure 2.1). This project focuses on treating the *presentation* of mathematics while preserving mathematics' implicit semantics.

2.2 Two-Dimensional

Some symbols and operators are not used linearly in presentation of mathematics. This characteristic makes recognizing mathematical handwriting not a trivial task. The two-dimensional relationships between the characters can be divided in the following categories by both Lavirotte and Wan:

- *Subscript* (Example: x_y)
- *Superscript* (Example: x^y)
- *Underscript* (Example: $\underset{\cdot}{\mathrm{x}}$)
- *Overscript* (Example: $\dot{x}$)

```
                $ {x}_{}^{3} $

  <OMOBJ>
    <OMA>
      <OMS cd="tranc1" name="exp"/>
      <OMV name="x"/>
      <OMI> 3 </OMI>
    </OMA>
  </OMOBJ>

<math xmlns="http://www.w3.org/MathML">
  <apply>
    <exp/>
    <ci> x </ci>
    <cn> 3 </cn>
  </apply>
</math>

<math xmlns="http://www.w3.org/MathML">
  <mrow>
    <msup>
      <mi> x </mi>
      <mn> 3 </mn>
    </msup>
  </mrow>
</math>
```

Figure 2.1: “x^3” written in TEX, OpenMath, Content MathML, and Presentation MathML.

- *Presuperscript* (Example: ${}^{y}F$)
- *Presubscript* (Example: ${}_{y}F$)
- *Inline* (Example: xy)
- *Include* (Example: $\sqrt{x}$)

Lavirotte [3] points out that certain spatial relationships between some symbols and operators are not allowed. For example, “$)^y$” alone is clearly not a valid mathematical expression, but “3^y” is. The randomness in the usage of symbols and operators is discussed in the next section.

2.3 Uses of Notations are Arbitrary

Mathematical operators can be categorized according to their usages. The usages of different operators are arbitrary. Lavirotte's classification of notation is similar to Wan's. A summary of interesting points in the classification schemes is as follows:

- *Linear*:
  - *Prefix* for unary negation etc. (Example: $-x$)
  - *Infix* for addition and subtraction etc. (Example: $1+2+3$)
  - *Postfix* for factorial (Example: $3!$)
  - *Bounding* for range of a polynomial (Example: $[a, b]$)
- *Vertical* for fraction (Example: $\frac{x}{y}$)
- *Implicit* for exponent and indexes (Example: x^2)
- *Two-Dimensional* for summation and matrix etc. (Example: $\sum_{x=1}^{10} x+y$ and $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$)
- *Include* for Square root (Example: $\sqrt{x}$)

Some operators can be further categorized according to the number of arguments that they take:

- *Unary* for unary negation and factorial etc.
- *Binary* for exponent
- *N-ary* for addition, subtraction, multiplication and division etc.

Some of the arguments of an operator may not be compulsory. For example, $\int (x+3)^2 dx$ is still perfectly legal even though the range of integration in is not specified. On the other hand, some of the arguments are compulsory. For example, $\sum^{10} x+1$ is not legal because the bound variable and the lower bound are not specified.

Some of the operators can be used recursively. This means that the argument(s) of the operators can be anything including an expression using the operator the same as itself. For example, the argument of the exponent can be any valid expressions including another number that is the power of another number like $x^{x^{x^x}}$.

Precedence exists among operators. One of the precedence rules is that multiplication has higher precedence than addition. Knowing precedence rules is important to group variables, symbols, and characters properly because the groupings preserve implicit semantics.

The meanings of a symbol must be determined globally. For example, a dot on the baseline between two digits ("3.5") is a decimal point. On the other hand, three dots together (“$\ldots$”) is an ellipse. Furthermore, a dot that is not on the baseline between two digits (“$3 \cdot 5$”) means “times”.

Although the operators indicating equality ($=$, $<$ and $>$ etc) are used as if they are infix, n-ary operators, expressions containing this sign should belong to a special class. For example, the expression $(1 + 2 = 3)^2$ does not make any sense at all.

2.4 Variations and Ambiguities in Notations

Since the uses of notations are arbitrary, it is not surprising to see different conventions exist. For example, a decimal point is usually indicated by a dot (.) in English-speaking countries, but it is indicated by a comma (,) in French-speaking ones.

