Tools for MathML

by

Igor Rodionov

Department of Computer Science

3

Submitted in partial fulfillment
of the requirements for the degree of
Master of Science

Faculty of Graduate Studies
The University of Western Ontario
London, Ontario
January 3, 2001

Abstract

Originally invented in the early 1990s at the dawn of the Internet era, Hypertext Markup Language (HTML) is the most widespread data format on the Web these days. However, it has a number of weaknesses, one of them being HTML's inability to represent and communicate mathematical objects. Recognizing the problem, a well-known Internet standards organization, The World Wide Web Consortium (W3C), undertook the daunting task of creating a standard to rectify the situation. The result is the standard called Mathematical Markup Language (MathML).

In this thesis we study two issues relating to the suitability of MathML as a data format: MathML abstraction mechanism, and visual rendering of MathML objects. An abstraction mechanism for MathML is necessary to structure mathematical data in a manner that allows the separation of form and content; separate concepts can then be introduced individually. Moreover, this makes MathML extremely flexible in terms of presentation, since any given mathematical concept can be rendered according to the user's preference, which can be achieved by using the Extensible Stylesheet Language Transformations (XSLT).

This thesis also introduces two software applications supporting the new MathML standard. The first of them is a XSLT stylesheet that enables a client application (e.g. a browser) to display MathML objects. This application is a good illustration that MathML abstraction mechanism really works. The second application makes it possible to convert TEX documents into MathML, which is very useful since a substantial portion of the technical documents over the past 20 years have relied on TEX for the mathematics. More importantly, it shows that MathML can be generated from other data formats, thus making MathML even more valuable.

Acknowledgements

I would like to thank my supervisor Dr. Stephen Watt for being so enthusiastic, demanding and encouraging, and for keeping me on the right track during the course of my graduate studies under his supervision, in particular for suggesting a very interesting topic for my research.

I would also like to thank all other faculty, staff, and fellow students whom I got to know while studying at the department – I had a great time!

Thanks go to my girlfriend for making it possible for me to keep concentrated on writing this thesis.

Finally, I would like to express my endless gratitude to my parents for their trust in me, for their love and support, and for always being there for me.

Table of Contents

Certificate ii

Abstract iii

Acknowledgements iv

Table of Contents v

1 Introduction 1

2 An Overview of MathML 5

2.1 Why Two Kinds of Markup? 5

2.2 MathML Content Elements 9

2.2.1 Token Elements 11

2.2.2 Basic Content Elements 13

2.2.3 Arithmetic, Algebra and Logic Content Elements 19

2.2.4 Relations 28

2.2.5 Calculus and Vector Calculus 29

2.2.6 Theory of Sets 33

2.2.7 Sequences and Series 36

2.2.8 Elementary Classical Functions 38

2.2.9 Statistics 40

2.2.10 Linear Algebra 42

2.2.11 Semantic Mapping Elements 46

2.3 MathML Presentation Elements 47
2.3.1 Token Elements 48
2.3.2 General Layout Schemata 53
2.3.3 Script and Limit Schemata 58
2.3.4 Tables and Matrices 62

3 An Overview of XSLT 64
3.1 Structure of a Stylesheet and Top Level Elements 64
3.2 Templates and XPath 69

4 Rendering Content MathML as Presentation MathML 76
4.1 Layout of the Stylesheet 76
4.2 The Six Modes of Operation 80
4.2.1 Stripping Semantics off 80
4.2.2 Passing Semantics 'As Is' 82
4.2.3 Adding Semantics at the Top Level Only 82
4.2.4 Adding Semantics at All Levels 84
4.2.5 Adding Semantics at Top Level Only with Cross-References 86
4.2.6 Generating Cross-References with no Semantics 88
4.3 Fine Points of Generating MathML Presentation 89
4.3.1 Operator Precedence and Parenthesizing 89
4.3.2 Output Optimization 92
4.4 Output Localization 96

5. Rendering TEX Mathematics to MathML Presentation 98
5.1 The TEX Macro Expander Application 98
5.2 The Main Application 100

6. Summary and Conclusion 105

Bibliography 106

Appendixes

A The MathML Content to MathML Presentation Stylesheet **107**

B The ID Generator for MathML Documents **162**

Vita **164**

Chapter 1

Introduction

These days, the vast majority of data on the Internet is stored in HTML format. Just like the Internet itself, HTML did not come to this world as something that was carefully planned. This data format was being developed on the fly as the Internet was booming in the early 1990s. As a result, HTML has a number of serious drawbacks. One such drawback is inconsistency of HTML's syntax. For example, it is not required that certain tags have to be closed, e.g. the <p> tag designating start of a new paragraph, because usually the end of a paragraph can be inferred. This saves storage space and makes it easier to write HTML by hand, but at the same time makes parsing of HTML documents harder, although from the user's point of view HTML syntax is more forgiving, and therefore "better". Nowadays, HTML editors and other visual tools are used to generate HTML, and therefore such syntax 'simplifications' provide no advantage whatsoever.

Another problem with HTML is its inflexibility. The HTML standard defines a number of tags with fixed meaning. This stands in contrast to the eXtensible Markup Language (XML), where new tags can be defined and assigned particular meaning in the context of a document by means of the DTD (Document Type Declaration). XML is expected to replace HTML and its derivatives (e.g. XHTML) over the next few years.

Among most serious weaknesses of HTML is its inability to represent and communicate mathematical objects, such as mathematical formulas. In order to alleviate the problem, a number of solutions were developed by different vendors over the past few years. However, none of them became widely accepted, mostly because of their proprietary nature, and because none of them could really offer all the functionality

needed for the effective authoring and exchange of mathematical objects. Recognizing the problem, a major Internet standards body, the World Wide Web Consortium (W3C), undertook the daunting task of creating a standard to rectify the situation. A number of leading scientists and industrial organizations had become involved in the development of the standard. The result was the Mathematical Markup Language (MathML) specification.

As the World Wide Web is slowly but surely migrating towards being XML-based, the fact that MathML is an XML application becomes quite significant. This means that general XML tools, such as the Extensible Stylesheet Language Transformations (XSLT) can be used with MathML, and that MathML can be easily embedded in any XML or XHTML document. This is important, because while MathML can be used to exchange mathematical objects between MathML-aware applications, the expected primary use of MathML is to encode mathematical expressions within larger documents published on the Web.

Another very attractive thing about MathML is that it is supported and promoted by the W3C. This pretty much ensures that the standard will find its way into major commercial products, including browsers and mathematical packages, which would guarantee the standard's viability and success.

In this thesis we take an in-depth look into the MathML standard, discuss the suitability of MathML as a data format for authoring, exchange and publication of mathematically rich content, and introduce two new applications supporting the standard to see how well MathML concepts work in practice.

Two issues will be of particular interest to us in this thesis: the MathML abstraction mechanism, and rendering of MathML objects. Abstraction is a very important concept introduced in MathML. The idea behind the abstraction mechanism is to separate presentation information about a mathematical object from its semantics. Such approach is very natural because mathematical concepts exist regardless of the notation used to represent them. Since different mathematical notations for the same

mathematical concept are often used concurrently, it is clear that the ambiguity of mathematical notation makes it very difficult (if possible at all) to introduce a system that would rely solely on the presentation aspect for encoding of mathematical information. On the other hand, using only the information about semantic meaning of a mathematical object for encoding it would not have the same problem, because such encoding is unambiguous. However, several questions arise here: since no information is now explicitly given about how the object should be visually presented (rendered), how could it be properly displayed? Also, is it possible at all to infer proper rendering for such an object? The answer to the latter question is especially important, because if it were possible, there would be no need to provide rendering information for every object! The obvious benefit of this is reduced time and space required to encode a mathematical object. The disadvantage is that the author would have no control over the display of such an object. A more flexible approach would be to combine the semantics and presentation in one entity so that the object could be rendered in accordance with its author's preferences, while keeping its semantic meaning. Thus yet another question is: Does MathML have provision for combining the two kinds of encoding?

In order to be able to answer the above questions we need to take a closer look at the MathML standard itself. Chapter 2 opens with the discussion of why two kinds of markup (content and presentation) are present in MathML. Then Sections 2.2 and 2.3 provide detailed description of all the content and most of the presentation elements (tags) of MathML, accompanied with many examples. Knowledge of the MathML elements is essential to understanding the rest of the work done in this thesis, especially Chapters 4 and 5.

Later in Chapter 2 we come to the conclusion that we need a particular tool, such as the XSLT, to perform the transformation of MathML content markup into presentation markup, and to combine them together. Therefore, Chapter 3 gives an overview of XSLT, and explains what XSLT stylesheets are and how they work, which prepares us for Chapter 4, where all the pieces finally come together: we discuss the XSLT stylesheet

that maps MathML content markup into presentation markup. There we also discuss different ways of generating the output (e.g. producing pure presentation markup, or a combination of the content and presentation), and a number of related issues, such as sub-expression parenthesizing and output optimization.

Finally, Chapter 5 discusses yet another aspect of MathML. One of the goals set by the developers of MathML was to create a markup language that would be easily interchangeable with other data formats, in particular, with TEX. In Chapter 5 we investigate how TEX-encoded mathematics can be converted into MathML presentation markup.

Chapter 2

An Overview of MathML

Currently, the MathML specification defines approximately 180 elements (also called tags), which can be divided into two groups. Tags from the first group (approximately 150 of them) can be used to unambiguously encode semantic meaning of mathematical expressions; they are not concerned at all with the way mathematical expressions are displayed. Tags in this group are commonly referred to as content elements, or content markup. Tags from the second group (approximately thirty of them) are solely concerned with the presentation aspect, and are commonly referred to as presentation elements, or presentation markup. At this point, the following question arises...

2.1 Why Two Kinds of Markup?

Mathematical concepts exist regardless of the notation used to represent them. At the same time, mathematical notation tends to change over time. It is also not uncommon that the same mathematical concept can be expressed differently in different geographical areas of the world, or even that several different notations for the same concept exist concurrently in the same area. Thus, it only seems natural to separate the semantics and presentation of a mathematical expression. Still, the question remains: could it be better to have only one type of markup? Theoretically, it would mean that there are fewer things to deal with, and simpler solutions are often better. Let us see what would happen if we chose to use only one kind of markup.

Let us say, we decided to use only the presentation markup. The fastest way to realize that this approach would not work very well is to consider the following example. Figure 2.1 shows three different ways to represent division of x by 13 (detailed explanation of the content and presentation elements involved in this and following examples in this section can be found in chapters 3 and 4). It is easy to see that there are at least three different ways to encode this expression using MathML presentation markup. This shows the inherent ambiguity of using only the presentation markup. One big disadvantage of such an approach is that objects encoded using only MathML presentation elements would have to be modified if they are to be viewed by users who prefer different mathematical notation. This defeats the purpose of MathML – to have a single universal language for exchange of mathematical information across the world.

```
<mrow>
  <mi> x </mi>
  <mo> / </mo>
  <mn> 13 </mn>
</mrow>
```

$x/13$

```
<mrow>
  <mi> x </mi>
  <mo> ÷ </mo>
  <mn> 13 </mn>
</mrow>
```

$x \div 13$

```
<mfrac>
  <mi> x </mi>
  <mn> 13 </mn>
</mfrac>
```

$$\frac{x}{13}$$

Figure 2.1 Examples of MathML presentation markup, with corresponding rendering

Another big problem with using only the presentation markup is that it would be extremely difficult to produce software (such as mathematical packages, e.g. Maple or Mathematica) to manipulate those mathematical objects, again because of the diversity of the existing mathematical notations.

The next question is: would it make sense to include into MathML only content markup elements without presentation elements? Let us consider another example.

Figure 2.2 shows MathML content markup corresponding to the presentation markup given in the Figure 2.1. Using only MathML content markup is certainly a possibility. However, that would mean that MathML would be very inflexible in terms of presentation because authors of MathML documents would not have any control over the display of their documents.

```
<apply>
  <divide/>
  <ci> x </ci>
  <cn> 13 </cn>
</apply>
```

Figure 2.2 Example of MathML presentation markup

This brings us to the conclusion that both kinds of markup are needed. This way, they can be used independently, or mixed, or combined, which makes MathML extremely flexible. Let us see why.

Let us suppose that a MathML object is encoded using only the content elements. Such an object cannot be displayed directly, because content markup only captures the semantic meaning of an expression, and not how the expression is to be displayed. In order to be able to view such an object on the screen it has to be transformed into the presentation markup first. Is such transformation possible? Yes it is, because MathML content encoding is unambiguous. Then how can such transformation be done? The easiest way to do this is to use a stylesheet. Just like CSS (Cascading Stylesheets) are commonly used with HTML documents, XSLT (Extensible Stylesheet Language Transformations) stylesheets can be used with MathML. XSLT was specifically designed for transforming XML documents, and thus can be used with MathML, which is an XML application.

Therefore, a localized stylesheet might be used to transform MathML content elements into MathML presentation elements, which could then be directly rendered by a client application. The advantage of such an approach is that the author of a MathML object does not have to worry about how it will be rendered. Any particular rendering is not enforced, and it is only dependant on the stylesheet used by the client application. For example, the markup in Figure 2.2 might be transformed into any of the three presentation fragments shown in Figure 2.1, depending on the user's choice (that is depending on the stylesheet the user chooses).

Another possibility would be to combine presentation and content markup. With this approach, the author of a MathML object has more control over how it is rendered, because the object contains both the content markup, and corresponding presentation. This means, that the client application can either use this suggested presentation markup by default, or extract the content markup, and use its own stylesheet to produce the presentation markup that could be subsequently rendered. An example of combined markup is given in Figure 2.3.

```
<semantics>
  <mfrac>
    <mi> x </mi>
    <mn> 13 </mn>
  </mfrac>
  <xml-encoding type='MathML'>
    <apply>
      <divide/>
      <ci> x </ci>
      <cn> 13 </cn>
    </apply>
  </xml-encoding>
</semantics>
```

Figure 2.3 Example of MathML combined markup

Now that we have talked about how things work in principle, it is time to take a closer look at MathML itself, and how MathML objects can be transformed using XSLT stylesheets, which we talk about in Chapter 3. But first we start with the discussion of MathML content and presentation elements. It is important that we consider all MathML elements in detail because the stylesheet that we talk about in Chapter 4 has to map *each and every* one of the content elements into presentation elements. We also need knowledge of MathML presentation elements when we discuss how to transform TEX documents into MathML presentation markup.

2.2 MathML Content Elements

The intent of the content markup in MathML is to provide an explicit encoding of the underlying mathematical structure of an expression, rather than any particular rendering for the expression.

The set of content elements of MathML does not span all of mathematics though. It would be extremely hard, if possible at all, to include every existing mathematical notion into MathML. Instead, the set of content elements was chosen to be sufficient for encoding most of mathematical expressions used in high school and college, with the provision for extensibility.

MathML content encoding is based on the concept of an expression tree. An example of such a tree is given in Figure 2.4. Note that when a mathematical expression is encoded the information about operator precedence is not needed, as the order of evaluation depends on the position of the operator in the tree. For instance, in the example given in Figure 2.4 the addition will be performed before the division.

```
<apply>
  <divide/>
  <apply>
    <plus/>
    <ci> x </ci>
    <ci> y </ci>
    <cn> 1 </cn>
  </apply>
  <ci> n </ci>
</apply>
```

Figure 2.4 MathML encoding for the expression $\frac{x+y+1}{n}$

All MathML content elements can be grouped into the following categories based on their usage:

- Containers
- Operators and functions
- Qualifiers
- Relations
- Conditions
- Semantic mappings
- Constants and symbols.

These are the building blocks from which MathML content expressions are constructed. Let us discuss them in more detail.

2.2.1 Token Elements

- **Number – `cn`**

Names of many content elements start with letter "c", which stands for content. The `cn` (content number) element is used to specify actual numerical constants. The content model must provide sufficient information that a number may be entered as data into a computational system. By default, it represents a signed real number in base 10. The `cn` element can use attributes to modify the default behavior. The `type` attribute specifies the type of the number being encoded. It can be `real`, `integer`, `rational`, `complex`, `complex-cartesian`, `complex-polar`, or `constant`. The `base` attribute, naturally, specifies the numeric base of the number, and has to be between 2 and 36. The only other element permitted inside the `cn` element is the `sep` element, which is used in conjunction with complex numbers. Here are some examples:

```
<cn> -33 </cn>
<cn type = "real"> 33.12 </cn>
<cn type = "integer" base="16"> 4A10BEEF </cn>
<cn type="complex"> 3.3 <sep/> -1.2 </cn>
```

In the examples above, the third number is the hexadecimal constant $4A10BEEF_{16}$, and the last number is the complex constant $3.3 - 1.2i$.

- **Identifier – `ci`**

The `ci` (content identifier) element is used to name an identifier in MathML. It can use the `type` attribute to specify the type of object that it represents. Valid types include `integer`, `rational`, `real`, `float`, `complex`, `constant`, and any of the names of

the MathML container elements, such as `vector`, `matrix`, etc. By default `ci` is assumed to represent complex scalars. The `ci` element also may contain arbitrary presentation markup in its content. Here are some examples:

```
<ci> x </ci>
<ci type="vector"> v </ci>
<ci>
  <msubsup>
    <mi> x </mi>
    <mi> i </mi>
    <mn> 0 </mn>
  </msubsup>
</ci>
```

In the second example above v is a vector, and would normally be rendered either in bold like $\mathbf{v}$, or as $\bar{v}$, but that is up to the stylesheet. The third example gives the encoding for x_i^0.

- **Externally defined symbol – `csymbol`**

The `csymbol` element makes it possible to create a new element whose semantics are externally defined. This is useful in cases when MathML does not have that element already built in. This new element can then be used in a MathML expression just like any other element. There are the following two attributes that `csymbol` can have. First is `definitionURL`, which points to the external definition of the semantics of the symbol. The second is the `encoding` attribute, which gives the syntax of the definition pointed to by `definitionURL`. An application can then test the value of this attribute

to determine whether it is able to process the target of the `definitionURL`. This syntax might be text, or a formal syntax such as OpenMath.

Here are some examples:

```
<csymbol encoding="text"
    definitionURL="www.any.org/DescriptionOfConstantG.html">
  G
</csymbol>

<csymbol encoding="OpenMath"
    definitionURL="www.openmath.org/cds/BesselFunctions.ocd">
  <msub>
    <mi> J </mi>
    <mn> 0 </mn>
  </msub>
</csymbol>
```

In the first example above the new constant G is introduced, and its description in the form of human readable text is given at the URL given in the `definitionURL` attribute. In the second example, the attributes indicate that the semantics for the symbol J_0 is given in the form of an OpenMath dictionary (OpenMath is a standard used for semantic encoding of mathematical objects; OpenMath dictionaries are collections of related mathematical concepts expressed in OpenMath markup).

2.2.2 Basic Content Elements

- **Apply – `apply`**

Nearly all expression construction in MathML content markup is done by applying operators or functions to arguments. The `apply` element is used to apply a function or

operator to its arguments. The first child of the `apply` is the operator to be applied, and the other children are its arguments.

The number of children of the `apply` element depends on the function or operator being applied. For instance, in case of multiplication the number of children might be two or more, as multiplication is an *n*-ary operation. However, in the case of the minus operator, the number or arguments can be either one or two, as minus can be either unary (negation) or binary (subtraction).

Some operators such as `diff` and `int` require that certain information be supplied, such as the variable with respect to which the differentiation or integration is carried out.

Here are some examples of use of the `apply` element:

```
<apply>
  <sin/>
  <ci> x </ci>
</apply>
```

```
<apply>
  <int/>
  <bvar>
    <ci> x </ci>
  </bvar>
  <apply>
    <power/>
    <ci> x </ci>
    <cn> 2 </cn>
  </apply>
</apply>
```

The first example provides the encoding for $\sin x$, and the second one shows how $\int x^2 dx$ can be encoded. Note the use of the `bvar` element (bound variable), which specifies the variable with respect to which integration is to be carried out.

Note that the `apply` element can be used recursively. Here is another example that shows the encoding for $(n+1)!$

```
<apply>
  <factorial/>
  <apply>
    <plus/>
    <ci> n </ci>
    <cn> 1 </cn>
  </apply>
</apply>
```

- **Interval – `interval`**

The `interval` element is used to represent mathematical intervals of the real number line. It takes an attribute `closure`, which can assume the following values: `open` (on both ends), `closed` (on both ends), `open-closed`, or `closed-open`. The default value is `closed`. Here is an example:

```
<interval closure="closed-open">
  <cn> 0 </cn>
  <ci> x </ci>
</interval>
```

The default rendering for this markup is $[0, x)$.

- **Inverse – `inverse`**

The `inverse` element is applied to a function in order to construct a generic expression for the functional inverse of that function. Here is an example:

```
<apply>
  <apply>
    <inverse/>
    <ci> f </ci>
  </apply>
  <ci> x </ci>
</apply>
```

The default rendering for the above markup is $f^{(-1)}(x)$.

- **Separator – `sep`**

The `sep` element is used to separate tokens in specialized forms of the `cn` element. Here is an example:

```
<cn type="complex"> -5 <sep/> 3 </cn>
```

The `sep` element is not directly rendered. The default rendering for the above is $-5+3i$.

- **Condition – `condition`**

The `condition` element is used to place a condition on one or more variables. The conditions may be specified in terms of relations that are to be satisfied by the variables. It is often used in conjunction with other elements, such as `min/max`, `int`, etc. Here is an example (for more examples see descriptions of e.g. `int`):

```
<condition>
  <apply>
    <gt/>
    <ci> x </ci>
    <cn> 0 </cn>
  </apply>
</condition>
```

The default rendering for the above is $x > 0$.

- **Declaration – `declare`**

The `declare` construct can be used to either associate a name with an object, or to declare an object's type. Either way, it is not directly rendered, and therefore we shall not talk about it any further.