Both Lavirotte [3] and Wan [4] point out that mathematical expressions can be ambiguous. For example, there are two possible interpretations in $\sum_{x=1}^{10} x + 1$. The two interpretations result in different answers. One of the interpretations is $1+2+3\ \ldots+10\ +1$. Another interpretation is $(1 + 1) + (2 + 1) + (3 + 1) + \cdots + (10 + 1)$. The intended interpretation *may* be determined by the relative distance between characters. Parentheses do help to eliminate ambiguities, but they are not always used in handwritten or typeset mathematics when ambiguities present.

Chapter 3

Requirements of the Software

This chapter presents the requirement of the software the simulates an **offline** process of re-organizing the characters in the **single-line** mathematical expressions. The assumptions in the symbol recognition step is introduced. Then, the order of character in the input file and the definition of single-line mathematical expression are discussed. Finally, the test cases and the method to generate input files are presented.

3.1 Assumptions in Symbol Recognition

Upon recognition, every character is fitted into a rectangular *bounding box* that is similar to the one described in [4] and [3][1]. A *bounding box* records the location and the size of a character. The information of every *bounding box* is stored in a flat file consisting of the following information for every character in the same mathematical expression:

- Absolute values:
  - Character
  - Absolute reference point in baseline (x and y)
- Values relative to the reference point
  - Body baseline (y)
  - Body midline (y)
  - Body topline (y)
  - Lower left hand corner of the *bounding box* (y)
  - Upper right hander corner of the *bounding box* (x and y)

The body baseline is not necessary to be the same as the bottom of the *bounding box*. Similarly, the body topline is not necessary to be the same as the top of the *bounding box*. Every character has its own “main” body part and may have a “stick” going up or down.

[1]Lavirotte calls these boxes “les boîtes englobantes rectangulaires”

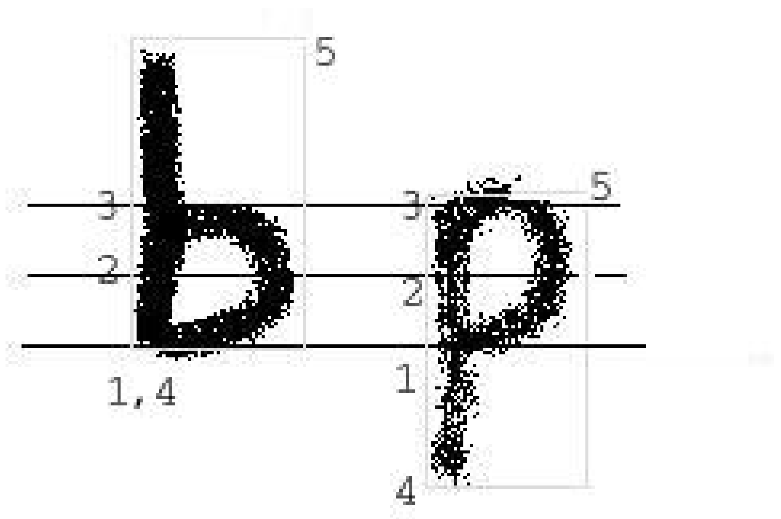

Figure 3.1: The grey rectangles denote the *bounding box*es. Information given by a *bounding box* are (1)Absolute reference point, (2)Body midline, (3)Body topline, (4)Lower left corner of the *bounding box*, (5)Upper right corner of the *bounding box*

The body topline is the same as the top of the *bounding box* and the body baseline is the same as the bottom of the *bounding box* of the characters such as “x”,”u”, and “v”. The bottom of the *bounding box* goes below the body baseline in characters such as “p” and “q”. Similarly, the top of the *bounding box* goes above the body topline in characters such as “b”, “l”, and “f”.

The body baseline, the body midline, and the body topline are used to determining superscript and subscript relationships. The superscripts and subscripts are usually written relative to the body. For example, one can recognize that “2” is the superscript of “b” in “b^2” even though the bottom of 2’s *bounding box* is not near the upper right corner of the b’s.

Some characters are broken down or grouped together for the purpose of this project. Wan [4] encountered a problem recognizing the “include” relationship in square roots. In this project, a square root is separated into two characters: the “check mark” part and the horizontal line. Some characters such as ellipse (“$\ldots$”) and plus-minus (“$\pm$”) can be further broken down into two or more characters. For the purpose of this project, every composite character is treated as one character instead of several characters.