- **Lambda expression – `lambda`**

The `lambda` element is used to construct a user-defined function from an expression and n variables, where n has to be 1 or more. The lambda construct then takes $n+1$ children. The first n of them identify the variables that are used as placeholders in the last child for actual parameter values. Here is an example:

```
<apply>
  <lambda/>
  <bvar>
    <ci> x </ci>
  </bvar>
  <apply>
    <factorial/>
    <ci> x </ci>
  </apply>
</apply>
```

The above should render as $\lambda(x, x!)$.

- **Function composition – `compose`**

The `compose` element represents the function composition operator, and can be used to compose two or more functions. Here is an example of how it can be used:

```
<apply>
  <apply>
    <compose/>
    <ci type="fn"> f </ci>
    <ci type="fn"> g </ci>
    <ci type="fn"> h </ci>
  </apply>
  <ci> x </ci>
  <ci> y </ci>
</apply>
```

The default rendering for the above is $(f \circ g \circ h)(x, y)$. Note that `ci` elements with the `type="fn"` attribute is not the only way to represent a function. Another way of doing it would be with the `fn` element:

```
<fn>
  <ci> f </ci>
</fn>
```

- **Identity function – `ident`**

The `ident` element represents an identity function. It is rendered by default as `id`.

2.2.3 Arithmetic, Algebra and Logic Content Elements

- **Quotient – `quotient`, division – `divide`, remainder – `rem`, subtraction – `minus`**

All of the above operators are binary arithmetical operators. The `quotient` element is used to represent the operator for division modulo a particular base. The `divide` element is the division operator. The `rem` element is the operator that returns the remainder of a division modulo a particular base. The `minus` element is the subtraction element. Here is how they can be used:

```
<apply>
  <quotient/>
  <ci> x </ci>
  <ci> y </ci>
</apply>
```

```
<apply>
  <divide/>
  <ci> x </ci>
  <ci> y </ci>
</apply>
```

```
<apply>
  <rem/>
  <ci> x </ci>
  <ci> y </ci>
</apply>
```

```
<apply>
  <minus/>
  <ci> x </ci>
  <ci> y </ci>
</apply>
```

The default renderings for the above are:

$\lfloor x/y \rfloor$ x/y $x \bmod y$ $x - y$

- **Factorial – `factorial`**

This element is used to construct factorials. It is a unary arithmetic operator. Here is an example of how it can be used:

```
<apply>
  <factorial/>
  <ci> n </ci>
</apply>
```

Default rendering for the above code is $n!$

- **Maximum and minimum – `max`, `min`**

These elements are used to compare the values of their arguments, and return the maximum and minimum of these values respectively. They are n-ary arithmetic operators:

```
<apply>
  <min/>
  <ci> x </ci>
  <ci> y </ci>
  <ci> z </ci>
</apply>
```

The above code should be rendered as $\min\{x, y, z\}$.

More complex versions of `min` and `max` can be constructed. Bound variables and condition imposed on them can be specified:

```
<apply>
  <max/>
  <bvar>
    <ci> x </ci>
  </bvar>
  <condition>
    <apply>
      <in/>
      <ci> x </ci>
      <ci type="set"> S </ci>
    </apply>
  </condition>
  <apply>
    <ci type="fn"> f </ci>
    <ci> x </ci>
  </apply>
</apply>
```

The above should render as $\max_{x} \{f(x) \mid x \in S\}$.

- **Addition – `plus`, multiplication – `times`**

The `plus` element is the addition operator. The `times` element is the multiplication operator. Both are *n*-ary arithmetic operators:

```
<apply>
  <plus/>
  <cn> 3 </cn>
  <ci> x </ci>
  <ci> y </ci>
</apply>
```

```
<apply>
  <times/>
  <cn> 3 </cn>
  <ci> x </ci>
  <ci> y </ci>
</apply>
```

Default renderings are $3+x+y$ and $3xy$ respectively.

- **Exponentiation – `power`**

The `power` element is used for the exponentiation operation. It is a binary arithmetic operator:

```
<apply>
  <power/>
  <apply>
    <plus/>
    <ci> a </ci>
    <ci> b </ci>
  </apply>
  <cn> 2 </cn>
</apply>
```

This should render as $(a+b)^2$.

- **Root – `root`**

The `root` element is used to construct roots. Degree can be specified by using the `degree` element, which should appear as the first child of the `apply` element enclosing the `root` element. If no degree is specified the default of 2 is assumed.

```
<apply>
  <root/>
  <degree>
    <ci type="integer"> n </ci>
  </degree>
  <ci> x </ci>
</apply>
```

The above should render as $\sqrt[n]{x}$.

- **Greatest common divisor – `gcd`, lowest common multiple – `lcm`**

The `gcd` element is used to denote the greatest common divisor or its arguments. The `lcm` element is used to denote the lowest common multiple of its arguments. Both are *n*-ary operators:

```
<apply>
  <gcd/>
  <ci> x </ci>
  <ci> y </ci>
  <ci> z </ci>
</apply>
```

```
<apply>
  <lcm/>
  <ci> x </ci>
  <ci> y </ci>
  <ci> z </ci>
</apply>
```

The above should render as $\gcd(x, y, z)$ and $lcm(x, y, z)$ respectively.

- **Logical operators – `and`, `or`, `xor`, `not`**

Elements `and`, `or` and `xor` are boolean and, or and exclusive or operators respectively. They are *n*-ary logical operators. The `not` element is the boolean not operator, and it is a unary logical operator:

```
<apply>
  <and/>
  <ci> x </ci>
  <ci> y </ci>
  <ci> z </ci>
</apply>
```

```
<apply>
  <or/>
  <ci> x </ci>
  <ci> y </ci>
  <ci> z </ci>
</apply>
```

```
<apply>
  <xor/>
  <ci> x </ci>
  <ci> y </ci>
  <ci> z </ci>
</apply>
```

```
<apply>
  <not/>
  <ci> x </ci>
</apply>
```

The above should be rendered as:

$x \wedge y \wedge z$ $x \vee y \vee z$ $x \text{ xor } y \text{ xor } z$ $\neg x$

- **Implies – `implies`**

The `implies` element is the boolean relational operator 'implies'. It is a binary logical operator:

```
<apply>
  <implies/>
  <ci> A </ci>
  <ci> B </ci>
</apply>
```

The above should be rendered as $A \Rightarrow B$.

- **Universal qualifier – `forall`**

The `forall` element represents the universal quantifier of logic. It is used in conjunction with one or more bound variables, and optional `condition` element:

```
<apply>
  <forall/>
  <bvar>
    <ci> x </ci>
  </bvar>
  <condition>
    <apply>
      <and/>
      <apply>
         <in/>
         <ci> x </ci>
         <ci type="set"> R </ci>
      </apply>
      <apply>
        <gt/>
        <ci> x </ci>
        <cn> 1 </cn>
      </apply>
    </apply>
  </condition>
  <apply>
    <lt/>
    <ci> x </ci>
    <apply>
      <power/>
      <ci> x </ci>
      <cn> 2 </cn>
    </apply>
  </apply>
</apply>
```

This should render as $\forall x : x \in R$ and $x > 1, x < x^2$.

- **Existential quantifier – `exists`**

The `exists` element represents the existential quantifier of logic. It must be used in conjunction with one or more bound variables, an optional `condition` element, and an assertion, so it is very similar to `forall`:

```
<apply>
  <exists/>
  <bvar>
    <ci> x </ci>
  </bvar>
  <apply>
    <eq/>
    <apply>
      <ci type="fn"> f </ci>
      <ci> x </ci>
    </apply>
    <ci> x </ci>
  </apply>
</apply>
```

This should render as $\exists x : f(x) = x$.

- **Absolute value – `abs`**

The `abs` element represents the absolute value of a real quantity or the modulus of a complex quantity. It is a unary arithmetic operator:

```
<apply>
  <abs/>
  <ci> x </ci>
</apply>
```

This should render as $|x|$.

- **Complex conjugate – `conjugate`**

The `conjugate` element represents the complex conjugate or a complex quantity. It is a unary arithmetic operator:

```
<apply>
  <conjugate/>
  <cn type="comlex"> 3 <sep/> -5 </cn>
</apply>
```

This should render as $\overline{3-5i}$.

- **Argument – `arg`, real part – `real`, imaginary part – `imaginary`**

The `arg` operator gives the 'argument' of a complex number, which is the angle it makes with the positive axis. The `real` operator gives the real part of a complex number. The `imaginary` operator gives the imaginary part or a complex number. All three are unary arithmetic operators:

```
<apply>
  <arg/>
  <cn
type="comlex"> 3
<sep/> -5 </cn>
</apply>
```

```
<apply>
  <real/>
  <cn
type="comlex"> 3
<sep/> -5 </cn>
</apply>
```

```
<apply>
  <imaginary/>
  <cn
type="comlex"> 3
<sep/> -5 </cn>
</apply>
```

The above should render as:

$\arg(3-5i)$ $\Re(3-5i)$ $\Im(3-5i)$

- **Round-down – `floor`, round-up – `ceiling`**

The `floor` element is used to denote the round-down (towards $-\infty$) operator. The `ceiling` element is used to denote the round-up (towards $+\infty$) operator. Both are unary operators:

```
<apply>
  <floor/>
  <ci> x </ci>
</apply>
```

```
<apply>
  <ceiling/>
  <ci> x </ci>
</apply>
```

These should render as $\lfloor x \rfloor$ and $\lceil x \rceil$ respectively.

2.2.4 Relations

- **Binary relations**

There are two binary relations defined in MathML: equals – `eq`, and approximately – `approx`:

```
<apply>
  <eq/>
  <ci> x </ci>
  <cn> 0 </cn>
</apply>
```

```
<apply>
  <approx/>
  <ci> x </ci>
  <cn> 0 </cn>
</apply>
```

The above should render as $x = 0$ and $x \approx 0$ respectively.

- ***N*-ary relations**

N-ary relations include not equals – `neq`, greater than – `gt`, less than – `lt`, greater than or equal – `geq`, less than or equal – `leq`, equivalent – `equialent`. These operators should be rendered as $\neq, >, <, \geq, \leq$ *and* $\equiv$ respectively. Here is an example:

```
<apply>
  <neq/>
  <ci> x </ci>
  <ci> y </ci>
  <ci> z </ci>
</apply>
```

The above should render as $x \neq y \neq z$.

2.2.5 Calculus and vector calculus

- **Integral – `int`**

The `int` element is the operator for an integral. The lower limit, upper limit and bound variable are given by optional child elements `lowelimit`, `uplimit`, and `bvar`

respectively. The domain of integration may alternatively be specified by using the `interval` element, or by the `condition` element.

```
<apply>
  <int/>
  <bvar>
    <ci> x </ci>
  </bvar>
  <lowlimit>
    <cn> 0 </cn>
  </lowlimit>
  <uplimit>
    <cn> 1 </cn>
  </uplimit>
  <apply>
    <exp/>
    <ci> x </ci>
  </apply>
</apply>
```

```
<apply>
  <int/>
  <bvar>
    <ci> x </ci>
  </bvar>
  <interval>
    <cn> 0 </cn>
    <cn> 1 </cn>
  </interval>
  <apply>
    <exp/>
    <ci> x </ci>
  </apply>
</apply>
```

Both fragments above should render as $\int_0^1 e^x dx$.

```
<apply>
  <int/>
  <bvar>
    <ci> x </ci>
  </bvar>
  <condition>
    <apply>
      <in/>
    <ci> x </ci>
    <ci type="set"> S </ci>
  </condition>
  <apply>
    <exp/>
    <ci> x </ci>
  </apply>
</apply>
```

The above should render as $\int_{x \in S} e^x dx$.

- **Differentiation – `diff`, and partial differentiation – `partialdiff`**

The `diff` element is the differentiation operator for a function of a single real variable. The `partialdiff` element is the partial differentiation operator for function of several variables. The bound variable(s) are given by the `bvar` element. The `bvar` element may also contain a `degree` element, which specifies the order of the derivative to be taken:

```
<apply>
  <diff/>
  <bvar>
    <ci> x </ci>
  </bvar>
  <apply>
    <plus/>
    <ci> x </ci>
    <cn> 3 </cn>
  </apply>
</apply>
```

```
<apply>
  <partialdiff/>
  <bvar>
    <ci> x </ci>
    <degree>
      <cn> 3 </cn>
    </degree>
  </bvar>
  <bvar>
    <ci> y </ci>
    <degree>
      <cn> 2 </cn>
    </degree>
  </bvar>
  <apply>
    <plus/>
    <ci> x </ci>
    <ci> y </ci>
    <cn> 3 </cn>
  </apply>
</apply>
```

The above should render as:

$$\frac{d}{dx}(x+3) \qquad \frac{\partial^5}{\partial x^3 \partial y^2}(x+y+3)$$

- **Divergence – `divergence`, gradient – `grad`**

The `divergence` element is the vector calculus divergence operator. The `grad` element is the vector calculus gradient operator. Both are unary calculus operators:

```
<apply>
  <divergence/>
  <ci> x </ci>
</apply>
```

```
<apply>
  <grad/>
  <ci> x </ci>
</apply>
```

The above should render as div x and grad x respectively.

- **Laplacian – `laplacian`**

The `laplacian` element is the vector calculus laplacian operator. It is a unary operator:

```
<apply>
  <laplacian/>
  <ci type="vector"> v </ci>
</apply>
```

The above should render as $\nabla^2 v$.

2.2.6 Theory of Sets

- **Set – `set`, and list – `list`**

The `set` element is the container element that constructs a set of elements. The `list` element is the container element that constructs a list of elements. The elements of both can be defined either by explicitly listing the elements, or by using the `bvar` and `condition` elements. Lists differ from sets in that there is an explicit order to the elements. Type of ordering can be specified by the `order` attribute, which can be either `lexicographic` or `numeric`:

```
<set>
  <ci> x </ci>
  <ci> y </ci>
  <ci> z </ci>
</set>
```

```
<list order="numeric">
  <bvar>
    <ci> x </ci>
  </bvar>
  <condition>
    <apply>
      <in/>
      <ci> x </ci>
      <ci type="set"> S </ci>
    </apply>
  </condition>
  <ci> x </ci>
</apply>
```

The above should render as $\{x, y, z\}$ and $[x|x \in S]$ respectively.

- **Union – `union`, intersect - `intersect`**

The `union` element is the operator for a set union or join of two or more sets. The `intersect` element is the operator for the set intersection or meet of two or more sets. Both are *n*-ary operators:

```
<apply>
  </union>
  <ci type="set"> A </ci>
  <ci type="set"> B </ci>
  <ci type="set"> C </ci>
</apply>
```

```
<apply>
  </intersect>
  <ci type="set"> A </ci>
  <ci type="set"> B </ci>
  <ci type="set"> C </ci>
</apply>
```

This should render as $A \bigcup B \bigcup C$ and $A \bigcap B \bigcap C$ respectively.

- **Set inclusion – `in`, set exclusion – `notin`**

The `in` element is the relational operator used for set inclusion. The `notin` element is the relational operator used for set exclusion. Both are binary operators:

```
<apply>
  <in/>
  <ci> a </ci>
  <ci type="set"> A </ci>
</apply>
```

```
<apply>
  <notin/>
  <ci> a </ci>
  <ci type="set"> A </ci>
</apply>
```

The above should render as $a \in A$ and $a \notin A$ respectively.

- **Subset – `subset`, proper subset – `prsubset`**

The `subset` element is the relational operator for a set containment. The `prsubset` element is the relational operator element for set proper containment. Both are *n*-ary set relations:

```
<apply>
  <subset/>
  <ci type="set"> A </ci>
  <ci type="set"> B </ci>
</apply>
```

```
<apply>
  <prsubset/>
  <ci type="set"> A </ci>
  <ci type="set"> B </ci>
</apply>
```

The above should render as $A \subset B$ and $A \subseteq B$ respectively.

- **Set difference – `setdiff`**

The `setdiff` element is the operator for a set difference of two sets. It is a binary set operator:

```
<apply>
  <setdiff/>
  <ci type="set"> A </ci>
  <ci type="set"> B </ci>
</apply>
```

The above should render as $A \setminus B$.

- **Cardinality – `card`**

The `card` element is the operator for deriving the size or cardinality of a set. It is a unary set operator:

```
<apply>
  <card/>
  <ci type="set"> A </ci>
</apply>
```

This should render as $|A|$.

2.2.7 Sequences and Series

- **Sum – `sum`, product – `product`**

The `sum` element denotes the summation operator. The `product` element denotes the product operator. Upper and lower limits for them, as well as domains for the bound variables are specified by `uplimit`, `lowlimit` or `condtion` elements. The index for the product is specified by a `bvar` element:

```
<apply>
  <sum/>
  <bvar>
    <ci> i </ci>
  </bvar>
  <lowlimit>
    <cn> 0 </cn>
  </lowlimit>
  <uplimit>
    <ci> n </ci>
  </uplimit>
  <apply>
    <exp/>
    <ci> i </ci>
  </apply>
</apply>
```

```
<apply>
  <sum/>
  <bvar>
    <ci> i </ci>
  </bvar>
    <condition>
      <apply>
        <in/>
        <ci> i </ci>
        <ci> A </ci>
      </apply>
    </condition>
  <apply>
    <exp/>
    <ci> i </ci>
  </apply>
</apply>
```

The above should render as $\sum_{i=0}^{n} e^i$ and $\sum_{i \in A} e^i$ respectively.

If the `product` element is used instead of `sum` the rendering should be $\prod_{i=0}^{n} e^i$ and $\prod_{i \in A} e^i$ respectively.

- **Limit – `limit`, tends to – `tendsto`**

The `limit` element represents the operation of taking a limit of a sequence. The limit point is expressed by specifying a `lowlimit` and a `bvar`, or by specifying a condition on one or more bound variables. The `tendsto` element is used to express the relation that a quantity is tending to a specified value. The two elements are often used together:

```
<apply>
  <limit/>
  <bvar>
    <ci> x </ci>
  </bvar>
  <lowlimit>
    <cn> 0 </cn>
  </lowlimit>
  <apply>
    <sin/>
    <ci> x </ci>
  </apply>
</apply>
```

```
<apply>
  <limit/>
  <bvar>
    <ci> x </ci>
  </bvar>
  <condition>
    <apply>
      <tendsto type="above"/>
      <ci> x </ci>
      <cn> 0 </cn>
    </apply>
  </condition>
  <apply>
    <sin/>
    <ci> x </ci>
  </apply>
</apply>
```

The above should render as $\lim_{x \to 0} \sin x$ and $\lim_{x \downarrow 0} \sin x$ respectively.

2.2.8 Elementary classical functions

- **Trigonometric functions**

The following elements representing trigonometric functions have been defined in MathML: `sin`, `cos`, `tan`, `sec`, `csc`, `cot`, `sinh`, `cosh`, `tanh`, `sech`, `csch`, `coth`, `arcsin`, `arccos`, `arctan`, `arccosh`, `arcot`, `arccoth`, `arcsc`, `arcot`, `arccoth`, `arccsc`, `arccsch`, `arcsec`, `arcsc`, `arccsch`, `arcsec`, `arcsech`, `arsinh`, `arctanh`. They are all unary trigonometric operators.

```
<apply>
  <sin/>
  <ci> x </ci>
</apply>
```

```
<apply>
  <arcsin/>
  <apply>
    <plus/>
    <ci>x</ci>
    <cn>3</cn>
  </apply>
</apply>
```

The above should render as $\sin x$ and $\arcsin(x+3)$ respectively.

- **Exponential – `exp`, natural logarithm – `ln`**

The `exp` element represents the exponential function associated with the inverse of the natural logarithm function. The `ln` element is the natural logarithm operator. Both are unary arithmetic operators:

```
<apply>
  <exp/>
  <ci> x </ci>
</apply>
```

```
<apply>
  <ln/>
  <ci> x </ci>
</apply>
```

The above should render as e^x and $\ln x$ respectively.

- **Logarithm – `log`**

The `log` element is the operator that returns a logarithm to a given base. The base may be specified using a `logbase` element, which should be the first element following `log`. If the `logbase` is not present, a default base of 10 is assumed:

```
<apply>
  <log/>
  <logbase>
    <cn> 8 </cn>
  </logbase>
  <ci> x </ci>
</apply>
```

```
<apply>
  <log/>
  <ci> x </ci>
</apply>
```

The above should render as $\log_8 x$ and $\log x$ respectively.

2.2.9 Statistics

- **Mean – `mean`, median – `median`, mode - `mode`**

The `mean` element is the operator for a mean or average. The `median` element is the operator for the median. The `mode` operator is the element for the statistical mode. All three are *n*-ary operators:

```
<apply>
  <mean/>
  <ci> X </ci>
</apply>
```

```
<apply>
  <median/>
  <ci> X </ci>
</apply>
```

```
<apply>
  <mode/>
  <ci> X </ci>
</apply>
```

The above should be rendered as:

$\overline{X}$ median(X) mode(X)

- **Standard deviation – `sdev`, variance – `variance`**

The `sdev` element is the operator for the standard deviation. The `variance` element is the operator for the statistical variance. Both are *n*-ary operators:

```
<apply>
  <sdev/>
  <ci> X </ci>
</apply>
```

```
<apply>
  <variance/>
  <ci> X </ci>
</apply>
```

The above should render as $\sigma(X)$ and $\sigma(X)^2$ respectively.

- **Moment – `moment`**

The `moment` element represents statistical moments. The `degree` quantifier is used to specify the degree:

```
<apply>
  <moment/>
  <degree>
    <cn> 4 </cn>
  </degree>
  <ci> X </ci>
</apply>
```

The above should render as $\langle X^4 \rangle$.

2.2.10 Linear Algebra

- **Vector – `vector`**

The `vector` element is the container for a vector. The child elements are the components of the vector:

```
<vector>
  <cn> 3 </cn>
  <cn> -2 </cn>
  <cn> 9 </cn>
</vector>
```

```
<matrixrow>
  <cn> 3 </cn>
  <cn> -2 </cn>
  <cn> 9 </cn>
</matrixrow>
```

The above should render as $\begin{pmatrix} 3 \\ -2 \\ 9 \end{pmatrix}$ and $\begin{pmatrix} 3 & -2 & 9 \end{pmatrix}$ respectively.

- **Matrix – `matrix`, matrix row – `matrixrow`**

The `matrix` element is the container for matrix rows, which are represented by `matrixrow`, which in turn contain the elements of a matrix:

```
<matrix>
  <matrixrow>
    <cn> 7 </cn>
    <ci> x </ci>
  </matrixrow>
  <matrixrow>
    <cn> -1 </cn>
    <apply>
      <plus/>
      <ci> x </ci>
      <cn> 3 </cn>
    </apply>
  </matrixrow>
  <matrixrow>
    <ci> x </ci>
    <cn> 1 </cn>
  </matrixrow>
</matrix>
```

This should be rendered as $\begin{pmatrix} 7 & x \\ -1 & x+3 \\ x & 1 \end{pmatrix}$.

- **Determinant – `determinant`**

The `determinant` element is the operator for constructing the determinant of a matrix:

```
<apply>
  <determinant/>
  <ci type="matrix"> A </ci>
</apply>
```

This should render as $\det A$.

- **Transpose – `transpose`**

The `transpose` element is the operator for constructing the transpose of a matrix:

```
<apply>
  <transpose/>
  <ci type="matrix"> A </ci>
</apply>
```

The above should render as A^T.

- **Selector – `selector`**

The `selector` element is the operator for indexing into vectors, matrices and lists. It accepts one or more arguments. The first argument identifies the vector, matrix, or list from which the selection is taking place, and the second and subsequent arguments, if given, indicate the kind of selection taking place. When used with a single argument, it should be interpreted as giving the sequence of all elements in the list, vector, or matrix given. The ordering of elements in the sequence for a matrix is understood to be first by column, then by row. When three arguments are given, the last one is ignored for a list or vector, and in the case of a matrix, the second and third arguments specify the row and column of the selected element. When two arguments are given, the first is a vector or list, the second argument specifies an element in the list or vector. When a matrix and only one index i are specified, it refers to the i-th matrix row.

The `selector` element renders the same as the expression it selects. Therefore, the following fragment:

```
<apply>
  <selector/>
  <matrix>
    <matrixrow>
      <cn> 1 </cn>
      <cn> 2 </cn>
    </matrixrow>
    <matrixrow>
      <cn> 3 </cn>
      <cn> 4 </cn>
    </matrixrow>
  </matrix>
  <cn> 2 </cn>
  <cn> 1 </cn>
</apply>
```

```
<apply>
  <selector/>
  <matrix>
    <matrixrow>
      <cn> 1 </cn>
      <cn> 2 </cn>
    </matrixrow>
    <matrixrow>
      <cn> 3 </cn>
      <cn> 4 </cn>
    </matrixrow>
  </matrix>
  <cn> 1 </cn>
</apply>
```

would select

```
<cn> 3 </cn>
```

```
<matrixrow>
  <cn> 1 </cn>
  <cn> 2 </cn>
</matrixrow>
```

and would render as 3 and $\begin{pmatrix} 1 & 2 \end{pmatrix}$ respectively.

- **Vector product – `vectorproduct`, scalar product – `scalarproduct`**

The `vectorproduct` is the operator for deriving the vector product of two vectors. The `scalarproduct` is the operator for deriving the scalar product of two vectors. Both are binary vector operators:

```
<apply>
  <vectorproduct/>
  <ci type="vector"> u </ci>
  <ci type="vector"> v </ci>
</apply>
```

```
<apply>
  <scalarproduct/>
  <ci type="vector"> u </ci>
  <ci type="vector"> v </ci>
</apply>
```

This should render as $u \times v$ and $u \cdot v$ respectively.

2.2.11 Semantic Mapping Elements

- **Semantics – `semantics`, XML-based encoding – `annotation-xml`**

The `semantics` element is the container element that associates additional representation with a given MathML construct. The `semantics` element has as its first child the expression being annotated, and the subsequent children are the annotations. There is no restriction on the kind of annotation that can be attached using the semantics element. For example, one might give a TEX encoding.

An important purpose of the `semantics` construct is to associate specific semantics with a particular presentation or additional presentation information with a content construct. The default rendering of a `semantics` element is the default rendering of its first child. When a MathML-presentation annotation is provided, a MathML renderer may optionally use this information to render the MathML construct. This would typically be the case when the first child is a MathML content construct and the annotation is provided to give a preferred rendering differing from the default for the content elements.

The annotation-xml container element is used to contain representations that are XML based. It is always used together with the semantics element, and takes the attribute encoding to define the encoding being used. Here is an example:

```
<semantics>
  <mrow>
    <mi> n </mi>
    <mo> ! </mo>
  </mrow>
  <annotation-xml encoding="MathML">
    <apply>
      <factorial/>
      <ci> n </ci>
    </apply>
  </annotation-xml>
</semantics>
```

The above should render as its first child, that is $n!$

2.3 MathML Presentation Elements

MathML presentation elements are, naturally, concerned with the way an expression is rendered, and are divided into two classes. Token elements represent individual symbols, names, numbers, etc. In general, tokens can have only have characters and mchar elements as content. Layout schemata build expressions out of parts, and can only have elements as content.

All individual symbols in a mathematical expression should be represented by MathML token elements. The primary MathML token element types are identifiers, numbers, and operators. There are also token elements for representing text or white space. In MathML expressions are constructed recursively, with the layout schemata

playing the role of the expression constructors. The layout schemata specify the way in which sub-expressions are built into larger expressions.

MathML has a very rich set of presentation elements. We shall not discuss all of them here, but rather talk about those that were actually used in the process of writing this thesis (i.e. were used in the stylesheet or in the application converting TEX to MathML).

2.3.1 Token Elements

Token elements can contain any sequence of zero or more MathML characters.

- **Identifier – `mi`**

Names of all MathML presentation elements start with "m". The `mi` element represents a symbolic name or arbitrary text that should be rendered as an identifier. Identifiers can include variables, function names, and symbolic constants. Note that certain mathematical identifiers, such as primed or subscripted variables, should be represented by `msub` or `msup` instead. The `mi` element can also take the `fontstyle` attribute with two possible values, `normal` (default) or `italic`. Here is an example:

```
<mi> x </mi>
<mi> cos </mi>
```

Inside the `mi` element can be another MathML element – `mchar`, which is used to refer to non-ASCII characters:

```
<mi> <mchar name="pi"/> </mi>
```

- **Number – mn**

The mn element represents a numeric literal or other data that should be rendered as a numeric literal. Here are some examples:

```
<mn> 13 </mn>
<mn> -0.171 </mn>
<mn> 1,234,567 </mn>
<mn> 1.23e13 </mn>
<mn> 0x0A55 </mn>
```

However, in some instances numbers should be constructed using other presentation elements, such as mrow, mo and mchar in this case:

```
<mrow>
  <mn> 7 </mn>
  <mo> + </mo>
  <mrow>
    <mn> 13 </mn>
    <mo> <mchar name="InvisibleTimes"/> </mo>
    <mi> <mchar name="ImaginaryI"/> </mi>
  </mrow>
</mrow>
```

The above should render as $7+13i$.

- **Operator, fence, separator or accent – mo**

The mo element represents an operator or anything that should be rendered as an operator. In general, the notational conventions for mathematical operators are quite

complicated, and therefore MathML provides a relatively sophisticated mechanism for specifying the rendering behavior of the mo element. As a consequence, in MathML the list of things that should render as an operator includes a number of notations that are not mathematical operators in the ordinary sense. Besides ordinary operators with infix, prefix or postfix forms, these include fence characters such as braces, parentheses, absolute value bars, separators such as comma and semicolon, and mathematical accents such as a bar or tilde over a symbol.

The key feature of the mo element is that its default attribute values are set on a case-by-case basis from an operator dictionary. The default attribute values can be found in the operator dictionary and therefore need not be specified on each mo element.

There are many attributes that can be used with a mo element, including the ones applicable to most presentation elements (e.g. `fontsize`, `fontweight`, `color`), as well as specific to the mo, such as `form`, which can assume the values `prefix`, `infix`, `postfix` (the default is set by the position of operator in an mrow), `fence`, which can be `true` or `false`, and the default is set by the dictionary to `false`, `separator` (`true`, `false`, `false` by default by dictionary), `stretchy` (`true`, `false`, `false` by default by dictionary), and others.

Here are some examples:

```
<mo> + </mo>
<mo> ( </mo>
<mo fence="false" stretchy="true"> | </mo>
<mo> , </mo>
```

What is the function of the dictionary and why is it needed? For one thing, there exist many different operators, and each of them has to be rendered in a different way. For example, the spacing around comma should be different on the left and on the right, while the spacing around plus should be the same on both sides. The same is true when an operator is overloaded in the sense that it can have more than one form (prefix, infix or

postfix). For example, the minus operator can be either a prefix or infix operator. If used in the infix form, the spacing around it should be the same on both sides, while in the prefix form spacing should be added only on the left-hand side.

Therefore, the dictionary is indexed by both the contents of the mo element, and the value of the form attribute. If the form attribute is not explicitly specified and there is more than one form in the dictionary for a mo with given contents then the following heuristics can be used to infer the value of the form attribute:

- if the operator is the first argument in an mrow with more than one child then the prefix form is used;
- if it is the last argument in an mrow with more than one child then the postfix form is used;
- in all other cases the infix form is used.

Note that these rules make reference to the mrow (discussed in section 2.3.2) to which the mo element is a child. In some situations, this mrow might be an inferred mrow, as in the following example:

```
<mfenced open="[" close=")">
  <mi> a </mi>
  <mrow>
    <mi> a </mi>
    <mo> + </mo>
    <mn> 1 </mn>
  </mrow>
</mfenced>
```

In this example, there is an inferred mrow (in italics):

```
<mfenced open="[" close=")">
  <mrow>
    <mi> a </mi>
    <mrow>
      <mi> a </mi>
      <mo> + </mo>
      <mn> 1 </mn>
    </mrow>
  </mrow>
</mfenced>
```

The rendering in the first case is $[a, a+1)$. However, in the second case with the mrow inserted explicitly the comma would not be inserted: $[aa+1)$. The reason is that the mfenced element automatically puts commas in between its children. If there is an explicit mrow it means that there is only one child to the mfenced element. Therefore, the comma is not inserted. Such default behavior of the mfenced element can be overridden by explicitly specifying the separators attribute. By default, it is set to "," in the dictionary. Whatever the separator is specified by the separators attribute, the infix form of it is used because it is supposed to be inserted in between the children of mfenced.

If the operator does not occur in the dictionary with the specified form, the renderer should use one of the forms available there, in the following order: infix, postfix, prefix. If no forms are available for the given mo element content, the renderer should use the defaults given in parentheses in the table of attributes for mo.

- **Referring to non-ASCII characters – mchar**

The mchar element is used to reference characters. This provides an alternative to using entity references. The mchar element can be used inside any MathML token element.

The mchar element has one required attribute name, which is a string. The name attribute must be one of the names specified in the Chapter 6 of the MathML specification. It is an error to use a name that is not in that list. Here is an example:

```
<mi> <mchar name='omega'/> </mi>
```

The above should render as ω.

2.3.2 General Layout Schemata

- **Horizontal grouping – mrow**

The mrow element has already been mentioned a few times in previous sections. It is used to group together any number of sub-expressions. It has also been mentioned that mrow can sometimes be inferred (implicit).

One situation where mrow is especially useful is for proper grouping of sub-expressions. Generally, sub-expressions should be grouped as they are grouped in the mathematical interpretation of the expression, in other words according to the underlying syntax tree. Proper grouping allows for more intelligent line breaking. For example the expression $1-3x$ could be represented as either

```
<mrow>
  <mn> 1 </mn>
  <mo> - </mo>
  <mn> 3 </mn>
  <mo> <mchar name="InvisibleTimes"/> </mo>
  <mi> x </mi>
</mrow>
```

or

```
<mrow>
  <mn> 1 </mn>
  <mo> - </mo>
  <mrow>
    <mn> 3 </mn>
    <mo> <mchar name="InvisibleTimes"/> </mo>
    <mi> x </mi>
  </mrow>
</mrow>
```

Without the added `mrow` in the first code fragment the renderer might want to split the expression anywhere between the 3 and x, which might lead the reader of the document to believe that $1-3$ and x are two unrelated expressions.

- **Fractions – `mfrac`**

The `mfrac` element is used for fractions. It can also be used to mark up fraction-like objects such as binomial coefficients. The syntax for `mfrac` is:

```
<mfrac>
  numerator
  denominator
</mfrac>
```

There are several attributes that can be used with `mfrac`. One of them is `linethickness`. This attribute indicates the thickness of the fraction bar. If it is set to 0 then it becomes invisible, which is useful when one needs to represent binomial coefficients. Two other attributes are `numalign` and `denomalign`, which control the

horizontal alignment of the numerator and denominator. They can assume values of `left`, `center`, and `right`. The last attribute, `bevelled`, determines whether the fraction is displayed with numerator above the denominator separated by a horizontal line (e.g $\frac{a}{b}$), or a diagonal line instead (e.g. a/b). This attribute can be set to either `true` or `false`. The default is `false`.

- **Radicals – `msqrt`, `mroot`**

These elements construct radicals. The `msqrt` element is used for square roots, while the `mroot` element is used to draw radicals with indices. The syntax is as follows:

```
<msqrt>
  base
</msqrt>
```

```
<mroot>
  base
  index
</mroot>
```

The `mroot` requires exactly two arguments. The `msqrt` takes any number of arguments, which are treated as one because of the inferred `mrow`, e.g.:

```
<msqrt>
  <mi> x </mi>
  <mo> + </mo>
  <mn> 1 </mn>
</msqrt>
```

```
<mroot>
  <mrow>
    <mi> x </mi>
    <mo> + </mo>
    <mn> 1 </mn>
  </mrow>
  <mi> n </mi>
</mroot>
```

The above should render as $\sqrt{x+1}$ and $\sqrt[n]{x+1}$ respectively.

- **Content inside a pair of fences – `mfenced`**

The `mfenced` element provides a convenient form in which to express common constructs involving fences, that is braces, brackets, and parentheses, possibly involving separators between arguments.

Very often it is better to use `mfenced` than `mrow`. For example, consider the following example:

```
<mrow>
  <mo> ( </mo>
  <mi> x </mi>
  <mo> , </mo>
  <mi> y </mi>
  <mo> ) </mo>
</mrow>
```

This markup represents (x, y). The same result could be obtained by using mfenced as follows:

```
<mfenced>
  <mi> x </mi>
  <mi> y </mi>
</mfenced>
```

The `mfenced` element can be used with the following three attributes. The first two are `open` and `close`, which can assume a value of a string. They are the opening and closing fence respectively. By default `open` is set to the opening parenthesis, that is "(", and `close` to ")". The third attribute is `separators`, which can be one or more characters. By default it is a comma, that is ",". If the value of the `separators` attribute is an empty string "", no separators inserted. If it is a single character then it is

inserted between all the elements that are children of `mfenced`. If there are more than one, then the first one inserted between the first and second child of `mfenced`, the second one is inserted between the second and third child, and so on. If there are more separators than needed then extra ones are ignored. If there are fewer of them than needed then the last one is reused.

Here are some examples:

```
<mfenced open="[" close="]" separators=",;,">
  <mi> a </mi>
  <mi> z </mi>
  <mi> A </mi>
  <mi> Z </mi>
</mfenced>
```

The above should render as $[a,z;A,Z]$.

```
<mfenced close="]">
  <mn> 0 </mn>
  <mn> 1 </mn>
</mfenced>
```

The above should render as $(0,1]$.

Another fine point is that it would be incorrect to do the following:

```
<mfenced separators="+">
  <mi> x </mi>
  <mi> y </mi>
  <mn> 1 </mn>
</mfenced>
```

to represent $(x+y+1)$ because the "+" operator would be rendered as a separator instead of an infix operator (no space would be added on the left of "+"), which would be wrong.

2.3.3 Script and Limit Schemata

The elements described in this section position one or more scripts around a base. Attaching various kinds of scripts and embellishments to symbols is a very common notational device in mathematics. In addition to sub- and superscript elements, MathML has `overscript` and `underscript` elements that place scripts above and below the base. These elements can be used to place limits on large operators, or for placing accents and liens above or below the base. The rules for rendering accents differ from those for `overscript` and `underscript`, and this difference can be controlled with the accents and `accentunder` attributes.

- **Subscript – msub, superscript – msup**

The syntax for the `msub` and `msup` elements is:

```
<msub>
  base
  subscript
</msub>
```

```
<msup>
  base
  superscript
</msup>
```

Here are some examples:

```
<msub>
  <mi> x </mi>
  <mi> i  </mi>
</msub>
```

```
<msup>
  <mi> x </mi>
  <mrow>
    <mi> n </mi>
    <mo> + </mo>
    <mn> 1 </mn>
  </mrow>
</msup>
```

The above code fragments should render as x_i and x^{n+1} respectively.

- **Subscript-superscript pair – `msubsup`**

The `msubsup` element is used to attach both a subscript and superscript to a base expression. Note that both scripts are positioned tight against the base. The syntax for the msubsup element is:

```
<msubsup>
  base
  subscript
  superscript
</msubsup>
```

Note that the rendering will be different if `msubsup` is used instead of a combination of `msup` and `msub` elements:

```
<msubsup>
  <mi> x </mi>
  <mi> i </mi>
  <mn> 2 </mn>
</msubsup>
```

```
<msup>
  <msub>
    <mi> x </mi>
    <mi> i </mi>
  </msub>
  <mn> 2 </mn>
</msup>
```

The above should render as x_i^2 and $x_i^{\,2}$ respectively.

- **Underscript – `munder`, overscript – `mover`**

The syntax for the `munder` and `mover` elements is as follows:

```
<munder>
  base
  underscript
</munder>
```

```
<mover>
  base
  overscript
</mover>
```

Here are some examples:

```
<munder>
  <mrow>
    <mi> x </mi>
    <mo> + </mo>
    <mi> y </mi>
  </mrow>
  <mo>
    <mchar name="UnderBrace"/>
  </mo>
</munder>
```

```
<mover>
  <mrow>
    <mi> x </mi>
    <mo> + </mo>
    <mi> y </mi>
  </mrow>
  <mo>
    <mchar name="OverBar"/>
  </mo>
</mover>
```

The above should render as $\underbrace{x+y}$ and $\overbrace{x+y}$ respectively.

- **Underscript-overscript pair – `munderover`**

The syntax for the `munderover` element is:

```
<munderover>
  base
  underscript
  overscript
</munderover>
```

The munderover element is used so that underscript and overscript are vertically spaced equally in relation to the base and so that they follow the slant of the base as shown below:

```
<munderover>
  <mo> <mchar name="int"/> </mo>
  <mn> 0 </mn>
  <mi> x </mi>
</munderover>
```

The above should render as $\int_0^x$. Note again that the combination of munder and mover is not the same as a single munderover.

- **Prescripts and tensor indices – mmultiscripts**

The syntax for the mmultiscripts is:

```
<mmultiscripts>
  base
  ( subscript superscript )*
  [ <mprescripts/> ( presubscript presuperscript )* ]
</mmultiscripts>
```

Pre- and subscripts and tensor notations are represented by a single element, mmultiscripts. This element allows the representation of any number of vertically aligned pairs of subscripts and superscripts attached to one base. It supports both postscripts and prescripts. Missing scripts can be represented by the empty element none. The prescripts are optional and when present are given after the postscripts. The

total number of arguments must be odd if `mprescripts` is not given, or even otherwise. Here are some examples:

```
<mmultiscripts>
  <mi> C </mi>
  <mi> n </mi>
  <none/>
  <mprescripts/>
  <mn> 3 </mn>
  <none/>
</mmultiscripts>
```

```
<mmultiscripts>
  <mi> F </mi>
  <mi> i </mi>
  <mi> k </mi>
  <mi> j </mi>
  <none/>
</mmultiscripts>
```

The above should render as ${}_3C_n$ and F_{ij}^{k} respectively.

2.3.4 Tables and Matrices

Matrices, arrays and other table-like mathematical notation are marked up using `mtable`, `mtr` and `mtd` elements.

- **Table or matrix – `mtable`**

A matrix or table is specified using the `mtable` element. Inside the `mtable` element only the `mtr` element may appear. Table rows that have fewer columns than other rows of the same table are effectively padded on the right with empty `mtd` elements so that the number of columns in each row equals the maximum number of columns in any row of the table.

The `mtr` element represents one row in a table or matrix. A `mtr` element is only allowed as a direct sub-expression of an `mtable` element, and specifies that its contents

should form one row of the table. Each argument of `mtr` is placed in a different column of the table, starting at the leftmost column.

The `mtd` element represents one entry in a table or matrix. A `mtd` element is only allowed as a direct sub-expression of an `mtr` element. The `mtd` element accepts any number of elements. If it is other than one then its contents are treated as a single inferred `mrow`.

Here is an example of a matrix:

```
<mfenced>
  <mtable>
    <mtr> <mi> x </mi> <mn> 2 </mn> <mn> 3 </mn> </mtr>
    <mtr> <mn> 4 </mn> <mi> y </mi> <mn> 6 </mn> </mtr>
    <mtr> <mn> 7 </mn> <mn> 8 </mn> <mi> z </mi> </mtr>
  </mtable>
</mfenced>
```

The above should render as $\begin{pmatrix} x & 2 & 3 \\ 4 & y & 6 \\ 7 & 8 & z \end{pmatrix}$.

Chapter 3

An Overview of XSLT

Now that we have considered in detail how MathML objects can be encoded using both content and presentation markup we are ready to look at how they can be transformed. One way to accomplish such transformation is by applying a stylesheet, which is very common with markup languages like XML and HTML. Cascading Stylesheets (CSS) are commonly used with HTML while XSLT (eXstensible Stylesheet Language Transformations), was specifically designed for use with XML documents.