The time that the character is recognized is implicitly expressed by the order of the characters in the input files.

3.2 Order of Characters

The characters in the input file must be in a specific order. The order of the characters is restricted to decrease the size of the problem. The characters are assumed to be written from left to right, from top to bottom. The first character must be one of the “main” characters. For example, the “$\vdash$”, the as and the xs are the “main” characters of $a_3x^3 + a_2x^2 + ax \vdash a_0$. The next character of any “main” characters can be (in the following order if applicable):

1. Presubscript
2. Presuperscript
3. Over
4. Superscript
5. Subscript
6. Inline

If the current character is not one of the “main” characters, the next character can be inline or the subscript of any ancestor characters. It is possible that the next character is the superscript or subscript of the current character as in $x^{x^{x^x}}$.

It is possible that the next character does not have any spatial relationship with the current one.

3.3 Single-Line Mathematical Expressions

For this project, only single-line mathematical expressions are considered. Single-line mathematical expressions do not use some two-dimensional constructs such as table, array, and fraction. In other words, these expressions’ height is the same as the surrounding handwriting or typeset text. For example, $|x| = \begin{cases} -x & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ x & \text{if } x > 0 \end{cases}$ and $\begin{bmatrix} x^2+1 & x-1 \\ x-1 & 1 \end{bmatrix}$ are obviously not single-line mathematical expressions.

In these expressions, all of the “main” characters lie on the row. These characters probably become the immediate row right under the root element. In other words, the characters are under the <mrow> which is the immediate child of <math>. The “main” characters can have superscripts, subscripts, and prescripts. The characters can be digits, alphabets, Greek letters, and symbols. Although notations involving “over” and “under” relationships such as $\sum_{x=1}^{10} x + y$ and recursive operators like $x^{x^{x^x}}$ may appear slightly taller than the surrounding text, they are still considered as single-line expressions.

Because the test cases only covers single-line expressions, only a restricted set of Presentation MathML are required (Figure 3.2).

Name	Description
mi	Identifier (example: Trigonometric functions, variables)
mn	Number (example: integer, read numbers)
mrow	Row (ie. characters sharing the same baseline or being in the same group)
msqrt	Square root
mfenced	Parenthesis
msub	Subscript
msup	Superscript
msubsup	Subscript and superscript
munder	Underscript
mover	Overscript
munderover	Overscript and underscript
mmultiscripts	Subscript, overscript, and prescript

Figure 3.2: Selected two-dimensional relationships handled by Presentation MathML

3.4 Test Cases

The test cases are selected to demonstrate different two-dimensional relationships and groupings of characters in single-line mathematical expressions:

1. $a_3x^3 + a_2x^2 + ax \vdash a_0$ for subscripts and superscripts
2. $\int_0^\infty \sin^{12} x + \cos^2 x \;\; dx$ for predefined character sequences (sin and cos etc.), and superscripts' groupings
3. $\sum_{i=0}^\infty i = 0 + 1 + 2 + 3 \ldots$ for large Greek characters, overscripts and underscripts
4. $b \pm \sqrt{b^2 + 4ac}$ for square root (Note: This is part of the Quadratic Equation)
5. $[a, b)$ for non-matching brackets
6. $x^{x^{x^x}} + 2x$ for recursive operators
7. $C = \{z | z = a + bi,\ a, b\ \in \Re\}$ for set notations
8. $x! + 10!$ for postfix operators
9. ${}^2He$ for presuperscripts
10. ${}_2F_1$ for presubscripts

3.5 Input File Generation

3.5.1 Using XML to Store Intermediate Data

A set of XML files are used to store the intermediate data of characters in every expression. Each character is assigned to an absolute reference position point in the baseline denoted by x and y coordinates. The magnification of every character is assigned. “1” is assigned as the magnification for every character that is a subscript, a superscript, or a subscript. “3” is assigned for every “main” character. “5” is assigned for big Greek characters.

If the character is an Unicode character, the entity name is used.

```
<glyph>
  <char>F</char>
  <posx>1</posx>
  <posy>1</posy>
  <mag>3</mag>
</glyph>
```

Figure 3.3: The datum for the character “F” in the 10th test case.