In this chapter we shall look at what an XSLT stylesheet is, its format, and most importantly, how to write templates that form the XSLT stylesheet. To be able to completely understand how things work we shall also discuss XPath – the language that is used to access nodes in the input tree.

Having done that, we shall finally be ready to discuss the stylesheet that transforms MathML documents encoded using content markup into presentation markup.

3.1 Structure of a Stylesheet and Top Level Elements

Every XSLT stylesheet must have the `xsl:stylesheet`, or `xsl:transform` element as the topmost element. Both are equivalent. The two different names reflect the fact that a XSLT stylesheet defines a transform of a source tree into the result tree. Just as in case of HTML and other markup languages, it is very natural to view an XML (or MathML) document as a tree with a single root. Each element in the XSLT stylesheet, including the topmost element (`stylesheet` or `transform`) is supposed

to be prefixed. In this case the prefix is `xsl`, and it must be defined as `http://www.w3.org/1999/XSL/Transform`. XSLT processors must use the XML namespaces mechanism to recognize elements and attributes from this namespace. Elements from the XSLT namespace are recognized only in the stylesheet and not in the source document.

There are several attributes that the `xsl:stylesheet` element can have. The most important of them is the `version` attribute, indicating the version of XSLT that the stylesheet requires. At this point it has to be 1.0, as it is the only version of XSLT existing to date.

Another attribute that can be used with `xsl:stylesheet` element is the optional `id` attribute. It can be used to assign a unique id to the stylesheet.

Also, the `xsl:stylesheet` element would normally have definitions for other namespaces used in the stylesheet. Here is an example:

```
<xsl:stylesheet id="http://www.scl.csd.uwo.ca/mmlidgen.xsl"
           version="1.0"
         xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
         xmlns:mml="http://www.w3.org/1998/Math/MathML">

  <!-- body of stylesheet -->

</xsl:stylesheet>
```

In the example above two namespaces are defined: `xsl` and `mml`.

The `xsl:stylesheet` may contain the following elements:

- `xsl:output`
- `xsl:strip-space`
- `xsl:variable`
- `xsl:param`

- `xsl:template`
- `xsl:import`
- `xsl:include`
- `xsl:preserve-space`
- `xsl:key`
- `xsl:decimal-format`
- `xsl:namespace-alias`
- `xsl:attribute-set`

We shall only talk about the first five of them as they were used in the stylesheet that will be discussed in the next chapter.

The first element, `xsl:output` allows stylesheet authors to specify how they want the result tree to be output. The `xsl:output` element can have a number of optional attributes. The `method` attribute can assume one of the three values: `xml`, `html`, or `text`. The first two ensure that the output conforms to the XML or HTML syntax. If the output method is set to `text` then the output can be any sequence of characters.

Another useful option is given by the `indent` attribute. It can assume values of `yes` or `no`. It gives the author control over indentation of the result tree. Here is an example of indented output:

```
<mrow>
   <mo> <mchar name="forall"/> </mo>
   <mi>x</mi>
   <mo>:</mo>
   <mrow>
      <mrow>
         <mi>x</mi>
         <mo>-</mo>
         <mi>x</mi>
      </mrow>
      <mo>=</mo>
      <mn>0</mn>
   </mrow>
</mrow>
```

The above is the MathML encoding for $\forall x, x - x = 0$. Note how the markup is indented due to the following declaration in the stylesheet:

```
<xsl:output method="xml" indent="yes"/>
```

After the tree of a source document has been constructed, but before it is otherwise processed by an XSLT processor, some text nodes are stripped. The `xsl:strip-space` element must have one single attribute called `elements`. This attribute specifies the names of the elements in the input tree that have to be white space stripped. Just like in XML, white space characters are space (ASCII #x20), tabulation (#x09), line feed (#x0A) and carriage return (#x0D). For example, the following declaration

```
<xsl:strip-space elements="ci cn"/>
```

ensures that any white space characters found inside the `ci` and `cn` elements will not be carried into the output tree. Consider the following MathML fragment:

```
<ci>
  x
</ci>
<cn> 13 </cn>
```

Without the above declaration this would be transformed into:

```
<mi>
  x
</mi>
<mn> 13 </mn>
```

If however the `xsl:strip-space` declaration is present, the output becomes:

```
<mi>x</mi>
<mn>13</mn>
```

In the `ci`, the new line character and the spaces in front of *x* have been stripped, and in the `cn` the two space characters surrounding 13 have been stripped.

Another important XSLT element is `xsl:variable`. In XSLT, the value to which a variable is bound can be of any of the types returned by expressions, such as an integer, a string, a node set, etc. In XSLT, once variable is bound, its value cannot change, so in some sense it is rather a constant than a variable. Here are some examples of how a variable can be bound to a value:

```
<xsl:variable name="var1" select="13"/>
<xsl:variable name="var2" select="var1 + 4"/>
<xsl:variable name="var3" select="'var1'"/>
```

Note that in the examples given above var1=13, var2=17, var3=var1. That is the type of the first two variables is integer, while the last one is a string.

A slightly different and more interesting element is `xsl:param`. This element behaves more like variable. However, the only way to change the value given to a parameter is by means of recursion. If `xsl:param` is used as a top-most element in the stylesheet then its value can be also set by the application processing the stylesheet. The difference between `xsl:variable` and `xsl:param` is that the latter can assume some default value. This can be very convenient in many situations. For example, if the application using the stylesheet does not specify a value for the stylesheet parameter, the default value for that parameter is assumed. We shall come across this element later. Syntactically it is very similar to `xsl:variable`:

```
<xsl:param name="par" select="2"/>
```

Finally, we are ready to discuss the most important of XSLT elements – `xsl:template`.

3.2 Templates and XPath

XSLT uses the expression language defined by XPath. Expressions are used in XSLT for a variety of purposes, including selecting nodes for processing, specifying conditions for different ways or processing the node, and generating parts of the output tree. Expressions occur as the value of certain attributes on XSLT elements, particularly in the match attribute of the `xsl:template` element. The expression in the match attribute of `xsl:template` element identifies the set of nodes in the input tree that this template will match. Here are examples of XPath expressions:

`/`	Matches the root node
`*`	Matches any element
`text()`	Matches any text node
`@*`	Matches any attribute
`.`	Matches the context (current) node
`..`	Matches the parent node
`ci`	Matches every `ci` element
`ci[1]`	Matches every `ci` element that is the first child of its parent
`*[position()>3]`	Matches every element that is at position 4, 5, etc. relative to its parent
`integral/bvar`	Matches every `bvar` element with `integral` as its parent
`integral/bvar[1]`	Matches every `bvar` element that is the first child of the `integral` element
`integral[bvar]`	Matches every `integral` element with `bvar` child
`ci[@type]`	Matches every `ci` element with type attribute
`ci[@type='fn']`	Matches every `ci` element with type attribute with value `fn`
`../*[position()=3]`	Matches a sibling at position 3 (relative to parent)
`*[self::ci]`	Matches every `ci` element
`integral//ci`	Matches every `ci` element with ancestor `integral` element

Note that in XPath there is more than one way to express the same relationship. Any of the expressions given above could be in the match attribute of the `xsl:template` element. Here is an example:

```
<xsl:template match="ci[@type='fn']">
  ...
</xsl:template>
```

The above template will match every `ci` node with the `type` attribute that has value `fn`. That is, the following node would be matched:

```
<ci type="fn"> F </ci>
```

Note that it is possible that more than one template matches the node in the source tree. In fact, it is almost always the case, because XSLT has a number of built-in templates. In particular, there are the templates that match the root node (i.e. /), every text node (i.e. `text()`) and every non-text node (i.e. * or `node()`). If it happens that more than one template matches the node, then conflict resolution rules apply. These rules may be quite complicated, and the interested reader might wish to consult section 5.5 of the XSLT 1.0 specification. Most of the time, however, the XSLT processor simply chooses the most specific one. For instance, let us assume that there are two more templates in the stylesheet:

```
<xsl:template match="ci">
  ...
</xsl:template>

<xsl:template match="ci[@type]">
  ...
</xsl:template>
```

Thus, there are total of four templates that match the node given above. The fourth template is the default template matching every node (i.e. *). Because the very first template is the most specific of the four, it will be given preference by a XSLT processor.

After a template is matched, it is applied, or instantiated. That is the markup (excluding XSLT processing instructions) inside the chosen template is copied into the result tree. To illustrate this, let us consider another example. Let us assume we have the following template:

```
<xsl:template match="interval">
  <mfenced separators=",">
    <xsl:if test="@closure='closed' or @closure='closed-open'">
      <xsl:attribute name="open">[</xsl:attribute>
    </xsl:if>
    <xsl:if test="@closure='closed' or @closure='open-closed'">
      <xsl:attribute name="close">]</xsl:attribute>
    </xsl:if>
    <xsl:apply-templates select="*"/>
  </mfenced>
</xsl:template>
```

Let us also assume that we have the following node in the source tree:

```
<interval closure="open-closed">
  <cn> 0 </cn>
  <cn> 1 </cn>
</interval>
```

The above template will match this node, and first the `mfenced` element will be inserted into the result tree. Then the XSLT processor will find the instruction that it has to process, because it is prefixed by the `xsl` prefix. The first instruction is the `xsl:if` instruction. It must have the `test` attribute with an XPath expression as its value. This expression will be evaluated, and depending on whether it evaluates to true or false the

elements inside the `xsl:if` element will or will not be processed. In this example the condition of the first `xsl:if` evaluates to false, while the condition of the second `xsl:if` is true. Therefore, the XSLT processor will go on to processing the `xsl:attribute` instruction inside that `xsl:if`. The result of it will be the addition of the `close` attribute with the value of "]" (not including the quotes) to the `mfenced` element.

The next instruction that will be processed will be `xsl:apply-templates`. It instructs the XSLT processor to continue applying templates that are specified by the `select` attribute of the `xsl:apply-templates` instruction. If no `select` attribute is given the templates will be applied to every child node of the current (or context) node, including text nodes. The current, or context node is the node of the source tree that was matched by the template. Therefore, the `xsl:apply-templates` instruction will make sure that the two `cn` elements given in the example above will be processed by matching and applying to them the appropriate templates. To complete the example above, let us assume that the following template exists:

```
<xsl:apply-templates match="cn">
  <mn>
    <xsl:apply-templates/>
  </mn>
</xsl:apply-templates>
```

This template will match every `cn` element. Then it will insert the `mn` element into the result tree, and then will invoke the template that would process the children of the matched `cn`. Normally, `cn` elements contain text (e.g. numbers 0 and 1 as in our example). Because we did not supply a template for handling text, the default template will be used. By default the text is simply copied into the result tree. Therefore, the text from the `cn` elements in the source tree will appear inside the `mn` elements in the result tree.

Thus, the complete fragment of the result tree will look like this:

```
<mfenced separators="," close="]">
  <mn> 0 </mn>
  <mn> 1 </mn>
</mfenced>
```

So far we have discussed only one attribute of `xsl:template` – `match`. Another useful attribute that can be used with the `xsl:template` is the `mode` attribute. When present, the template is matched if and only if there is a `xsl:apply-templates` instruction with the `mode` attribute that has the same value as the mode attribute of the template being matched. This is particularly useful for creating loops. This normally involves passing a parameter, which is incremented with each iteration. The parameter has to be declared and bound to a default value at the very beginning of a template. The default value of this parameter will be used if the template has not been supplied a value for this parameter from where it was called. If a template is to be called with a parameter, the `xsl:with-param` instruction is used. It has to be a child of the `xsl:apply-templates` instruction, like in the example below:

```
<xsl:apply-templates match="*" mode="myLoop">
  <xsl:param name="myParam" select="0"/>
  ...
  <xsl:apply-templates select="." mode="myLoop">
    <xsl:with-param name="myParam" select="$myParam + 2"/>
  </xsl:apply-templates>
</xsl:apply-templates>
```

Note that the name of the parameter being passed is specified in the `name` attribute of the `xsl:with-param` instruction, while the value for that parameter is specified in the `select` attribute. Whenever a variable or parameter is referenced, its name has to be preceded with the dollar character `$`.

Yet another useful XSLT instruction is `xsl:copy-of`. It can be used to copy elements of the source tree directly into the output tree. The nodes to be selected are specified in the select attribute, for example:

```
<xsl:copy-of select="*[2]"/>
```

In this example the second child of the current node will be copied through.

There are many other XSLT instructions, as well as attributes that can be used with the instructions that we have discussed in this section. However, we shall not mention them here because they do not appear in the stylesheet that will be discussed in the following chapter. The interested reader is advised to consult the complete XSLT specification.

Chapter 4

Rendering Content MathML as Presentation MathML

The stylesheet that will be discussed in this chapter is the one given in the Appendix A. In fact, there are two stylesheets. The second stylesheet only generates ids on every node of a MathML document. Ids are needed for some of the modes of the main stylesheet when it generates cross-references to the elements in the original document. Id generation on the nodes of the source document is not part of the MathML content to MathML presentation transformation, and for that reason this functionality was moved to a separate stylesheet, which can be found in Appendix B.

4.1 Layout of the Stylesheet

The main stylesheet starts with the following declaration, meaning that the stylesheet itself is a valid XML document:

```
<?xml version="1.0"?>
```

The body of the stylesheet starts with the usual `xsl:stylesheet` element:

```
<xsl:stylesheet id="http://www.scl.csd.uwo.ca/mml2mml.xsl"
          version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:mml="http://www.w3.org/1998/Math/MathML"
            xmlns="http://www.w3.org/1998/Math/MathML">
```

The compulsory `version` attribute says that the stylesheet requires a XSLT version 1.0 compliant processor (or later). The `id` attribute gives a unique id to the stylesheet. The following two attributes define the namespaces for prefixes `xsl` and `mml`. The first of the two prefixes is used with the XSLT instructions throughout the stylesheet. For example, the `xsl:stylesheet` element itself has the `xsl` prefix. Whenever an XSLT processor encounters a prefixed element it forms the complete name for that element by combining the prefix with the name of the element, and then checks if the element with that name is in its dictionary. If it is then it is a valid XSLT instruction, and it can be processed accordingly.

The next declaration is

```
<xsl:output method="xml" indent="yes"/>
```

This declaration controls the way the output is generated. Because every MathML document is also an XML document, the `method` attribute has to be set to `xml`, or otherwise there is a possibility that the output might not be valid MathML. The `indent` attribute is not as crucial as it only controls the visual appearance of the output. Most people agree that it is easier to read code that is indented.

The next declaration, `xsl:strip-space`, specifies the elements that need to be space stripped before they are otherwise processed. The reason for that was explained in Section 3.1.

The block of instructions that follows contains declarations of the six constants:

```
<xsl:variable name="SEM_STRIP" select="-1"/>
<xsl:variable name="SEM_PASS" select="0"/>
<xsl:variable name="SEM_TOP" select="1"/>
<xsl:variable name="SEM_ALL" select="2"/>
<xsl:variable name="SEM_XREF" select="3"/>
<xsl:variable name="SEM_XREF_EXT" select="4"/>
```

Each of the above constants is a unique integer, and each corresponds to a particular mode in which the stylesheet can work. All six modes shall be discussed in detail in the next section.

The next instruction is a parameter declaration:

```
<xsl:param name="SEM_SW" select="SEM_PASS"/>
```

This parameter, called SEM_SW, controls the output of the stylesheet. The value of this parameter can be set externally by the application that uses the stylesheet. If it is not explicitly specified, the default value equal to the value of the constant SEM_PASS (which is defined to be 0) is assumed.

The next block contains definitions for operator precedences:

```
<xsl:variable name="NO_PREC" select="0"/>
<xsl:variable name="UNION_PREC" select="1"/>
<xsl:variable name="SETDIFF_PREC" select="1"/>
<xsl:variable name="INTERSECT_PREC" select="3"/>
<xsl:variable name="OR_PREC" select="5"/>
<xsl:variable name="XOR_PREC" select="5"/>
<xsl:variable name="AND_PREC" select="7"/>
<xsl:variable name="PLUS_PREC" select="9"/>
<xsl:variable name="MINUS_PREC" select="9"/>
<xsl:variable name="MUL_PREC" select="11"/>
<xsl:variable name="DIV_PREC" select="11"/>
<xsl:variable name="NEG_PREC" select="13"/>
<xsl:variable name="FUNCTION_PREC" select="99"/>
```

A particular level of precedence has to be assigned to every operator, so that when the presentation markup is generated, parentheses can be inserted where appropriate. More on the issue of parenthesizing in Section 4.3.

The rest of the stylesheet contains templates. All the templates are grouped into the following categories:

- The topmost element – `math`
- Semantics handling
- Basic container elements
- Basic content elements
- Arithmetic, algebra and logic
- Relations
- Calculus
- Set theory
- Sequences and series
- Trigonometry
- Statistics
- Linear algebra

These categories correspond to those discussed in Section 2.2. The only element that has not been mentioned so far is the `math` element. The reason is that it has nothing to do with semantics or presentation. The sole purpose of this element is to provide the interface with other markup languages, such as HTML or XML. All MathML markup always has to be inside the `math` element. Therefore, the `math` element is the starting point, and all the processing in the stylesheet starts with the template matching this element.

We can now proceed to the discussion of the six modes in which the stylesheet can work.

4.2 The Six Modes of Operation

Each of the six modes differs from the others in the way semantics is handled. Apart from that, the presentation markup that is generated is the same for all six modes. Let us now consider each of them.

4.2.1 Stripping Semantics off

In this mode any semantics present in the source MathML markup is stripped. The output generated in this case will only have MathML presentation elements. If any `semantics` elements are present in the input they will be removed. Let us consider the following example. Here is a fragment of MathML markup that is to be transformed:

```
<apply>
  <tan/>
  <ci> x </ci>
</apply>
<semantics>
  <mfrac>
    <mi> x </mi>
    <mi> N </mi>
  </mfrac>
  <annotation-xml encoding="MathML">
    <apply>
      <divide/>
      <ci> x </ci>
      <ci> N </ci>
    </apply>
  </annotation-xml>
</semantics>
```

If the stylesheet is applied to this input the output would be

```
<mrow>
  <mo> tan </mo>
  <mi> x </mi>
</mrow>
<mrow>
  <mi> x </mi>
  <mo> / </mo>
  <mi> N </mn>
</mrow>
```

In other words, the stylesheet would process the MathML content markup present in the source document, and would generate output containing pure presentation markup, with no semantics. This mode is useful when the user of the document wants to get rid of the presentation suggested by the author of the document, and semantics is not needed in the output.

4.2.2 Passing Semantics 'As Is'

When used in this mode, the stylesheet will generate presentation markup without adding new content markup to the generated output. However, if the `semantics` element is found in the source document, the content markup contained in it will be preserved. Continuing with the example from the previous section, the output would be

```
<mrow>
  <mo> tan </mo>
  <mi> x </mi>
</mrow>
<semantics>
  <mrow>
    <mi> x </mi>
    <mo> / </mo>
    <mi> N </mn>
  </mrow>
  <annotation-xml encoding="MathML">
    <apply>
      <divide/>
      <ci> x </ci>
      <ci> N </ci>
    </apply>
  </annotation-xml>
</semantics>
```

4.2.3 Adding Semantics at the Top Level Only

In this mode, the content markup will be attached at the top level only. To clarify this let us again look at the following code fragment:

```
<math>
  <apply>
    <tan/>
    <ci> x </ci>
  </apply>
  <apply>
    <divide/>
    <ci> x </ci>
    <ci> N </ci>
  </apply>
</math>
```

After the stylesheet is applied to this content markup the output will look like this:

```
<math>
  <semantics>
    <mrow>
      <mrow>
        <mo> tan </mo>
        <mi> x </mi>
      </mrow>
      <mrow>
        <mi> x </mi>
        <mo> / </mo>
        <mi> N </mn>
      </mrow>
    </mrow>
    <annotation-xml encoding="MathML">
      <apply>
        <tan/>
        <ci> x </ci>
      </apply>
      <apply>
        <divide/>
        <ci> x </ci>
        <ci> N </ci>
      </apply>
    </annotation-xml>
  </semantics>
</math>
```

The original content markup now appears inside the `annotation-xml` element. This mode is useful when the whole MathML content-encoded fragment is manipulated as a whole. The obvious drawback of this approach is that it would be impossible to select a sub-expression of a formula while preserving the semantics of the selection. To be able to do that one of the other modes (considered in the following sections) has to be used.

4.2.4 Adding Semantics at All Levels

Big advantage of this mode is that every sub-expression of the output document can be manipulated independently (e.g. selected, pasted) without loss of its semantics. Again, let us consider an example. Here is a fragment of MathML content markup:

```
<apply>
  <divide/>
  <ci> x </ci>
  <ci> N </ci>
</apply>
```

After the stylesheet is applied the output will look like this:

```
<semantics>
  <mrow>
    <semantics>
      <mi> x </mi>
      <annotation-xml encoding="MathML">
        <ci> x </ci>
      </annotation-xml>
    </semantics>
    <mo> / </mo>
    <semantics>
      <mi> N </mi>
      <annotation-xml encoding="MathML">
        <ci> N </ci>
      </annotation-xml>
    </semantics>
  </mrow>
  <annotation-xml encoding="MathML">
    <apply>
      <divide/>
      <ci> x </ci>
      <ci> N </ci>
    </apply>
  </annotation-xml>
</semantics>
```

Apparently, the greater flexibility comes at the price of larger size, as now semantics has to be attached to each node of the input document. The size of the output grows quadratically. It is easy to see that there is redundancy in the data. In general, for any node in the source tree its content encoding will appear exactly n times if that node is at the n-th level in the source tree. There is a way to eliminate this redundancy. This can be done by generating cross-references between the presentation markup and the original content markup.

4.2.5 Adding Semantics at Top Level Only with Cross-References

In this mode, the stylesheet adds semantics at top level only. However, the presentation markup that is generated by the stylesheet contains cross-references to the corresponding content markup. In a sense, there are two parallel encodings: the original content encoding, and the presentation encoding. One important thing to note here is that every node in the source document has to be tagged with an id prior to the actual transformation. Therefore, two "passes" are required. It is fairly easy to put an `id` attribute on each node in a document without otherwise modifying it. This can be done with the stylesheet that can be found in Appendix B.

To illustrate the idea let us again consider an example. Let us suppose that originally we have the following content markup:

```
<math>
  <apply>
    <tan/>
    <ci> x </ci>
  </apply>
  <apply>
    <divide/>
    <ci> x </ci>
    <ci> N </ci>
  </apply>
</math>
```

After the first pass, each node will have an `id` attribute attached to it:

```
<math>
  <apply id="b2ab1b1">
    <tan id="b2ab1b1b1"/>
    <ci id="b2ab1b1b3"> x </ci>
  </apply>
  <apply id="b2ab1b3">
    <divide id="b2ab1b3b1"/>
    <ci id="b2ab1b3b3"> x </ci>
    <ci id="b2ab1b3b5"> N </ci>
  </apply>
</math>
```

After the actual transformation the result would be:

```
<semantics>
  <mrow>
    <mrow xref="b2ab1b1">
      <mo xref="b2ab1b1a">tan</mo>
      <mo> <mchar name="ApplyFunction"/> </mo>
      <mi xref="b2ab1b1b1">x</mi>
    </mrow>
    <mrow xref="b2ab1b3">
      <mi xref="b2ab1b3b3">x</mi>
      <mo>/</mo>
      <mi xref="b2ab1b3b5">N</mi>
    </mrow>
  </mrow>
  <annotation-xml encoding="MathML">
    <apply id="b2ab1b1">
      <tan id="b2ab1b1a"/>
      <ci id="b2ab1b1b1">x</ci>
    </apply>
    <apply id="b2ab1b3">
      <divide id="b2ab1b3b1"/>
      <ci id="b2ab1b3b3">x</ci>
      <ci id="b2ab1b3b5">N</ci>
    </apply>
  </annotation-xml>
</semantics>
```

Apparently, this is the most elegant and practical solution. Now there is a single instance of the original content markup, so there is no redundancy that we saw in the previous section. It is easy to see how this works. For example, if you want to select the whole expression x / N drawn on the screen then you will be selecting the `mrow` element with the `xref="b2ab1b3"`. This means that the corresponding semantics for this expression is contained in the element with that same id, that is `id="b2ab1b3"`. The element with that id can be easily retrieved, and combined with the presentation markup into a single object (by means of the `semantics` element), which then can be pasted into any other application, and without loss of its semantic meaning.

4.2.6 Generating Cross-References with no Semantics

Finally, there is the mode in which cross-references are generated just like in the previous case, but the difference is that the original content markup is not appended to the presentation markup. Therefore, the generated `xrefs` now point to the nodes in the external document. Hence, the output would look like this:

```
<mrow xref="b2ab1b1">
  <mo xref="b2ab1b1a">sin</mo>
  <mo> <mchar name="ApplyFunction"/> </mo>
  <mi xref="b2ab1b1b1">x</mi>
</mrow>
<mrow xref="b2ab1b3">
  <mi xref="b2ab1b3b3">x</mi>
  <mo>/</mo>
  <mi xref="b2ab1b3b5">N</mi>
</mrow>
```

This is the most efficient approach of all considered so far. However, with this approach the client application has to be aware that the references are made to the external document, and it has to be able to retrieve that document when needed.

4.3 Fine Points of Generating MathML Presentation

We are now finally ready to discuss the most interesting part of this thesis – the fine points of rendering with MathML presentation markup. There are several of them, and we shall start with the issue of operator precedence and parenthesizing.

4.3.1 Operator Precedence and Parenthesizing

When a mathematical object is encoded using MathML content markup, there is no explicit information in the markup about operator precedence. Naturally, any expression encoded with MathML can be viewed as a tree. The position of an operator in the expression tree implies the order of its evaluation. Therefore, neither explicit information about operator precedence, nor parentheses are needed. However, when the presentation markup is generated, parentheses have to be inserted in appropriate places. To clarify the point let us consider the following example:

```
<apply>
  <divide/>
  <apply>
    <plus/>
    <ci> a </ci>
    <ci> b </ci>
  </apply>
  <cn> 2 </cn>
</apply>
```

The above is the MathML content encoding for the expression $(a+b)/2$. After the stylesheet is applied to this source document, the output should look like this:

```
<mrow>
  <mfenced open="(" close=")" separators="">
    <mi> a </mi>
    <mo> + </mo>
    <mi> b </mi>
  </mfenced>
  <mo> / </mo>
  <mn> 2 </mn>
</mrow>
```

If the parentheses were not inserted then the rendering would be wrong, as it would be

```
<mrow>
  <mrow>
    <mi> a </mi>
    <mo> + </mo>
    <mi> b </mi>
  </mrow>
  <mo> / </mo>
  <mn> 2 </mn>
</mrow>
```

which would render as $a+b/2$, and that would be wrong.

Therefore, each of the MathML operators has to be assigned a certain level of precedence. Consequently, when the transformation is being performed, the precedence of current node has to be compared to the precedence of the parent node, and if the current node has lower precedence, then the current sub-expression has to be parenthesized. This was the case in the example given earlier in this section. In this stylesheet operator precedences are assigned to operators in the same way as they are

assigned in Java. Section 4.1 has the block of code taken from the stylesheet that shows concrete assignments of the level of precedence to each MathML operator.

Another subtle point concerns parenthesizing of negative numbers. There are several situations where negative numbers should not be parenthesized. For example, if a negative number appears in plain text it normally should not be parenthesized. It is much harder to decide whether to parenthesize a negative number or not when it occurs in a mathematical expression. Here are some examples: $x(-5)$, $-5x$, $y/(-5)x$, $y/(-5x)$, $y+(-5)x$, $\sqrt{-5x+y}$. One way of handling the situation would be to blindly parenthesize negative numbers in all situations, which is, obviously, not a very good solution. Alternatively, special steps have to be taken to prevent parenthesizing of negative numbers where it is not needed. While the problem at hand seems trivial, the solution to it is not so simple for several reasons. First, there are several tricky situations, in particular when a negative number is deeply buried in the expression tree as in the following example:

```
<apply>
  <minus/>
  <apply>
    <plus/>
    <apply>
      <multiply/>
      <cn> -3 </cn>
      <ci> x </ci>
    </apply>
    <ci> y </ci>
  </apply>
  <cn> 8 </cn>
</apply>
```

The above is the MathML encoding for the expression $-3x+y-8$. If this whole expression is a standalone expression, that is it is not a part of a larger expression, then the negative number -3 should not be parenthesized. If however it is a part of a larger

expression, then -3 might need parentheses, or it might not. For example, in the following two situations it will not need parentheses: $2/(-3x+y-8)$, and $f(-3x+y-8)$, while in the following situation parentheses are needed: $2+(-3)x+y-8$. (Note that the latter is not simplified to $2-3x+y-8$, but rather rendered exactly the way it was encoded, because in certain situations the author might want to keep the minus in front of a term, like in $5+(-2)=5-2$). The problem stems from the fact that -3 is the leftmost leaf in the expression tree, and it can be buried arbitrarily deeply. The decision as to whether parentheses should be added or not around a node has to be made when the node is visited. To be able to do that, information about the context in which the sub-expression is used has to be passed down the tree so that when the node representing the negative number is visited, it could decide whether to parenthesize itself or not. While tricky and cumbersome, this solution was successfully implemented in XSLT, so that the stylesheet correctly handles all of the above-mentioned situations.

4.3.2 Output Optimization

In this section we shall talk about how to optimize the output produced by a stylesheet. We are talking here about optimality in terms of size of the output. The goal is to produce the smallest output possible while preserving correct rendering of the markup. Why is this important? One reason is reduced storage requirements. Smaller size would also mean reduced download times. Analog V.34 and V.90 modems (33.6kbps and 56kbps) are still very common and a 20% reduction in size of a file can lead to substantial improvements of the download time.

Although there are a few little tricks leading to the reduction of the output size, the major contribution comes from eliminating the unneeded `mrow` elements, which are

probably the most common of MathML presentation elements. There are many situations where unneeded mrows may appear. Let us consider the following example:

```
<apply>
  <exp/>
  <ci> x </ci>
</apply>
```

```
<apply>
  <exp/>
  <apply>
    <plus/>
    <ci> x </ci>
    <ci> y </ci>
  </apply>
</apply>
```

The above content markup will be transformed into the following presentation markup:

```
<msup>
  <mo> e </mo>
  <mi> x </mi>
</msup>
```

```
<msup>
  <mo> e </mo>
  <mrow>
    <mi> x </mi>
    <mo> + </mo>
    <mi> y </mi>
  </mrow>
</msup>
```

The msup presentation element requires exactly two children. If the number of children is other than two it is a MathML error. Therefore, the easiest and safest way to generate the presentation markup in this case would be to blindly put the mrow element around the exponent, regardless of what that expression is. In the second case the mrow is indeed required, because otherwise the msup element would have four children instead of the required two. In the first case, however, the mrow around the *x* would be absolutely redundant.

Consider another example:

```
<apply>
  <factorial/>
  <ci> x </ci>
</apply>
```

```
<apply>
  <factorial/>
  <apply>
    <plus/>
    <ci> x </ci>
    <ci> y </ci>
  </apply>
</apply>
```

The optimal presentation markup in this case is the following:

```
<mrow>
  <mi> x </mi>
  <mo> ! </mo>
</mrow>
```

```
<mrow>
  <mfenced separators="">
    <mi> x </mi>
    <mo> + </mo>
    <mi> y </mi>
  </mfenced>
  <mo> ! </mo>
</mrow>
```

There are several important things to note here. The first point is that just like in the previous example the x is not surrounded by the `mrow` element, but the $x + y$ expression now has parentheses around it instead of just `mrow`. That is, the presentation markup for the *same* content markup

```
<apply>
  <plus/>
  <ci> x </ci>
  <ci> y </ci>
</apply>
```

is now different. In other words, the expression $x + y$ reacts to the context in which it is used. The most important bit here is that the decision about whether to put an `mrow` or `mfenced` element, or nothing at all around the expression should be delayed until the expression itself is processed (that is matched by a template), as opposed to when its parent or predecessor is matched and processed (this is very similar to the issue of parenthesizing discussed in the previous section). Otherwise, redundancy will result. For instance, one way to process the factorial might be as follows. When it is matched, the template would check for its second child in order to decide whether it should insert parentheses around it. There are many possibilities here, and it would be probably impossible to enumerate all possible types of expressions that should not be parenthesized. Here are some examples: $5!$, $n!$, $n^2!$, $\lfloor x \rfloor!$, etc. However, even if all possibilities could be enumerated, such an approach would still not be optimal. The reason is that when the expression $x + y$ is processed there will be a similar dilemma: either blindly insert an `mrow` element around the expression, or otherwise check to see what its parent or predecessor is (again, there are too many possibilities here so it is not plausible), and make a decision based on that. If an `mrow` is inserted, it would be an immediate and only child of the `mfenced`. That is unnecessary because the `mfenced` element implies an `mrow` around its children.

One point needs clarification here: why is `mrow` required at all around an expression such as $x + y$? Normally, the `mrow` element is needed so that a mathematical expression is broken up in appropriate places. However, sometimes `mrow` may not be needed, as in this example showing the MathML encoding for $x + y - 1$:

```
<apply>
  <minus/>
  <apply>
    <plus/>
    <ci> x </ci>
    <ci> y </ci>
  </apply>
  <cn> 1 </cn>
</apply>
```

The optimal presentation markup in this case is

```
<mrow>
  <mi> x </mi>
  <mo> + </mo>
  <mi> y </mi>
  <mo> - </mo>
  <mn> 1 </mi>
</mrow>
```

Note the absence of an `mrow` around $x + y$.

To sum it up, the solution to the problem is to pass the information about the context in which the expression is used down the expression tree. This eliminates the need to check at each node for the type of its child or parent/predecessor, thus making the stylesheet itself much simpler and robust and at the same time makes it possible to produce output that is closer to being optimal.

4.4 Output Localization

It is a well-known fact that different countries often use different notation for the same mathematical notions. For example, division of a by b can be expressed as either a / b,

or $a \div b$, or $a\overline{)b}$, to name only a few possibilities. Another example is the binomial coefficient, which can be represented as follows: $\binom{m}{n}$, or ${}_nC_m$, or C_m^n.

Naturally, the user wants to control how the output is generated. One possible way to implement this is to introduce another parameter to the stylesheet. Depending on the value of that parameter localized output could be generated. Such an approach, however, has some disadvantages. There are several hundred localizations possible. Taking each of them into account would make the stylesheet extremely large and complex and thus less robust. Another possibility would be to introduce one parameter per each operator and depending on the value of that parameter generate particular presentation elements for that operator. This approach would be more flexible, but would still suffer from the same problem as the previous one.

The more elegant approach would be to generate the templates of the stylesheet itself according to local preferences. That would keep the stylesheet as simple and small as possible while providing the needed functionality. A stylesheet generated in this way would have the exact same set of templates as the original, but each of them would be slightly different from the templates in the original stylesheet given in the Appendix A. The only drawback of this approach is that there must be a special application that would generate a localized version of a stylesheet before it can be used. For example, a browser might need a special plug-in, or otherwise this functionality has to be built into the browser or other client application. This, however, is a topic for further research...

Chapter 5

Rendering TEX Mathematics to MathML Presentation

In this chapter we shall discuss how TEX documents can be transformed into MathML presentation markup. Why bother? One reason is that majority of scientific documents and papers in the last 20 years have been typeset using TEX. TEX is still widely used, and as the Internet has established itself as the primary way of the information exchange, it seems only natural to want to be able to publish TEX documents on the Web. That is where MathML can really help. However, at this point there aren't any applications that are capable of transforming a TEX document into MathML format. This part of the thesis is an attempt to create an application that would be able to convert mathematics encoded in TEX into MathML.

5.1 The TEX Macro Expander Application

The process of transforming a TEX document into a MathML document is by far harder than transforming MathML content markup into presentation markup. The reasons are many. For one thing, TEX has a very rich set of predefined macros. Secondly, transforming TEX macros into MathML presentation markup is not as natural as transforming MathML content markup into presentation markup. Tools like XSLT cannot be used anymore either because TEX is not a valid XML.

An additional complication is that TEX also allows the user to define some new macros. This poses a problem because the application trying to convert a TEX document into MathML markup would only be able to recognize those macros that are native to TEX. Therefore, before the actual transformation can be performed, the user-defined macros have to be expanded down to the native TEX code. This means that such TEX to MathML transformation has to be a two-stage process. Hence, two applications have to be developed.

The first application would be a TEX macro expander. It takes as an input the original TEX file, and produces the output file in TEX format as well, but with all the user-defined macros expanded. All other macros are kept because they can be transformed into MathML directly. The application performing macro expansion was written in Java for portability reasons. Portability is especially important when it comes to Web-related applications. Potentially (with some minor changes) this application might become a downloadable applet so that TEX documents could be transformed into MathML 'on the fly', and consequently be rendered by a browser or other client application.

The application expects two parameters. The first parameter is the name of the TEX file being transformed. The extension of the filename is expected to be .tex, and can be omitted. Very often the TEX file itself contains all the user macro definitions. However, the user might want to specify an external file containing such definitions. In such a case it can be specified as a second parameter.

The whole application consists of one class called Texmex (which stands for TEX Macro EXpander). There are two public methods in this class:

```
public Texmex()
public boolean texmex(String texfile, String stylefile)
```

The first method is the constructor. The second method accepts two parameters, both of type `String`: first parameter is name of a TEX file being expanded, and second parameter is the name of a file containing definitions for user's macros. If the second parameter is not used it has to be `null`. The output is stored in the file with the same name as the original TEX file, but the extension is ".out".

5.2 The Main Application

The main application is the one that does the actual transformation. It also consists of one class called Tex2mml, which has two methods:

```
public Tex2mml()
public Boolean tex2mml(Vector options,
                       String infile,
                       String mapfile)
```

The first parameter is a vector containing command line parameters. The second parameter is the name of the TEX file being transformed. The last parameter is the name of an additional file that defines how TEX macros should be mapped to MathML markup. Only the second parameter is required. If other parameters are not supplied they have to be `null`.

The only command line parameter defined so far is `t`. This parameter defines whether non-math mode TEX markup is included in the output or not. Note that names of options stored in the vector (as `Strings`) do not have the leading dash, although it has to precede each parameter given in the command line.

There is a special file named tex2mml.map that defines how TEX macros are mapped into MathML markup. This file is used by the main application by default. The

third parameter, if used, should be name of the file containing alternative definitions of the mappings. They will then supersede the default mappings stored in the tex2mml.map file.

The mappings in the file are stored in a particular format. Each of them starts with a double right-angle bracket. What follows is the name of a macro (starting with a backslash), or any other sequence of one or more characters. If the macro does not expect any parameters then the new line character has to follow the name of the macro or a token, and the following line (or lines) contain the MathML markup that would replace the macro. Here are some examples of mapping definitions:

```
>>\ne
<mo> <mchar name="NotEqual"/> </mo>

>>\not=
<mo> <mchar name="NotEqual"/> </mo>

>>\neq
<mo> <mchar name="NotEqual"/> </mo>
```

Therefore, all three TEX macros would be mapped to the same MathML markup. Note that the MathML markup replacing the TEX macro starts on the next line below the right double angle bracket, and go on until the next right double angle bracket is found. The empty lines at the end of the MathML markup are clipped.

A more interesting situation is when a macro has some parameters. In such a case they have to be specified in the list of arguments to the macro. The name of a parameter can be a single digit or letter (case sensitive) preceded by the @ character. In the MathML markup they have to be preceded by the # character, e.g.:

```
>>\zzz @1@2
<couple>#2<sep/>#1</couple>
```

In this example the `\zzz` macro takes two parameters immediately following each other, and they are inserted in the appropriate places in the MathML markup. So the following TEX markup

```
\zzz 13 2
```

would be transformed into

```
<couple>2<sep/>13</couple>
```

Here is another example:

```
\yyy {a@Xr@Yeh?} ;
<couple>#Y<sep/>#X</couple>
```

Now, if TEX contains the following markup:

```
\yyy {a13 r 7 eh?} ;
```

then it would be transformed into

```
<couple>7<sep/>13</couple>
```

It is an error to make references to parameters that do not appear in the list of parameters to the TEX macro, like in the following example, where parameter `u` is referenced:

```
\xxx @a,@b
<mo>#b</mo>
<mo>#u</mo>
```

Note that it is ok not to reference one or more of the macro's parameters (in the example above parameter a is not used).

One big advantage of the mapping mechanism described above is its flexibility. Should some new macros be introduced in TEX (including user-defined ones), or some new elements (tags or attributes) be added to the next MathML specification, the mappings can be easily modified, or overridden by the user.

Unfortunately, some macros have to be hard coded. For example, the matrix macro has variable number of columns and rows, which poses a problem to the mapping mechanism described earlier in this section. Also note that this application is only concerned with the TEX markup given in 'math mode', that is what is put in between either `$` or `$$`. The rest of the TEX markup is treated as plain text, and not transformed in any way.

Finally, there are a few more points worth mentioning. One of them concerns operand grouping. Let us consider the following TEX markup:

```
$1+c/x$
```

When MathML markup is generated it is important that `mrow` elements are inserted in the appropriate places. Here is the corresponding MathML markup:

```
<mrow>
  <mn>1</mn>
  <mrow>
    <mi>c</mi>
    <mo>/</mo>
    <mi>x</mi>
  </mrow>
</mrow>
```

Note the use of the innermost `mrow`. Without this `mrow` the renderer might split the expression anywhere between `c` and `x`. In order to correctly generate the `mrow` elements in the output each operator has to be assigned a certain level of precedence, similar to how it was done in the stylesheet discussed in Chapter 4. The difference is that operator precedences were used there to insert parentheses in appropriate places, while here this information is only used to generate the `mrow` elements for correct grouping of operands.

Finally, there are situations when the explicit line breaks present in the TEX markup have to be ignored. One situation is when they occur inside the `matrix` macro:

```
\matrix {a&b\cr c&d\cr}
```

Because it is part of the syntax, it is very easy to deal with explicit line breaks as in the example above. However, in other situations they might be used simply to break up a mathematical expression in the appropriate places. Cases like this have to be recognized and the explicit line breaks have to be ignored. Explicit line breaks are not needed in MathML markup is because once the `mrow` elements are inserted in the appropriate places, it is the renderer's job to break up the expression where needed.

In conclusion, please note again that this application does not transform any TEX markup that is not in math mode (i.e. it is treated as plain text). Maybe a topic for another thesis...

Summary and Conclusion

Developed under the auspices of the World Wide Web Consortium, MathML is the first standard that makes it possible to exchange mathematical objects among applications in a semantically meaningful way. In this thesis we introduced two software applications supporting the standard to prove that MathML is not only a viable but also a very elegant solution to the problem of authoring, exchange and publishing of mathematically rich content.

The first of the two applications is a XSLT stylesheet that transforms MathML content markup into MathML presentations markup. It serves as evidence that the abstraction mechanism on which MathML is based and which makes MathML so flexible and versatile, indeed works.

The purpose of the second application was to show that it is possible to convert documents from legacy formats (particularly TEX) into MathML markup, and thus one of the goals established by the developers of MathML was successfully achieved.

Bibliography

[1] W3C Mathematical Markup Language (MathML) Version 1.01 Specification, W3C Recommendation, from July 7, 1999.
http://www.w3.org/1999/07/REC-MathML-19990707

[2] W3C Mathematical Markup Language (MathML) Version 2.0 Candidate Recommendation, from November 13, 2000.
http://www.w3.org/TR/2000/CR-MathML2-20001113

[3] W3C XSL Transformations (XSLT) Version 1.0 Recommendation, from November 16, 1999.
http://www.w3.org/TR/1999/REC-xslt-19991116

[4] W3C XSL Transformations (XSLT) Version 1.1 Working Draft, from December 12, 2000.
http://www.w3.org/TR/2000/WD-xslt11-20001212

[5] W3C XML Path Language (XPath) Version 1.0 Recommendation, from November 16, 1999.
http://www.w3.org/TR/1999/REC-xpath-19991116

[6] Donald E. Knuth. *The TEXbook.* 13-th edition, Addison-Wesley, 1987.