Then, the input data are generated by a XSLT stylesheet. This XSLT stylesheet calculates the relative values for every character using the absolute position and the magnification provided in the XML files. Width and height of the a character’s body is the magnification. The length of the sticks are assumed to be $\frac{3}{2}$ of the magnification. When it determines the position of the top and the bottom of the *bounding box* relative to the body topline and the body baseline, it takes into account whether a character is a descender (“p”, ”j”, and “y” etc.) or an ascender (“b”,“d”, and “t” etc.).

```
1 1 0 1.5 3 0 3 4.5 F
0 0 0 0.5 1 0 1 1.5 2
3 0 0 0.5 1 0 1 1.5 1
```

Figure 3.4: The “perfect” input data for the 10th test case. Each row denotes the information collected upon recognition of a character.

3.5.2 Noise Generation

Noise and inconsistency in handwriting present difficulties in mathematical handwriting recognition, so both “perfect” cases and “imperfect” cases of mathematics are needed. The “perfect” cases, which are presented in the previous section, are the ones in which the characters are aligned and placed exactly in the correct place. The “imperfect” cases are the ones in which noise is introduced. In these cases, the characters may not be placed perfectly.

The “imperfect” test data are generated using the “perfect” ones. A C program is written to apply a random factor to the points in every character. As the result, the size and position of the characters may change.

```
-0.234353 0.308271 0.308271 1.475603 2.975603 -0.288061 2.419283 2.997321 F
-0.317701 -1.071782 -1.071782 -0.347062 0.152938 -1.056416 0.249886 2.306676 2
2.883301 0.068236 0.068236 -0.509777 -0.009777 -0.013239 0.146039 1.554766 1
```

Figure 3.5: One of the “imperfect” input data of the 10th test case.

Chapter 4

Design of the Software

After the test data are gathered, an approach that is modified from Wan's is used to generate Presentation MathML. This chapter describes the modules of the software: Tree Element and Presentation MathML Generator. These two modules are necessary to build an internal representation of the tree, and generate final MathML output from the internal tree.

The software is implemented in Java. The Java compiler version 1.3.1_01 installed in the computers of the ORCCA is used during development.

4.1 Modules

4.1.1 Tree Element

Every XML document can be visualized as a n-way tree. Presentation MathML markup is no exception. This module is responsible for recording an element of such tree. A tree element records some meaningful groupings in mathematical expressions.

Attributes

- Flag
  - Controls whether the current element is displayed in the final MathML generation or not.
- Children
  - An array of tree element children of the current element
- Tag
  - The MathML tag (see Figure 3.2)
  - All attributes associated with the tag

- Contents
  - The immediate text child of the current element
- All information about the *bounding box* obtained from the input file (described in section 3.1)

Methods

- Initialize
  - Create a new tree element that may appear in the final Presentation MathML generation
- Accesor methods for all attributes
  - These methods are necessary for constructing the *bounding box* of the new tree elements
- Add attribute
  - Add an extra attribute to the current tree element
- Turn off flag
  - Suppress the current tree element from printing
- Set reference point
  - Assign the absolute x-y reference point to the newly created *bounding box*
- Set body lines
  - Assign body baseline, body midline, and body topline to the newly created *bounding box*
- Set bounding box points
  - Assign the x-y values to the lower left corner and the upper right corner of the newly created *bounding box*
- Copy left element attributes
  - Assign values associated with the left hand side of the *bounding box*
  - For creating new bounding boxes that group several characters together
- Copy right element attributes
  - Assign values associated with the right hand side of the *bounding box*
  - For creating new bounding boxes that group several characters together

- Set children
    - Directly assign an array of children to the current tree element
- Add children
    - Add tree element children to the end of current tree element's children
- Insert child at
    - Directly assign a new child at a specific position of the array of children
- Remove children
    - Delete a specified range of children from the array of children
    - Suppress the deleted children from final MathML generation
- Move children
    - Delete a specified range of children **without** suppressing the deleted children from final MathML generation
- To String
    - Return the Presentation MathML markup represented by the current tree element

4.1.2 Presentation MathML Generator

Attributes

- Root
    - The topmost tree element of the mathematical expression
    - It is often a <`math`> element
- Baseline
    - Baseline of all of the "main" characters
- Operator file
    - A configuration file listing all operators
- Functions file
    - A configuration file listing all functions as predefined character sequences
- Configuration file
    - A configuration file controlling the noise that can be tolerated