Appendix A

The MathML Content to MathML Presentation Stylesheet

```
<?xml version="1.0"?>

<!-- ************************************************** -->
<!-- XSL Transform of MathML content to MathML presentation -->
<!--    By Igor Rodionov, Computer Science Department of    -->
<!--   the University of Western Ontario, London, Canada    -->
<!-- Partially complies with the W3C spec from Nov.13, 2000 -->
<!--              Version 0.14 from Dec. 13, 2000           -->
<!--               Comments to:  igor@csd.uwo.ca            -->
<!-- ************************************************** -->

<xsl:stylesheet id="http://www.scl.csd.uwo.ca/mml2mml.xsl"
                version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:mml="http://www.w3.org/1998/Math/MathML"
                xmlns="http://www.w3.org/1998/Math/MathML">

<xsl:output method="xml" indent="yes"/>

<xsl:strip-space elements="apply semantics annotation-xml
        csymbol fn cn ci interval matrix matrixrow vector
        lambda bvar condition logbase degree set list
        lowlimit uplimit"/>


<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!--         Parameters, variables and constants           -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->

<!-- ~~~~~~~~ Semantics related *constants*: ~~~~~~~~ -->

<!-- Strip off semantics -->
<xsl:variable name="SEM_STRIP" select="-1"/>

<!-- Pass semantics "as is" -->
<xsl:variable name="SEM_PASS" select="0"/>

<!-- Add semantics at top level only -->
```

```
<xsl:variable name="SEM_TOP" select="1"/>

<!-- Add semantics at all levels -->
<xsl:variable name="SEM_ALL" select="2"/>

<!-- Semantics at top level only, with id refs -->
<!-- NOTE: ids have to be already present in the
           input for this feature to work. -->
<xsl:variable name="SEM_XREF" select="3"/>

<!-- No semantics at top level, with id refs -->
<!-- NOTE: ids have to be already present in the
           input for this feature to work. -->
<xsl:variable name="SEM_XREF_EXT" select="4"/>


<!-- ~~~~~~~~~~ Stylesheet *parameter*: SEM_SW ~~~~~~~~~~~~~~ -->
<!-- Assumes one of the above values; SEM_PASS is the default -->
<!-- The default can be overridden by specifying different    -->
<!-- value on the command line when the stylesheet is invoked -->

<xsl:param name="SEM_SW" select="SEM_PASS"/>


<!-- ~~~~~~ Operator precedence definitions ~~~~~~ -->

<xsl:variable name="NO_PREC" select="0"/>
<xsl:variable name="UNION_PREC" select="1"/>
<xsl:variable name="SETDIFF_PREC" select="1"/>
<xsl:variable name="INTERSECT_PREC" select="3"/>
<xsl:variable name="OR_PREC" select="5"/>
<xsl:variable name="XOR_PREC" select="5"/>
<xsl:variable name="AND_PREC" select="7"/>
<xsl:variable name="PLUS_PREC" select="9"/>
<xsl:variable name="MINUS_PREC" select="9"/>
<xsl:variable name="MUL_PREC" select="11"/>
<xsl:variable name="DIV_PREC" select="11"/>
<xsl:variable name="NEG_PREC" select="13"/>
<xsl:variable name="FUNCTION_PREC" select="99"/>

<!-- ~~~~~ Miscellaneous constant definitions ~~~~~ -->

<xsl:variable name="YES" select="1"/>
<xsl:variable name="NO" select="0"/>
<xsl:variable name="NO_PARAM" select="-1"/>
<xsl:variable name="PAR_SAME" select="-3"/>
<xsl:variable name="PAR_YES" select="-5"/>
<xsl:variable name="PAR_NO" select="-7"/>
```

```
<!-- +++++++++++++++++ INDEX OF TEMPLATES +++++++++++++++++++ -->

<!-- All templates are subdivided into the following categories
     (listed in the order of appearance in the stylesheet):

THE TOPMOST ELEMENT: MATH
 math

SEMANTICS HANDLING
 semantics

BASIC CONTAINER ELEMENTS
 cn, ci, csymbol

BASIC CONTENT ELEMENTS
 fn, interval, inverse, sep, condition, declare, lambda, compose, ident

ARITHMETIC, ALGEBRA & LOGIC
 quotient, exp, factorial, max, min, minus, plus, power, rem, divide,
 times, root, gcd, and, or, xor, not, forall, exists, abs, conjugate,
 arg, real, imaginary

RELATIONS
 neq, approx, tendsto, implies, in, notin, notsubset, notprsubset,
 subset, prsubset, eq, gt, lt, geq, leq, equivalent

CALCULUS
 ln, log, diff, partialdiff, lowlimit, uplimit, bvar, degree,
 logbase, divergence, grad, curl, laplacian

SET THEORY
 set, list, union, intersect, setdiff, card

SEQUENCES AND SERIES
 sum, product, limit

TRIGONOMETRY
 sin, cos, tan, sec, csc, cot, sinh, cosh, tanh, sech, csch, coth,
 arcsin, arccos, arctan

STATISTICS
 mean, sdev, variance, median, mode, moment

LINEAR ALGEBRA
 vector, matrix, matrixrow, determinant, transpose, selector,
 vectorproduct, scalarproduct, outerproduct
-->


<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
```

```
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~ TEMPLATES ~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->

<!-- ***************** THE TOPMOST ELEMENT: MATH ***************** -->

<xsl:template match = "mml:math">
  <xsl:choose>
    <xsl:when test="$SEM_SW=$SEM_TOP or $SEM_SW=$SEM_ALL and *[2] or
                                                    $SEM_SW=$SEM_XREF">
      <semantics>
        <mrow>
          <xsl:apply-templates mode = "semantics"/>
        </mrow>
        <annotation-xml encoding="MathML">
          <xsl:copy-of select="*"/>
        </annotation-xml>
      </semantics>
    </xsl:when>
    <xsl:otherwise>
      <mrow>
        <xsl:apply-templates mode = "semantics"/>
      </mrow>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>


<!-- ***************** SEMANTICS HANDLING ***************** -->

<!-- This template is called recursively.  At each level   -->
<!-- in the source tree it decides whether to strip off,   -->
<!-- pass or add semantics at that level (depending on the -->
<!-- value of SEM_SW parameter).  Then the actual template -->
<!-- is applied to the node.                               -->

<xsl:template match = "mml:*" mode = "semantics">
  <xsl:param name="IN_PREC" select="$NO_PREC"/>
  <xsl:param name="PARAM" select="$NO_PARAM"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:choose>
    <xsl:when test="$SEM_SW=$SEM_STRIP and self::mml:semantics">
      <xsl:apply-templates
                 select="mml:annotation-xml[@encoding='MathML']">
        <xsl:with-param name="IN_PREC" select="$IN_PREC"/>
        <xsl:with-param name="PARAM" select="$PARAM"/>
        <xsl:with-param name="PAREN" select="$PAREN"/>
        <xsl:with-param name="PAR_NO_IGNORE" select="$PAR_NO_IGNORE"/>
      </xsl:apply-templates>
```

```
</xsl:when>
<xsl:when test="($SEM_SW=$SEM_PASS or $SEM_SW=$SEM_TOP) and
    '           self::mml:semantics">
  <semantics>
    <xsl:apply-templates select="*[1]">
      <xsl:with-param name="IN_PREC" select="$IN_PREC"/>
      <xsl:with-param name="PARAM" select="$PARAM"/>
      <xsl:with-param name="PAREN" select="$PAREN"/>
      <xsl:with-param name="PAR_NO_IGNORE"
                      select="$PAR_NO_IGNORE"/>
    </xsl:apply-templates>
    <xsl:copy-of select="mml:annotation-xml"/>
  </semantics>
</xsl:when>
<xsl:when test="$SEM_SW=$SEM_ALL">
  <semantics>
    <xsl:choose>
      <xsl:when test="self::mml:semantics">
        <xsl:apply-templates select="*[1]">
          <xsl:with-param name="IN_PREC" select="$IN_PREC"/>
          <xsl:with-param name="PARAM" select="$PARAM"/>
          <xsl:with-param name="PAREN" select="$PAREN"/>
          <xsl:with-param name="PAR_NO_IGNORE"
                          select="$PAR_NO_IGNORE"/>
        </xsl:apply-templates>
        <xsl:copy-of select="mml:annotation-xml"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:apply-templates select=".">
          <xsl:with-param name="IN_PREC" select="$IN_PREC"/>
          <xsl:with-param name="PARAM" select="$PARAM"/>
          <xsl:with-param name="PAREN" select="$PAREN"/>
          <xsl:with-param name="PAR_NO_IGNORE"
                          select="$PAR_NO_IGNORE"/>
        </xsl:apply-templates>
        <annotation-xml encoding="MathML">
          <xsl:copy-of select="."/>
        </annotation-xml>
      </xsl:otherwise>
    </xsl:choose>
  </semantics>
</xsl:when>
<xsl:when test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                              @id">
  <xsl:choose>
    <xsl:when test="self::mml:sematics">
      <xsl:copy>
        <xsl:copy-of select="@*"/>
        <xsl:attribute name="xref">
          <xsl:value-of select="@id"/>
```

```
            </xsl:attribute>
            <xsl:copy-of select="*[1]"/>
            <xsl:copy-of select="mml:annotation-xml"/>
          </xsl:copy>
        </xsl:when>
        <xsl:otherwise>
          <xsl:apply-templates select=".">
            <xsl:with-param name="IN_PREC" select="$IN_PREC"/>
            <xsl:with-param name="PARAM" select="$PARAM"/>
            <xsl:with-param name="PAREN" select="$PAREN"/>
            <xsl:with-param name="PAR_NO_IGNORE"
                            select="$PAR_NO_IGNORE"/>
          </xsl:apply-templates>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:when>
    <xsl:otherwise>
      <xsl:choose>
        <xsl:when test="self::mml:semantics">
          <xsl:copy-of select="."/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:apply-templates select=".">
            <xsl:with-param name="IN_PREC" select="$IN_PREC"/>
            <xsl:with-param name="PARAM" select="$PARAM"/>
            <xsl:with-param name="PAREN" select="$PAREN"/>
            <xsl:with-param name="PAR_NO_IGNORE"
                            select="$PAR_NO_IGNORE"/>
          </xsl:apply-templates>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match = "mml:semantics">
  <xsl:apply-templates select="." mode = "semantics"/>
</xsl:template>


<!-- ***************** BASIC CONTAINER ELEMENTS ***************** -->

<xsl:template match = "mml:cn">
  <xsl:param name="IN_PREC" select="$NO_PREC"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:choose>
    <xsl:when test=". &lt; 0 and $IN_PREC &gt; $NO_PREC and
                     $PAREN=$PAR_NO and $PAR_NO_IGNORE=$NO">
      <mfenced separators="">
```

```
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                    @id">
        ' <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="." mode="cn"/>
      </mfenced>
    </xsl:when>
    <xsl:otherwise>
      <xsl:choose>
        <xsl:when test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT)
                                                          and @id">
          <xsl:apply-templates select="." mode="cnid"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:apply-templates select="." mode="cn"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match = "mml:cn" mode="cn">
  <xsl:choose>
    <xsl:when test="@base and (not(@type) or @type='integer' or
                                              @type='real')">
      <msub>
        <mn> <xsl:apply-templates mode = "semantics"/> </mn>
        <mn> <xsl:value-of select="@base"/> </mn>
      </msub>
    </xsl:when>
    <xsl:when test="@type='complex' and not(@base) and
                                    child::mml:sep[1]">
      <mfenced separators="">
        <mn>
          <xsl:apply-templates select="text()[1]" mode = "semantics"/>
        </mn>
        <xsl:if test="text()[2] &lt; 0">
          <mo>-</mo>
          <mn> <xsl:value-of select="-text()[2]"/> </mn>
        </xsl:if>
        <xsl:if test="not(text()[2] &lt; 0)">
          <mo>+</mo>
          <mn> <xsl:value-of select="text()[2]"/> </mn>
        </xsl:if>
        <mo> <mchar name="InvisibleTimes"/> </mo>
        <mo>i</mo>
      </mfenced>
    </xsl:when>
```

```
<xsl:when test="@type='complex' and @base and child::mml:sep[1]">
  <msub>
    <mfenced separators="">
      <mn> <xsl:apply-templates select="text()[1]"/> </mn>
      <xsl:if test="text()[2] &lt; 0">
        <mo>-</mo>
        <mn> <xsl:value-of select="-text()[2]"/> </mn>
      </xsl:if>
      <xsl:if test="not(text()[2] &lt; 0)">
        <mo>+</mo>
        <mn> <xsl:apply-templates select="text()[2]"/> </mn>
      </xsl:if>
      <mo> <mchar name="InvisibleTimes"/> </mo>
      <mo>i</mo>
    </mfenced>
    <mn> <xsl:value-of select="@base"/> </mn>
  </msub>
</xsl:when>
<xsl:when test="@type='rational' and not(@base) and
                                child::mml:sep[1]">
  <mfrac>
    <mn> <xsl:apply-templates select="text()[1]"/> </mn>
    <mn> <xsl:apply-templates select="text()[2]"/> </mn>
  </mfrac>
</xsl:when>
<xsl:when test="@type='rational' and @base and child::mml:sep[1]">
  <msub>
    <mfenced>
      <mfrac>
        <mn> <xsl:apply-templates select="text()[1]"/> </mn>
        <mn> <xsl:apply-templates select="text()[2]"/> </mn>
      </mfrac>
    </mfenced>
    <mn> <xsl:value-of select="@base"/> </mn>
  </msub>
</xsl:when>
<xsl:when test="@type='polar' and not(@base) and
                              child::mml:sep[1]">
  <mrow>
    <mo>Polar</mo>
    <mo> <mchar name="InvisibleTimes"/> </mo>
    <mfenced separators=",">
      <mn> <xsl:apply-templates select="text()[1]"/> </mn>
      <mn> <xsl:apply-templates select="text()[2]"/> </mn>
    </mfenced>
  </mrow>
</xsl:when>
<xsl:when test="@type='polar' and @base and child::mml:sep[1]">
  <msub>
    <mrow>
```

```
            <mo>Polar</mo>
            <mo> <mchar name="InvisibleTimes"/> </mo>
            <mfenced separators=",">
              <mn> <xsl:apply-templates select="text()[1]"/> </mn>
              <mn> <xsl:apply-templates select="text()[2]"/> </mn>
            </mfenced>
          </mrow>
          <mn> <xsl:value-of select="@base"/> </mn>
        </msub>
      </xsl:when>
      <xsl:otherwise>
        <mn> <xsl:apply-templates mode = "semantics"/> </mn>
      </xsl:otherwise>
    </xsl:choose>
</xsl:template>

<xsl:template match = "mml:cn" mode="cnid">
  <xsl:choose>
    <xsl:when test="@base and (not(@type) or @type='integer' or
                                                    @type='real')">
      <msub xref="{@id}">
        <mn> <xsl:apply-templates mode = "semantics"/> </mn>
        <mn> <xsl:value-of select="@base"/> </mn>
      </msub>
    </xsl:when>
    <xsl:when test="@type='complex' and not(@base) and
                                    child::mml:sep[1]">
      <mfenced separators="" xref="{@id}">
        <mn>
          <xsl:apply-templates select="text()[1]" mode = "semantics"/>
        </mn>
        <xsl:if test="text()[2] &lt; 0">
          <mo>-</mo>
          <mn> <xsl:value-of select="-text()[2]"/> </mn>
        </xsl:if>
        <xsl:if test="not(text()[2] &lt; 0)">
          <mo>+</mo>
          <mn> <xsl:value-of select="text()[2]"/> </mn>
        </xsl:if>
        <mo> <mchar name="InvisibleTimes"/> </mo>
        <mo>i</mo>
      </mfenced>
    </xsl:when>
    <xsl:when test="@type='complex' and @base and child::mml:sep[1]">
      <msub xref="{@id}">
        <mfenced separators="">
          <mn> <xsl:apply-templates select="text()[1]"/> </mn>
          <xsl:if test="text()[2] &lt; 0">
            <mo>-</mo>
            <mn> <xsl:value-of select="-text()[2]"/> </mn>
```

```
      </xsl:if>
      <xsl:if test="not(text()[2] &lt; 0)">
        <mo>+</mo>
        <mn> <xsl:apply-templates select="text()[2]"/> </mn>
      </xsl:if>
      <mo> <mchar name="InvisibleTimes"/> </mo>
      <mo>i</mo>
    </mfenced>
    <mn> <xsl:value-of select="@base"/> </mn>
  </msub>
</xsl:when>
<xsl:when test="@type='rational' and not(@base) and
                                  child::mml:sep[1]">
  <mfrac xref="{@id}">
    <mn> <xsl:apply-templates select="text()[1]"/> </mn>
    <mn> <xsl:apply-templates select="text()[2]"/> </mn>
  </mfrac>
</xsl:when>
<xsl:when test="@type='rational' and @base and child::mml:sep[1]">
  <msub xref="{@id}">
    <mfenced>
      <mfrac>
        <mn> <xsl:apply-templates select="text()[1]"/> </mn>
        <mn> <xsl:apply-templates select="text()[2]"/> </mn>
      </mfrac>
    </mfenced>
    <mn> <xsl:value-of select="@base"/> </mn>
  </msub>
</xsl:when>
<xsl:when test="@type='polar' and not(@base) and
                                child::mml:sep[1]">
  <mrow xref="{@id}">
    <mo>Polar</mo>
    <mo> <mchar name="InvisibleTimes"/> </mo>
    <mfenced separators=",">
      <mn> <xsl:apply-templates select="text()[1]"/> </mn>
      <mn> <xsl:apply-templates select="text()[2]"/> </mn>
    </mfenced>
  </mrow>
</xsl:when>
<xsl:when test="@type='polar' and @base and child::mml:sep[1]">
  <msub xref="{@id}">
    <mrow>
      <mo>Polar</mo>
      <mo> <mchar name="InvisibleTimes"/> </mo>
      <mfenced separators=",">
        <mn> <xsl:apply-templates select="text()[1]"/> </mn>
        <mn> <xsl:apply-templates select="text()[2]"/> </mn>
      </mfenced>
    </mrow>
```

```
        <mn> <xsl:value-of select="@base"/> </mn>
      </msub>
    </xsl:when>
    <xsl:otherwise>
      <mn xref="{@id}"> <xsl:apply-templates mode = "semantics"/> </mn>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match = "mml:ci">
  <xsl:choose>
    <xsl:when test="@type='vector' or @type=matrix or @type=set">
      <mi fontweight="bold">
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                    @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates mode = "semantics"/>
      </mi>
    </xsl:when>
    <xsl:when test="child::text() and not(child::*[1])">
      <mi>
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                    @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates/>
      </mi>
    </xsl:when>
    <xsl:when test="child::text() and *[1] and not(*[1]=mml:sep)">
      <mrow>
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                    @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates/>
      </mrow>
    </xsl:when>
    <xsl:otherwise>
      <xsl:if test="*[2]">
        <mrow>
          <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT)
                                                        and @id">
            <xsl:attribute name="xref">
```

```
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="*"/>
      </mrow>
    </xsl:if>
    <xsl:if test="not(*[2])">
      <xsl:apply-templates select="*[1]"/>
    </xsl:if>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>

<xsl:template match = "mml:ci/mml:*[not(self::mml:sep)]">
 <xsl:copy-of select = "."/>
</xsl:template>

<xsl:template match = "mml:csymbol/mml:*">
 <xsl:choose>
  <xsl:when test="self::mml:cn or self::mml:ci">
   <xsl:apply-templates mode = "semantics"/>
  </xsl:when>
  <xsl:otherwise>
   <xsl:copy-of select = "."/>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>

<xsl:template match = "mml:csymbol/text()">
 <xsl:choose>
  <xsl:when test=". &lt; 0 or . = 0 or . &gt; 0">
   <mn> <xsl:copy-of select = "."/> </mn>
  </xsl:when>
  <xsl:otherwise>
   <mi> <xsl:copy-of select = "."/> </mi>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>


<!-- ***************** BASIC CONTENT ELEMENTS ***************** -->

<xsl:template match = "mml:apply[mml:*[1][self::mml:fn]]">
 <mrow>
  <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                          @id">
   <xsl:attribute name="xref">
    <xsl:value-of select="@id"/>
   </xsl:attribute>
  </xsl:if>
```

```
    <xsl:apply-templates select = "mml:fn[1]" mode = "semantics"/>
    <mo> <mchar name='ApplyFunction'/> </mo>
    <mfenced separators=",">
      <xsl:apply-templates select = "*[position()>1]"
                           mode = "semantics"/>
    </mfenced>
  </mrow>
</xsl:template>

<xsl:template match = "mml:fn">
  <xsl:apply-templates select = "*[1]" mode = "semantics">
    <xsl:with-param name="IN_PREC" select="$FUNCTION_PREC"/>
  </xsl:apply-templates>
</xsl:template>

<xsl:template match = "mml:interval">
  <mfenced separators=",">
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                          @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="@closure='closed' or @closure='closed-open'">
      <xsl:attribute name="open">[</xsl:attribute>
    </xsl:if>
    <xsl:if test="@closure='closed' or @closure='open-closed'">
      <xsl:attribute name="close">]</xsl:attribute>
    </xsl:if>
    <xsl:apply-templates select="*" mode = "semantics"/>
  </mfenced>
</xsl:template>

<xsl:template match =
              "mml:apply[mml:*[1][self::mml:apply[mml:inverse[1]]]]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                          @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:apply-templates select = "*[1]" mode = "semantics"/>
    <mo> <mchar name='ApplyFunction'/> </mo>
    <mfenced separators=",">
      <xsl:apply-templates select="*[position()>1]" mode="semantics"/>
    </mfenced>
 </mrow>
</xsl:template>
```

```
<xsl:template match = "mml:apply[*[1][self::mml:inverse]]">
  <xsl:choose>
    <xsl:when test="*[2]=mml:exp or *[2]=mml:ln or *[2]=mml:sin or
                    *[2]=mml:cos or *[2]=mml:tan or *[2]=mml:sec or
                    *[2]=mml:csc or *[2]=mml:cot or *[2]=mml:sinh or
                    *[2]=mml:cosh or *[2]=mml:tanh or *[2]=mml:csch or
                    *[2]=mml:coth or *[2]=mml:arcsin or *[2]=mml:arccos or
                    *[2]=mml:arctan or *[2]=mml:sech">
      <mo>
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                    @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="*[2]" mode="inverse"/>
      </mo>
    </xsl:when>
    <xsl:otherwise>
      <msup>
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                    @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select = "*[2]"/>
        <mfenced>
          <mn>-1</mn>
        </mfenced>
      </msup>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match = "*" mode="inverse">
  <xsl:choose>
    <xsl:when test="self::mml:exp">
      <xsl:value-of select="'ln'"/>
    </xsl:when>
    <xsl:when test="self::mml:ln">
      <xsl:value-of select="'exp'"/>
    </xsl:when>
    <xsl:when test="self::mml:sin">
      <xsl:value-of select="'arcsin'"/>
    </xsl:when>
    <xsl:when test="self::mml:cos">
      <xsl:value-of select="'arccos'"/>
    </xsl:when>
    <xsl:when test="self::mml:tan">
```

```
      <xsl:value-of select="'arctan'"/>
    </xsl:when>
    <xsl:when test="self::mml:sec">
      <xsl:value-of select="'arcsec'"/>
    </xsl:when>
    <xsl:when test="self::mml:csc">
      <xsl:value-of select="'arccsc'"/>
    </xsl:when>
    <xsl:when test="self::mml:cot">
      <xsl:value-of select="'arccot'"/>
    </xsl:when>
    <xsl:when test="self::mml:sinh">
      <xsl:value-of select="'arcsinh'"/>
    </xsl:when>
    <xsl:when test="self::mml:cosh">
      <xsl:value-of select="'arccosh'"/>
    </xsl:when>
    <xsl:when test="self::mml:tanh">
      <xsl:value-of select="'arctanh'"/>
    </xsl:when>
    <xsl:when test="self::mml:sech">
      <xsl:value-of select="'arcsech'"/>
    </xsl:when>
    <xsl:when test="self::mml:csch">
      <xsl:value-of select="'arccsch'"/>
    </xsl:when>
    <xsl:when test="self::mml:coth">
      <xsl:value-of select="'arccoth'"/>
    </xsl:when>
    <xsl:when test="self::mml:arcsin">
      <xsl:value-of select="'sin'"/>
    </xsl:when>
    <xsl:when test="self::mml:arccos">
      <xsl:value-of select="'cos'"/>
    </xsl:when>
    <xsl:when test="self::mml:arctan">
      <xsl:value-of select="'tan'"/>
    </xsl:when>
  </xsl:choose>
</xsl:template>

<xsl:template match = "mml:sep"/>

<xsl:template match = "mml:condition">
  <xsl:choose>
    <xsl:when test="parent::mml:apply[mml:forall[1]]"/>
    <xsl:otherwise>
      <xsl:choose>
        <xsl:when test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT)
                                                    and @id">
```

```
          <mrow xref="{@id}">
            <xsl:apply-templates select="*" mode = "semantics"/>
          </mrow>
        </xsl:when>
        <xsl:otherwise>
          <xsl:if test="not(*[2])">
            <xsl:apply-templates select="*" mode = "semantics"/>
          </xsl:if>
          <xsl:if test="*[2]">
            <mrow>
              <xsl:apply-templates select="*" mode = "semantics"/>
            </mrow>
          </xsl:if>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match = "mml:declare"/>

<xsl:template match = "mml:lambda">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <mo> <mchar name='Lambda'/> </mo>
    <mo> <mchar name='ApplyFunction'/> </mo>
    <mfenced separators=",">
      <xsl:for-each select = "*">