Methods

- Initialize
  - Read input and configuration files
  - Put all characters under the same row regardless of their actual position
  - Predefined character sequences are not treated yet
- Group digits
  - Merge digits that are next to each other to form a single number
- Group Function Names
  - Merge characters together that form a name of a function
- Group Scripts
  - Recognize two-dimensional relationships between characters
  - (Recursive relationships are allowed)
- Add `<mfenced>`
  - Replace open and close parenthesis with `<mfenced>`
  - Add attributes to the `<mfenced>` if the parenthesis used is not a default one
  - Note that "(" and ")" are default parenthesis in Presentation MathML. Other kinds of parenthesis can be controlled by the attributes of `<mfenced>`
- Add `<mrow>`
  - Group implicit meaningful groupings in the mathematical expressions
  - (Not implemented)
- Various methods to determine the two-dimensional relationships (inline, superscript, subscript etc.) between two tree elements
  - (Except the methods for inline, superscript and subscript, none of the methods are implemented)
- To String
  - Obtain the Presentation MathML generated so far

4.2 Configuration Files

Three configuration files are needed:

- Operators
  - Define which character(s) should be included in <`mo`>s (Presentation markup for operators)
- Functions
  - Listing of predefined character sequences (Example: sin, cos)
- Other Configurations
  - Set the maximum degree of noises that the MathML generator can tolerate

Chapter 5

Presentation MathML Generation

This chapter describes how Presentation MathML is produced using an approach modified from Wan's. First, a tree that is in an internal representation is built. After this, because of the internal tree's representation it is easy to generate Presentation MathML output.

5.1 Building an Internal Tree

An internal tree is built in five steps by the Presentation MathML Generator module described in the previous chapter. First, it is necessary to read the input files and get all information of the characters. After this, digits that are next to each other and characters sequences representing known function are merged. With the updated mathematical expression, two-dimensional relationships such as superscript and subscript are recognized. To ensure proper nesting of the parenthesis, <mfenced>s replace both open and close parenthesis. To further encode the implicit semantics in Presentation MathML, <mrow>s put together some meaningful groups in expressions.

5.1.1 Generate "Flat" Presentation MathML

Upon reading the information given from the input file, every character is categorized. Every digit is put in separate <mn>s; Every letter is put in <mi>s; Every character that is an operator is put in <mo>s (Figure 5.1 and 5.2).

5.1.2 Merge Digits and Predefined Character Sequences

Without a doubt, the digits that are next to each other are to be merged to form a number. In the second test case shown in figure 5.2 and 5.1, the superscript of sin is 12 ("twelve") and not 1 and 2 separately. However, only predefined character sequences should be merged. In the second test case, the characters together that form trigonometric functions "sin" and "cos" are to be merged. The predefined character sequences are configurable using one of the configuration files.

```
0  2 0 1.5 3 0 3 4.5 int
2  8 0 0.5 1 0 1 1.5 infin
2  0 0 0.5 1 0 1 1.5 0
3  3 0 1.5 3 0 3 3   s
5  3 0 1.5 3 0 3 3   i
6  3 0 1.5 3 0 3 4.5 n
9  6 0 0.5 1 0 1 1.5 1
10 6 0 0.5 1 0 1 1.5 2
11 3 0 1.5 3 0 3 3   x
14 3 0 1.5 3 0 3 4.5 +
17 3 0 1.5 3 0 3 3   c
20 3 0 1.5 3 0 3 3   o
23 3 0 1.5 3 0 3 3   s
26 3 0 0.5 1 0 1 1.5 2
27 3 0 1.5 3 0 3 3   x
31 3 0 1.5 3 0 3 4.5 d
34 3 0 1.5 3 0 3 3   x
```

Figure 5.1: The source input of the second test case: $\int_0^\infty \sin^{12} x + \cos^2 x \; dx$

Upon merging digits and predefined character sequences, a new and larger bounding box is built (Figure 5.3). The new bounding box surrounds the smaller ones that contains separate characters. To represent such process in the internal tree, a new tree element is made. The text child of the new tree element is the matched character sequence. The original separate tree elements are suppressed from printing and become the children of the new tree element (Figure 5.4).