        <xsl:choose>
          <xsl:when test="self::mml:ci or self::mml:cn">
            <xsl:apply-templates select = "." mode="semantics"/>
          </xsl:when>
          <xsl:otherwise>
            <mrow>
              <xsl:apply-templates select = "." mode="semantics"/>
            </mrow>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:for-each>
    </mfenced>
  </mrow>
</xsl:template>

<xsl:template match =
              "mml:apply[*[1][self::mml:apply[mml:compose[1]]]]">
```

```
    <mrow>
      <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                    @id">
        <xsl:attribute name="xref">
          <xsl:value-of select="@id"/>
        </xsl:attribute>
      </xsl:if>
      <xsl:apply-templates select = "*[1]" mode = "semantics"/>
      <mo> <mchar name='ApplyFunction'/> </mo>
      <mfenced separators=",">
        <xsl:apply-templates select="*[position()>1]" mode="semantics"/>
      </mfenced>
  </mrow>
</xsl:template>

<xsl:template match = "mml:apply[*[1][self::mml:compose]]">
  <mfenced separators="">
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                  @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:apply-templates select = "mml:*[2][self::mml:ci[@type='fn'] |
                                    self::mml:fn]" mode="semantics"/>
    <xsl:for-each select="mml:*[position()>2][self::mml:ci[@type='fn']
                                                    | self::mml:fn]">
      <mo> <mchar name="SmallCircle"/> </mo>
      <xsl:apply-templates select = "." mode="semantics"/>
    </xsl:for-each>
  </mfenced>
</xsl:template>

<xsl:template match = "mml:ident">
  <xsl:choose>
    <xsl:when test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                  @id">
      <mo xref="{@id}">id</mo>
    </xsl:when>
    <xsl:otherwise>
      <mo>id</mo>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>


<!-- *************** ARITHMETIC, ALGEBRA & LOGIC *************** -->

<xsl:template match = "mml:apply[mml:quotient[1]]">
  <mrow>
```

```
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                              @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <mo form="prefix" fence="true" stretchy="true" lspace="0em"
                                                   rspace="0em">
      <mchar name="LeftFloor"/>
    </mo>
    <mfrac>
      <mrow>
        <xsl:apply-templates select="*[2]" mode = "semantics">
          <xsl:with-param name="IN_PREC" select="$FUNCTION_PREC"/>
        </xsl:apply-templates>
      </mrow>
      <mrow>
        <xsl:apply-templates select="*[3]" mode = "semantics">
          <xsl:with-param name="IN_PREC" select="$FUNCTION_PREC"/>
          <xsl:with-param name="PAR_NO_IGNORE" select="$NO"/>
        </xsl:apply-templates>
      </mrow>
    </mfrac>
    <mo form="postfix" fence="true" stretchy="true" lspace="0em"
                                                   rspace="0em">
      <mchar name="LeftFloor"/>
    </mo>
  </mrow>
</xsl:template>

<xsl:template match = "mml:apply[*[1][self::mml:exp]]">
  <msup>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                              @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <mo>e</mo>
    <xsl:apply-templates select = "*[2]" mode = "semantics"/>
  </msup>
</xsl:template>

<xsl:template match = "mml:apply[mml:factorial[1]]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                              @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
```

```
    </xsl:if>
    <xsl:apply-templates select = "*[2]" mode = "semantics">
      <xsl:with-param name="IN_PREC" select="$FUNCTION_PREC"/>
      <xsl:with-param name="PAR_NO_IGNORE" select="$NO"/>
    </xsl:apply-templates>
    <mo>!</mo>
  </mrow>
</xsl:template>

<xsl:template match = "mml:apply[mml:max[1] | mml:min[1]]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                              @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="*[2]=mml:bvar">
      <munder>
        <xsl:if test="*[1]=mml:max">
          <mo>max</mo>
        </xsl:if>
        <xsl:if test="*[1]=mml:min">
          <mo>min</mo>
        </xsl:if>
        <xsl:apply-templates select="*[2]" mode = "semantics"/>
      </munder>
      <xsl:if test="*[3]=mml:condition">
        <mfenced open="{{" close="}}" separators="">
          <mfenced open="" close="" separators=",">
            <xsl:for-each select = "*[position()>3]">
              <xsl:apply-templates select = "." mode="semantics"/>
            </xsl:for-each>
          </mfenced>
          <mo>|</mo>
          <xsl:apply-templates select="*[3]" mode = "semantics"/>
        </mfenced>
      </xsl:if>
      <xsl:if test="not(*[3]=mml:condition)">
        <mfenced open="{{" close="}}" separators=",">
          <xsl:for-each select = "*[position()>2]">
            <xsl:apply-templates select = "." mode="semantics"/>
          </xsl:for-each>
        </mfenced>
      </xsl:if>
    </xsl:if>
    <xsl:if test="*[2]=mml:condition">
      <xsl:if test="*[1]=mml:max">
        <mo>max</mo>
      </xsl:if>
```

```
      <xsl:if test="*[1]=mml:min">
        <mo>min</mo>
      </xsl:if>
      <mfenced open="{{" close="}}" separators="">
        <xsl:if test="*[3]">
          <mfenced open="" close="" separators=",">
            <xsl:for-each select = "*[position()>2]">
              <xsl:apply-templates select = "." mode="semantics"/>
            </xsl:for-each>
          </mfenced>
          <mo>|</mo>
        </xsl:if>
        <xsl:apply-templates select="*[2]" mode = "semantics"/>
      </mfenced>
    </xsl:if>
    <xsl:if test="not(*[2]=mml:condition) and not(*[2]=mml:bvar)">
      <xsl:if test="*[1]=mml:max">
        <mo>max</mo>
      </xsl:if>
      <xsl:if test="*[1]=mml:min">
        <mo>min</mo>
      </xsl:if>
      <mfenced open="{{" close="}}" separators=",">
        <xsl:for-each select = "*[position()>1]">
          <xsl:apply-templates select = "." mode="semantics"/>
        </xsl:for-each>
      </mfenced>
    </xsl:if>
  </mrow>
</xsl:template>

<xsl:template match = "mml:apply[mml:minus[1]]">
  <xsl:param name="IN_PREC" select="$NO_PREC"/>
  <xsl:param name="PARAM" select="$NO_PARAM"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:choose>
    <xsl:when test="$IN_PREC &gt; $MINUS_PREC or $IN_PREC=$MINUS_PREC
                                                and $PARAM=$PAR_SAME">
      <mfenced separators="">
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="." mode="minus">
          <xsl:with-param name="PARAM" select="$PARAM"/>
          <xsl:with-param name="PAREN" select="$PAR_YES"/>
          <xsl:with-param name="PAR_NO_IGNORE"
```

```
                        select="$PAR_NO_IGNORE"/>
        </xsl:apply-templates>
      </mfenced>
    </xsl:when>
    <xsl:when test="$IN_PREC &gt; $NO_PREC and $IN_PREC &lt;
                    $FUNCTION_PREC and not($SEM_SW=$SEM_ALL) and
                    not($SEM_SW=$SEM_XREF) and
                    not($SEM_SW=$SEM_XREF_EXT)">
      <xsl:apply-templates select="." mode="minus">
        <xsl:with-param name="PARAM" select="$PARAM"/>
        <xsl:with-param name="PAREN" select="$PAREN"/>
        <xsl:with-param name="PAR_NO_IGNORE" select="$PAR_NO_IGNORE"/>
      </xsl:apply-templates>
    </xsl:when>
    <xsl:otherwise>
      <mrow>
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                    @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="." mode="minus">
          <xsl:with-param name="PARAM" select="$PARAM"/>
          <xsl:with-param name="PAREN" select="$PAREN"/>
          <xsl:with-param name="PAR_NO_IGNORE"
                          select="$PAR_NO_IGNORE"/>
        </xsl:apply-templates>
      </mrow>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match = "mml:apply[mml:minus[1]]" mode="minus">
  <xsl:param name="PARAM" select="$NO_PARAM"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:if test="not(*[3])">
    <mo>-</mo>
    <xsl:apply-templates select="*[2]" mode = "semantics">
      <xsl:with-param name="IN_PREC" select="$NEG_PREC"/>
      <xsl:with-param name="PAREN" select="$PAREN"/>
      <xsl:with-param name="PAR_NO_IGNORE" select="$NO"/>
    </xsl:apply-templates>
  </xsl:if>
  <xsl:if test="*[3]">
    <xsl:apply-templates select="*[2]" mode = "semantics">
      <xsl:with-param name="IN_PREC" select="$MINUS_PREC"/>
      <xsl:with-param name="PARAM" select="$PARAM"/>
      <xsl:with-param name="PAREN" select="$PAREN"/>
```

```
      <xsl:with-param name="PAR_NO_IGNORE" select="$PAR_NO_IGNORE"/>
    </xsl:apply-templates>
    <mo>-</mo>
    <xsl:apply-templates select="*[3]" mode = "semantics">
      <xsl:with-param name="IN_PREC" select="$MINUS_PREC"/>
      <xsl:with-param name="PARAM" select="$PAR_SAME"/>
      <xsl:with-param name="PAREN" select="$PAREN"/>
      <xsl:with-param name="PAR_NO_IGNORE" select="$NO"/>
    </xsl:apply-templates>
  </xsl:if>
</xsl:template>

<xsl:template match = "mml:apply[mml:plus[1]]">
  <xsl:param name="IN_PREC" select="$NO_PREC"/>
  <xsl:param name="PARAM" select="$NO_PARAM"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:choose>
    <xsl:when test="$IN_PREC &gt; $PLUS_PREC or $IN_PREC=$PLUS_PREC and
                                                    $PARAM=$PAR_SAME">
      <mfenced separators="">
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                  @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="." mode="plus">
          <xsl:with-param name="PARAM" select="$IN_PREC"/>
          <xsl:with-param name="PAREN" select="$PAR_YES"/>
          <xsl:with-param name="PAR_NO_IGNORE"
                          select="$PAR_NO_IGNORE"/>
        </xsl:apply-templates>
      </mfenced>
    </xsl:when>
    <xsl:when test="$IN_PREC &gt; $NO_PREC and $IN_PREC &lt;
                    $FUNCTION_PREC and not($SEM_SW=$SEM_ALL) and
                    not($SEM_SW=$SEM_XREF) and
                    not($SEM_SW=$SEM_XREF_EXT)">
      <xsl:apply-templates select="." mode="plus">
        <xsl:with-param name="PARAM" select="$PARAM"/>
        <xsl:with-param name="PAREN" select="$PAREN"/>
        <xsl:with-param name="PAR_NO_IGNORE" select="$PAR_NO_IGNORE"/>
      </xsl:apply-templates>
    </xsl:when>
    <xsl:otherwise>
      <mrow>
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                  @id">
          <xsl:attribute name="xref">
```

```
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="." mode="plus">
          <xsl:with-param name="PARAM" select="$IN_PREC"/>
          <xsl:with-param name="PAREN" select="$PAREN"/>
          <xsl:with-param name="PAR_NO_IGNORE"
                          select="$PAR_NO_IGNORE"/>
        </xsl:apply-templates>
      </mrow>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match = "mml:apply[mml:plus[1]]" mode="plus">
  <xsl:param name="PARAM" select="$NO_PARAM"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:if test="*[2]">
    <xsl:apply-templates select="*[2]" mode = "semantics">
      <xsl:with-param name="IN_PREC" select="$PLUS_PREC"/>
      <xsl:with-param name="PARAM" select="$PARAM"/>
      <xsl:with-param name="PAREN" select="$PAREN"/>
      <xsl:with-param name="PAR_NO_IGNORE" select="$PAR_NO_IGNORE"/>
    </xsl:apply-templates>
    <xsl:for-each select = "*[position()>2]">
      <xsl:choose>
        <xsl:when test=". &lt; 0">
          <mo>-</mo>
          <mn> <xsl:value-of select="-."/> </mn>
        </xsl:when>
        <xsl:when test="self::mml:apply[mml:minus[1]] and not(*[3])">
          <xsl:apply-templates select="." mode = "semantics">
            <xsl:with-param name="IN_PREC" select="$PLUS_PREC"/>
            <xsl:with-param name="PAREN" select="$PAREN"/>
            <xsl:with-param name="PAR_NO_IGNORE" select="$NO"/>
          </xsl:apply-templates>
        </xsl:when>
        <xsl:otherwise>
          <mo>+</mo>
          <xsl:apply-templates select="." mode = "semantics">
            <xsl:with-param name="IN_PREC" select="$PLUS_PREC"/>
            <xsl:with-param name="PAREN" select="$PAREN"/>
            <xsl:with-param name="PAR_NO_IGNORE" select="$NO"/>
          </xsl:apply-templates>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each>
  </xsl:if>
</xsl:template>
```

```
<xsl:template match = "mml:apply[mml:power[1]]">
  <xsl:choose>
    <xsl:when test="*[2]=mml:apply[mml:ln[1] | mml:log[1] | mml:abs[1]
                                 | mml:gcd[1] | mml:sin[1] | mml:cos[1]
                                 | mml:tan[1] | mml:sec[1] | mml:csc[1]
                                 | mml:cot[1] | mml:sinh[1]
                                 | mml:cosh[1] | mml:tanh[1]
                                 | mml:sech[1] | mml:csch[1]
                                 | mml:coth[1] | mml:arcsin[1]
                                 | mml:arccos[1] | mml:arctan[1]]">
      <xsl:apply-templates select="*[2]" mode = "semantics"/>
    </xsl:when>
    <xsl:otherwise>
      <msup>
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                  @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select = "*[2]" mode = "semantics">
          <xsl:with-param name="IN_PREC" select="$FUNCTION_PREC"/>
          <xsl:with-param name="PAR_NO_IGNORE" select="$NO"/>
        </xsl:apply-templates>
        <xsl:apply-templates select = "*[3]" mode = "semantics"/>
      </msup>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match = "mml:apply[mml:rem[1] | mml:divide[1]]">
  <xsl:param name="IN_PREC" select="$NO_PREC"/>
  <xsl:param name="PARAM" select="$NO_PARAM"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:choose>
    <xsl:when test="$IN_PREC &gt; $DIV_PREC or $IN_PREC=$DIV_PREC and
                                                  $PARAM=$PAR_SAME">
      <mfenced separators="">
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                  @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="." mode="remdiv">
          <xsl:with-param name="PARAM" select="$IN_PREC"/>
          <xsl:with-param name="PAREN" select="$PAR_YES"/>
          <xsl:with-param name="PAR_NO_IGNORE"
```

```
                        select="$PAR_NO_IGNORE"/>
        </xsl:apply-templates>
      </mfenced>
    </xsl:when>
    <xsl:when test="$IN_PREC &gt; $NO_PREC and $IN_PREC &lt;
                    $FUNCTION_PREC and not($SEM_SW=$SEM_ALL)
                    and not($SEM_SW=$SEM_XREF)
                    and not($SEM_SW=$SEM_XREF_EXT)">
      <xsl:apply-templates select="." mode="remdiv">
        <xsl:with-param name="PARAM" select="$PARAM"/>
        <xsl:with-param name="PAREN" select="$PAREN"/>
        <xsl:with-param name="PAR_NO_IGNORE" select="$PAR_NO_IGNORE"/>
      </xsl:apply-templates>
    </xsl:when>
    <xsl:otherwise>
      <mrow>
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                    @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="." mode="remdiv">
          <xsl:with-param name="PARAM" select="$IN_PREC"/>
          <xsl:with-param name="PAREN" select="$PAREN"/>
          <xsl:with-param name="PAR_NO_IGNORE"
                          select="$PAR_NO_IGNORE"/>
        </xsl:apply-templates>
      </mrow>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match = "mml:apply[mml:rem[1] | mml:divide[1]]"
              mode="remdiv">
  <xsl:param name="PARAM" select="$NO_PARAM"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:apply-templates select = "*[2]" mode = "semantics">
    <xsl:with-param name="IN_PREC" select="$DIV_PREC"/>
      <xsl:with-param name="PARAM" select="$PARAM"/>
      <xsl:with-param name="PAREN" select="$PAREN"/>
      <xsl:with-param name="PAR_NO_IGNORE" select="$PAR_NO_IGNORE"/>
  </xsl:apply-templates>
  <mo>
    <xsl:if test="mml:rem">
      <xsl:value-of select="'%'"/>
    </xsl:if>
    <xsl:if test="mml:divide">
      <xsl:value-of select="'/'"/>
```

```
    </xsl:if>
  </mo>
  <xsl:apply-templates select = "*[3]" mode = "semantics">
    <xsl:with-param name="IN_PREC" select="$DIV_PREC"/>
    <xsl:with-param name="PARAM" select="$PAR_SAME"/>
    <xsl:with-param name="PAREN" select="$PAREN"/>
    <xsl:with-param name="PAR_NO_IGNORE" select="$NO"/>
  </xsl:apply-templates>
</xsl:template>

<xsl:template match = "mml:apply[mml:times[1]]">
  <xsl:param name="IN_PREC" select="$NO_PREC"/>
  <xsl:param name="PARAM" select="$NO_PARAM"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:choose>
    <xsl:when test="$IN_PREC &gt; $MUL_PREC or $IN_PREC=$MUL_PREC and
                                                $PARAM=$PAR_SAME">
      <mfenced separators="">
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                              @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="." mode="times">
          <xsl:with-param name="PARAM" select="$IN_PREC"/>
          <xsl:with-param name="PAREN" select="$PAR_YES"/>
          <xsl:with-param name="PAR_NO_IGNORE"
                          select="$PAR_NO_IGNORE"/>
        </xsl:apply-templates>
      </mfenced>
    </xsl:when>
    <xsl:when test="$IN_PREC &gt; $NO_PREC and $IN_PREC &lt;
                    $FUNCTION_PREC and not($SEM_SW=$SEM_ALL)
                    and not($SEM_SW=$SEM_XREF)
                    and not($SEM_SW=$SEM_XREF_EXT)">
      <xsl:apply-templates select="." mode="times">
        <xsl:with-param name="PARAM" select="$PARAM"/>
        <xsl:with-param name="PAREN" select="$PAREN"/>
        <xsl:with-param name="PAR_NO_IGNORE" select="$PAR_NO_IGNORE"/>
      </xsl:apply-templates>
    </xsl:when>
    <xsl:otherwise>
      <mrow>
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                              @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
```

```
        </xsl:if>
        <xsl:apply-templates select="." mode="times">
         <xsl:with-param name="PARAM" select="$IN_PREC"/>
          <xsl:with-param name="PAREN" select="$PAREN"/>
          <xsl:with-param name="PAR_NO_IGNORE"
                          select="$PAR_NO_IGNORE"/>
        </xsl:apply-templates>
      </mrow>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match = "mml:apply[mml:times[1]]" mode="times">
  <xsl:param name="PARAM" select="$NO_PARAM"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:apply-templates select="*[2]" mode = "semantics">
    <xsl:with-param name="IN_PREC" select="$MUL_PREC"/>
    <xsl:with-param name="PARAM" select="$PARAM"/>
    <xsl:with-param name="PAREN" select="$PAREN"/>
    <xsl:with-param name="PAR_NO_IGNORE" select="$PAR_NO_IGNORE"/>
  </xsl:apply-templates>
  <xsl:if test="*[3]">
    <xsl:for-each select = "*[position()>2]">
      <mo> <mchar name="InvisibleTimes"/> </mo>
      <xsl:apply-templates select="." mode = "semantics">
        <xsl:with-param name="IN_PREC" select="$MUL_PREC"/>
        <xsl:with-param name="PAREN" select="$PAREN"/>
        <xsl:with-param name="PAR_NO_IGNORE" select="$NO"/>
      </xsl:apply-templates>
    </xsl:for-each>
  </xsl:if>
</xsl:template>

<xsl:template match = "mml:apply[mml:root[1]]">
  <msqrt>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                              @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="*[2]=mml:degree">
      <xsl:apply-templates select="*[3]" mode = "semantics">
        <xsl:with-param name="IN_PREC" select="$FUNCTION_PREC"/>
        <xsl:with-param name="PAR_NO_IGNORE" select="$NO"/>
      </xsl:apply-templates>
      <xsl:apply-templates select="*[2]" mode = "semantics"/>
    </xsl:if>
    <xsl:if test="not(*[2]=mml:degree)">
```

```
      <xsl:apply-templates select="*[2]" mode = "semantics">
        <xsl:with-param name="IN_PREC" select="$FUNCTION_PREC"/>
        <xsl:with-param name="PAR_NO_IGNORE" select="$NO"/>
      </xsl:apply-templates>
      <mn>2</mn>
    </xsl:if>
  </msqrt>
</xsl:template>

<xsl:template match = "mml:apply[mml:gcd[1]]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="not(parent::mml:apply[mml:power[1]])">
      <mo>gcd</mo>
    </xsl:if>
    <xsl:if test="parent::mml:apply[mml:power[1]]">
      <msup>
        <mo>gcd</mo>
        <xsl:apply-templates select = "../*[3]" mode = "semantics"/>
      </msup>
    </xsl:if>
    <mfenced separators=",">
      <xsl:for-each select = "*[position()>1]">
        <xsl:apply-templates select = "." mode="semantics"/>
      </xsl:for-each>
    </mfenced>
  </mrow>
</xsl:template>

<xsl:template match = "mml:apply[mml:and[1]]">
  <xsl:param name="IN_PREC" select="$NO_PREC"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:choose>
    <xsl:when test="$IN_PREC &gt; $AND_PREC">
      <mfenced separators="">
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="." mode="and">
          <xsl:with-param name="PARAM" select="$IN_PREC"/>
          <xsl:with-param name="PAREN" select="$PAR_YES"/>
```

```
        </xsl:apply-templates>
      </mfenced>
    </xsl:when>
    <xsl:when test="$IN_PREC &gt; $NO_PREC and $IN_PREC &lt;
                    $FUNCTION_PREC and not($SEM_SW=$SEM_ALL)
                    and not($SEM_SW=$SEM_XREF)
                    and not($SEM_SW=$SEM_XREF_EXT)">
      <xsl:apply-templates select="." mode="and">
        <xsl:with-param name="PARAM" select="$IN_PREC"/>
        <xsl:with-param name="PAREN" select="$PAREN"/>
        <xsl:with-param name="PAR_NO_IGNORE" select="$PAR_NO_IGNORE"/>
      </xsl:apply-templates>
    </xsl:when>
    <xsl:otherwise>
      <mrow>
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                    @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="." mode="and">
          <xsl:with-param name="PARAM" select="$IN_PREC"/>
          <xsl:with-param name="PAREN" select="$PAREN"/>
          <xsl:with-param name="PAR_NO_IGNORE"
                          select="$PAR_NO_IGNORE"/>
        </xsl:apply-templates>
      </mrow>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match = "mml:apply[mml:and[1]]" mode="and">
  <xsl:param name="PARAM" select="$NO_PARAM"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:apply-templates select="*[2]" mode = "semantics">
    <xsl:with-param name="IN_PREC" select="$AND_PREC"/>
    <xsl:with-param name="PARAM" select="$PARAM"/>
    <xsl:with-param name="PAREN" select="$PAREN"/>
    <xsl:with-param name="PAR_NO_IGNORE" select="$PAR_NO_IGNORE"/>
  </xsl:apply-templates>
  <xsl:for-each select = "*[position()>2]">
    <mo> <mchar name="wedge"/> </mo>
    <xsl:apply-templates select="." mode = "semantics">
      <xsl:with-param name="IN_PREC" select="$AND_PREC"/>
      <xsl:with-param name="PAREN" select="$PAREN"/>
      <xsl:with-param name="PAR_NO_IGNORE" select="$NO"/>
    </xsl:apply-templates>
  </xsl:for-each>
```

```
</xsl:template>

<xsl:template match = "mml:apply[mml:or[1]]">
  <xsl:param name="IN_PREC" select="$NO_PREC"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:choose>
    <xsl:when test="$IN_PREC &gt; $OR_PREC">
      <mfenced separators="">
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                    @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="." mode="or">
          <xsl:with-param name="PARAM" select="$IN_PREC"/>
          <xsl:with-param name="PAREN" select="$PAR_YES"/>
        </xsl:apply-templates>
      </mfenced>
    </xsl:when>
    <xsl:when test="$IN_PREC &gt; $NO_PREC and $IN_PREC &lt;
                    $FUNCTION_PREC and not($SEM_SW=$SEM_ALL)
                    and not($SEM_SW=$SEM_XREF)
                    and not($SEM_SW=$SEM_XREF_EXT)">
      <xsl:apply-templates select="." mode="or">
        <xsl:with-param name="PARAM" select="$IN_PREC"/>
        <xsl:with-param name="PAREN" select="$PAREN"/>
        <xsl:with-param name="PAR_NO_IGNORE" select="$PAR_NO_IGNORE"/>
      </xsl:apply-templates>
    </xsl:when>
    <xsl:otherwise>
      <mrow>
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                    @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="." mode="or">
          <xsl:with-param name="PARAM" select="$IN_PREC"/>
          <xsl:with-param name="PAREN" select="$PAREN"/>
          <xsl:with-param name="PAR_NO_IGNORE"
                          select="$PAR_NO_IGNORE"/>
        </xsl:apply-templates>
      </mrow>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