The merged digits are placed in a <mn> and the merged predefined character sequences are placed in a <mo>. Note that the separate characters are placed in <mi> by default when generating “flat” Presentation MathML. In other words, one should expect “s”, “i”, and “n” together result in <mo>sin</mo> as final Presentation MathML output.

5.1.3 Recognize Two-Dimensional Relationships

A tree element is fetched at a time. From the point where tree element fetched, the next tree elements can be (in this order) presubscript, presuperscript, over, superscript, subscript, under, or inline of the current tree element fetched. Once a new two-dimensional relationship is recognized, a new tree element is constructed and the applicable tree elements from the “flat” MathML become the children of the new element. The children of the new element is not suppressed from printing. Two-dimensional relationships can be applied recursively.

A larger *bounding box* is then constructed in a similar fashion of what merging digits upon the creation of a new tree element.

This step does not have knowledge of any disallowed relationships between charac-

```
<math xmlns="http://www.w3.org/MathML/">
  <mrow>
    <mi>&int;</mi>
    <mi>&infin;</mi>
    <mn>0</mn>
    <mi>s</mi>
    <mi>i</mi>
    <mi>n</mi>
    <mn>1</mn>
    <mn>2</mn>
    <mi>x</mi>
    <mo>+</mo>
    <mi>c</mi>
    <mi>o</mi>
    <mi>s</mi>
    <mn>2</mn>
    <mi>x</mi>
    <mi>d</mi>
    <mi>x</mi>
   </mrow>
  </math>
```

Figure 5.2: The “flat Presentation MathML output generated by the first step

ters. In other words, meaningless mathematical expressions such as “$)^3$” do not generate an error.

After this step, the Presentation MathML generated should be displayed “correctly” with the added two-dimensional relationships.

5.1.4 Replace Parenthesis with <mfenced>s

Having a mathematical expression that is displayed “correctly” is not good enough. Instead, implicit semantics should be added within the markup. To ensure that the parenthesis are well nested, every pair of matching open and close parenthesis is replaced by a pair of `<mfenced>`. Attributes of `<mfenced>` can indicate the kind of open and close parenthesis. This operation should be applied recursively.

5.1.5 Add <mrow>s to Denote Implicit Groupings

`<mrow>`s can do more than indicating the elements that belong to the same row. It can also indicate implicit proper groupings of the elements like what “{“ and “}” do in TEX. The MathML recommendation suggests some rules for for proper grouping to further indicate the implicit semantics within the presentation markup. These rules include:

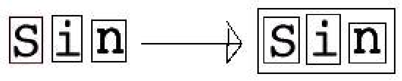

Figure 5.3: Putting characters into a larger *bounding box*

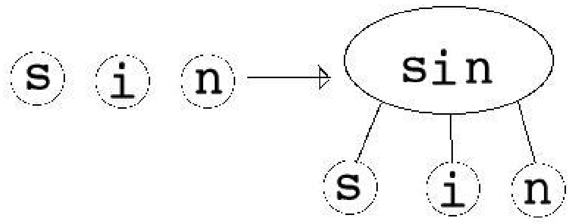

Figure 5.4: The internal representation of the merging

- Operands between operators should be enclosed by <mrow>s
- Within a <mrow>, include only one or two operators
  - It is necessary to determine the precedence of the operators
  - Nested <mrow>s may be used

5.2 Generating Presentation MathML

Obtaining Presentation MathML can be obtained by performing a preorder traversal of the internal tree. This step is straight-forward because Presentation MathML relationships are annotated in the internal tree.

Although Presentation MathML provides methods ("mspace" and "mpadded") to adjust space around and within the mathematical expression, adjusting spaces for aesthetic purpose is not considered in MathML generation in this project. This project concerns with the two-dimensional relationships between the characters, so adjusting space for aesthetic purpose is outside of the scope. Furthermore, handwriting is inconsistent. It is hard to set the rule of thumb whether space should be adjusted in the final Presentation MathML generation.

Chapter 6

Conclusion

This chapter concludes this report by suggesting future works in this area and the experience of this project. A more intelligent way to generate noise in the input data and tolerate noise in the software is needed. The approaches to determine two-dimensional relationships must improve. If the system knows more mathematics, it is possible to include implicit semantics in the Presentation MathML markup.

6.1 Possible Future Works

6.1.1 Improve Methodology to Generate and Tolerate Noise

To add noise the test cases, a random factor from -1 to 1 is added to or subtracted from the values that describe the *bounding boxes*. As the result, each character may change in size or position. Actual handwriting's noise may involve more than mere applying a random factor. Further studies in handwriting recognition are needed to generate realistic test cases using a computer.

6.1.2 Better Approaches to Determine Two-Dimensional Relationships

The implementation of the MathML generator uses some naive approaches to determine two-dimensional relationships. The most naive one is the one to determine the "inline" relationship. To say if two characters are inline, both of the baselines of the boxes must be exactly the same. Actually, it is impossible to make the users to align the characters perfectly in handwriting.

Other approaches to determine two-dimensional relationships are not as naive as the "inline" one, but there is room to improve. To say if one character is over or under another one, the central line of the overscript must be vertically aligned between the two sides of the "parent" character. Sometimes, it is hard to determine the difference between a superscript and a simple overscript, and between a subscript and a simple underscript. This approach also fails to recognize a overscript or underscript of several characters such as the sixth test case ($\sum_{i=0}^{\infty} i = 0 + 1 + 2 + 3 \ldots$), so the approach must be refined to do so.

Distance between characters should be added as one of the criteria for recognizing relationships. In the sixth test case ($x^{x^{x^{x}}} + 2x$), there is no intermediate relationships between "$+$" and the last "x" in "$x^{x^{x^{x}}}$". Associating such characters correctly involves knowing the relative distance between the characters. "$+$" can go to one of the superscripts like "$x^{x^{x^{x}+}}$", and $+$ is surely not the subscript of the last "x".

Because of these oversimplified approaches are used, only the "perfect" test cases is used to test the software.

6.1.3 Insert More Implicit Semantics in MathML Generation

To inserting implicit semantics in Presentation MathML, the system must know mathematics such as precedence rules and implicit operators. At the same time, it is not good to hard code these knowledge into the software because such software does not allow one to introduce new mathematical notations. The current implementation partly achieve this goal by using configuration files for defining operators and predefined character sequences.

A representation for precedence rules is necessary to add <mrow>s properly. It may be sufficient to hard code the routine for adding invisible operators such as `&invisibleTimes;` because there is only a limited set of such operators.

When merging predefined character sequences, it is possible to add `&invisibleTimes;` between characters that do not form any function names.

6.2 Experience

A system must know mathematics to treat mathematics, and treating the representation of mathematics is not easy. Handwriting devices today cannot be used to input mathematics because they do not understand the two-dimensional relationships between characters. TeX/MathML converter cannot preserve and add the implicit semantics because it does not know mathematics.

On one hand, too much mathematical knowledge in the system limits the kinds of mathematics that one can enter. On the other hand, a system that has a limited knowledge of mathematics cannot possibly captures and preserves the implicit semantics of mathematics. This project proposes what should a system knows in order to treat mathematics. Such system should know the two-dimensional relationships between characters and the convention of mathematics. Since mathematical notations are semi-standardized, it is desirable to have a system that is very flexible.

Bibliography

[1] David Carlisle, Patrick Ion, Robert Miner, Nico Poppelier, Editors. Ron Ausbrooks, Stephen Buswell, David Carlisle, Stéphane Dalmas, Stan Devitt, Angel Diaz, Roger Hunter, Patrick Ion, Robert Miner, Nico Poppelier, Bruce Smith, Neil Soiffer, Robert Sutor, Stephen Watt, Principal Authors. *Mathematical Markup Language (MathML) Version 2.0 (2nd Edition)*, W3C Recommendation. Available: http://www.w3.org/TR/MathML2/, December 19, 2002

[2] *OpenMath Website* Available: http://www.openmath.org

[3] Stéphane Lavirotte. *Reconnaissance structurelle de formules mathématiques typographié et manuscrites*, Doctoral Thesis, École Doctoale des Sciences et Technologies de l'Information et de la Communication, Université de Nice - Sophia Antipolis, France, June, 2000. (In French)

[4] Bo Wan. *An Interactive Mathematical Handwriting Recognizer for the Pocket PC*, MSc Thesis. Deparment of Computer Science, University of Western Ontario, December, 2001.