```
<xsl:template match = "mml:apply[mml:or[1]]" mode="or">
  <xsl:param name="PARAM" select="$NO_PARAM"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:apply-templates select="*[2]" mode = "semantics">
    <xsl:with-param name="IN_PREC" select="$OR_PREC"/>
    <xsl:with-param name="PARAM" select="$PARAM"/>
    <xsl:with-param name="PAREN" select="$PAREN"/>
    <xsl:with-param name="PAR_NO_IGNORE" select="$PAR_NO_IGNORE"/>
  </xsl:apply-templates>
  <xsl:for-each select = "*[position()>2]">
    <mo> <mchar name="vee"/> </mo>
    <xsl:apply-templates select="." mode = "semantics">
      <xsl:with-param name="IN_PREC" select="$OR_PREC"/>
      <xsl:with-param name="PAREN" select="$PAREN"/>
      <xsl:with-param name="PAR_NO_IGNORE" select="$NO"/>
    </xsl:apply-templates>
  </xsl:for-each>
</xsl:template>

<xsl:template match = "mml:apply[mml:xor[1]]">
  <xsl:param name="IN_PREC" select="$NO_PREC"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:choose>
    <xsl:when test="$IN_PREC &gt; $XOR_PREC">
      <mfenced separators="">
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                    @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="." mode="xor">
          <xsl:with-param name="PARAM" select="$IN_PREC"/>
          <xsl:with-param name="PAREN" select="$PAR_YES"/>
          <xsl:with-param name="PAR_NO_IGNORE"
                          select="$PAR_NO_IGNORE"/>
        </xsl:apply-templates>
      </mfenced>
    </xsl:when>
    <xsl:when test="$IN_PREC &gt; $NO_PREC and $IN_PREC &lt;
                    $FUNCTION_PREC and not($SEM_SW=$SEM_ALL)">
      <xsl:apply-templates select="." mode="xor">
        <xsl:with-param name="PARAM" select="$IN_PREC"/>
        <xsl:with-param name="PAREN" select="$PAREN"/>
      </xsl:apply-templates>
    </xsl:when>
    <xsl:otherwise>
      <mrow>
```

```
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                    @id">
        ' <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="." mode="xor">
          <xsl:with-param name="PARAM" select="$IN_PREC"/>
          <xsl:with-param name="PAREN" select="$PAREN"/>
          <xsl:with-param name="PAR_NO_IGNORE"
                          select="$PAR_NO_IGNORE"/>
        </xsl:apply-templates>
      </mrow>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match = "mml:apply[mml:xor[1]]" mode="xor">
  <xsl:param name="PARAM" select="$NO_PARAM"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:apply-templates select="*[2]" mode = "semantics">
    <xsl:with-param name="IN_PREC" select="$XOR_PREC"/>
    <xsl:with-param name="PARAM" select="$PARAM"/>
    <xsl:with-param name="PAREN" select="$PAREN"/>
    <xsl:with-param name="PAR_NO_IGNORE" select="$PAR_NO_IGNORE"/>
  </xsl:apply-templates>
  <xsl:for-each select = "*[position()>2]">
    <mo> <mchar name="xor"/> </mo>
    <xsl:apply-templates select="." mode = "semantics">
      <xsl:with-param name="IN_PREC" select="$XOR_PREC"/>
      <xsl:with-param name="PAREN" select="$PAREN"/>
      <xsl:with-param name="PAR_NO_IGNORE" select="$NO"/>
    </xsl:apply-templates>
  </xsl:for-each>
</xsl:template>

<xsl:template match = "mml:apply[mml:not[1]]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                  @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <mo>not</mo>
    <xsl:apply-templates select = "*[2]" mode = "semantics">
      <xsl:with-param name="IN_PREC" select="$FUNCTION_PREC"/>
    </xsl:apply-templates>
  </mrow>
```

```
</xsl:template>

<xsl:template match = "mml:apply[mml:forall[1]]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <mo>for all</mo>
    <xsl:if test="count(mml:bvar) &gt; 1">
      <mfenced separators=",">
        <xsl:for-each select = "mml:bvar">
          <xsl:apply-templates select = "." mode="semantics"/>
        </xsl:for-each>
      </mfenced>
    </xsl:if>
    <xsl:if test="count(mml:bvar)=1">
      <xsl:apply-templates select = "mml:bvar" mode="semantics"/>
    </xsl:if>
    <xsl:if test="mml:condition">
      <mo>:</mo>
      <xsl:apply-templates select="mml:condition/*" mode="semantics"/>
      <mo>,</mo>
    </xsl:if>
    <xsl:apply-templates select = "*[position()>1 and
                                     not(self::mml:bvar) and
                                     not(self::mml:condition)]"
                         mode = "semantics"/>
  </mrow>
</xsl:template>

<xsl:template match = "mml:apply[mml:exists[1]]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <mo> <mchar name="Exists"/> </mo>
    <xsl:if test="count(mml:bvar) &gt; 1">
      <mfenced separators=",">
        <xsl:for-each select = "mml:bvar">
          <xsl:apply-templates select = "." mode="semantics"/>
        </xsl:for-each>
      </mfenced>
    </xsl:if>
    <xsl:if test="count(mml:bvar)=1">
```

```
      <xsl:apply-templates select = "mml:bvar" mode="semantics"/>
    </xsl:if>
    <xsl:if test="mml:condition">
      <mo>,</mo>
      <xsl:apply-templates select="mml:condition/*" mode="semantics"/>
    </xsl:if>
    <xsl:if test="*[position()>1 and not(self::mml:bvar) and
                  not(self::mml:condition)]">
      <mo>:</mo>
      <xsl:apply-templates select = "*[position()>1 and
                                       not(self::mml:bvar) and
                                       not(self::mml:condition)]"
                           mode = "semantics"/>
    </xsl:if>
  </mrow>
</xsl:template>

<xsl:template match = "mml:apply[mml:abs[1]]">
  <xsl:if test="not(parent::mml:apply[mml:power[1]])">
    <mfenced open="|" close="|" separators="">
      <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                  @id">
        <xsl:attribute name="xref">
          <xsl:value-of select="@id"/>
        </xsl:attribute>
      </xsl:if>
      <xsl:apply-templates select="*[position()>1]" mode="semantics"/>
    </mfenced>
  </xsl:if>
  <xsl:if test="parent::mml:apply[mml:power[1]]">
    <msup>
      <mfenced open="|" close="|" separators="">
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                    @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="*[position()>1]"
                             mode="semantics"/>
      </mfenced>
      <xsl:apply-templates select = "../*[3]" mode = "semantics"/>
    </msup>
  </xsl:if>
</xsl:template>

<xsl:template match = "mml:apply[mml:conjugate[1]]">
  <mover>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                          @id">
```

```
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <mrow>
      <xsl:apply-templates select="*[position()>1]" mode="semantics"/>
    </mrow>
    <mo> <mchar name="UnderBar"/> </mo>
  </mover>
</xsl:template>

<xsl:template match = "mml:apply[mml:arg[1] | mml:real[1] |
                        mml:imaginary[1]]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                              @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <mo>
      <xsl:if test="mml:arg">
        <xsl:value-of select="'Arg'"/>
      </xsl:if>
      <xsl:if test="mml:real">
        <xsl:value-of select="'Re'"/>
      </xsl:if>
      <xsl:if test="mml:imaginary">
        <xsl:value-of select="'Im'"/>
      </xsl:if>
    </mo>
    <mfenced separators="">
      <xsl:apply-templates select = "*[2]" mode = "semantics"/>
    </mfenced>
  </mrow>
</xsl:template>


<!-- ***************** RELATIONS ***************** -->

<xsl:template match = "mml:apply[mml:neq | mml:approx | mml:tendsto |
                                 mml:implies | mml:in | mml:notin |
                                 mml:notsubset | mml:notprsubset |
                                 mml:subset | mml:prsubset | mml:eq |
                                 mml:gt | mml:lt | mml:geq | mml:leq |
                                 mml:equivalent]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                              @id">
      <xsl:attribute name="xref">
```

```
      <xsl:value-of select="@id"/>
    </xsl:attribute>
  </xsl:if>
  <xsl:if test="*[1]=mml:neq or *[1]=mml:approx or *[1]=mml:tendsto
              or *[1]=mml:implies or *[1]=mml:in or *[1]=mml:notin
              or *[1]=mml:notsubset or *[1]=mml:notprsubset">
    <xsl:apply-templates select = "*[2]" mode = "semantics"/>
    <mo>
      <xsl:if test="*[1]=mml:neq">
        <mchar name="NotEqual"/>
      </xsl:if>
      <xsl:if test="*[1]=mml:approx">
        <mchar name="approxeq"/>
      </xsl:if>
      <xsl:if test="*[1]=mml:tendsto">
        <mchar name="RightArrow"/>
      </xsl:if>
      <xsl:if test="*[1]=mml:implies">
        <mchar name="DoubleRightArrow"/>
      </xsl:if>
      <xsl:if test="*[1]=mml:in">
        <mchar name="Element"/>
      </xsl:if>
      <xsl:if test="*[1]=mml:notin">
        <mchar name="NotElement"/>
      </xsl:if>
      <xsl:if test="*[1]=mml:notsubset">
        <mchar name="NotSubset"/>
      </xsl:if>
      <xsl:if test="*[1]=mml:notprsubset">
        <mchar name="NotSubsetEqual"/>
      </xsl:if>
    </mo>
    <xsl:apply-templates select = "*[3]" mode = "semantics"/>
    <xsl:if test="*[1]=mml:tendsto and
                  mml:tendsto[1][@type='below']">
      <mo>-</mo>
    </xsl:if>
    <xsl:if test="*[1]=mml:tendsto and
                  mml:tendsto[1][@type='above']">
      <mo>+</mo>
    </xsl:if>
  </xsl:if>
  <xsl:if test="*[1]=mml:subset or *[1]=mml:prsubset or *[1]=mml:eq
              or *[1]=mml:gt or *[1]=mml:lt or *[1]=mml:geq
              or *[1]=mml:leq or *[1]=mml:equivalent">
    <xsl:apply-templates select = "*[2]" mode="semantics"/>
    <xsl:for-each select = "*[position()>2]">
      <mo>
        <xsl:if test="../*[self::mml:subset][1]">
```

```
            <mchar name="SubsetEqual"/>
          </xsl:if>
          <xsl:if test="../*[self::mml:prsubset][1]">
            <mchar name="subset"/>
          </xsl:if>
          <xsl:if test="../*[self::mml:eq][1]">
            <xsl:value-of select="'='"/>
          </xsl:if>
          <xsl:if test="../*[self::mml:gt][1]">
            <xsl:value-of select="'&gt;'"/>
          </xsl:if>
          <xsl:if test="../*[self::mml:lt][1]">
            <xsl:value-of select="'&lt;'"/>
          </xsl:if>
          <xsl:if test="../*[self::mml:geq][1]">
            <mchar name="geq"/>
          </xsl:if>
          <xsl:if test="../*[self::mml:leq][1]">
            <mchar name="leq"/>
          </xsl:if>
          <xsl:if test="../*[self::mml:equivalent][1]">
            <mchar name="Congruent"/>
          </xsl:if>
        </mo>
        <xsl:apply-templates select = "." mode="semantics"/>
      </xsl:for-each>
    </xsl:if>
  </mrow>
</xsl:template>


<!-- ***************** CALCULUS ***************** -->

<xsl:template match = "mml:apply[*[1][self::mml:ln]]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                          @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:choose>
      <xsl:when test="parent::mml:apply[mml:power[1]]">
        <msup>
          <mo>ln</mo>
          <xsl:apply-templates select = "../*[3]" mode = "semantics"/>
        </msup>
      </xsl:when>
      <xsl:otherwise>
        <mo>ln</mo>
```

```
      </xsl:otherwise>
    </xsl:choose>
    <mo>' <mchar name='ApplyFunction'/> </mo>
    <xsl:apply-templates select = "*[2]" mode = "semantics">
      <xsl:with-param name="IN_PREC" select="$FUNCTION_PREC"/>
    </xsl:apply-templates>
  </mrow>
</xsl:template>

<xsl:template match = "mml:apply[mml:log[1]]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                              @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:choose>
      <xsl:when test="parent::mml:apply[mml:power[1]]">
        <xsl:if test="not(*[2]=mml:logbase)">
          <msup>
            <mo>log</mo>
            <xsl:apply-templates select = "../*[3]" mode="semantics"/>
          </msup>
        </xsl:if>
        <xsl:if test="*[2]=mml:logbase">
          <msubsup>
            <mo>log</mo>
            <xsl:apply-templates select = "../*[3]" mode="semantics"/>
            <xsl:apply-templates select="mml:logbase"
                                 mode="semantics"/>
          </msubsup>
        </xsl:if>
      </xsl:when>
      <xsl:otherwise>
        <xsl:if test="not(*[2]=mml:logbase)">
          <mo>log</mo>
        </xsl:if>
        <xsl:if test="*[2]=mml:logbase">
          <msub>
            <mo>log</mo>
            <xsl:apply-templates select = "mml:logbase"
                                 mode = "semantics"/>
          </msub>
        </xsl:if>
      </xsl:otherwise>
    </xsl:choose>
    <mo> <mchar name='ApplyFunction'/> </mo>
    <xsl:if test="*[2]=mml:logbase">
      <xsl:apply-templates select = "*[3]" mode = "semantics">
```

```
        <xsl:with-param name="IN_PREC" select="$FUNCTION_PREC"/>
      </xsl:apply-templates>
    </xsl:if>
    <xsl:if test="not(*[2]=mml:logbase)">
      <xsl:apply-templates select = "*[2]" mode = "semantics">
        <xsl:with-param name="IN_PREC" select="$FUNCTION_PREC"/>
      </xsl:apply-templates>
    </xsl:if>
  </mrow>
</xsl:template>

<xsl:template match = "mml:apply[mml:diff[1]]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                              @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="*[2]=mml:bvar and mml:bvar[*[2]=mml:degree]">
      <mfrac>
        <msup>
          <mo>d</mo>
          <xsl:apply-templates select = "mml:bvar/mml:degree"
                               mode="semantics"/>
        </msup>
        <mrow>
          <mo>d</mo>
          <msup>
            <xsl:apply-templates select="mml:bvar/*[1]"
                                 mode="semantics"/>
            <xsl:apply-templates select="mml:bvar/mml:degree"
                                 mode="semantics"/>
          </msup>
        </mrow>
      </mfrac>
    </xsl:if>
    <xsl:if test="*[2]=mml:bvar and not(mml:bvar[*[2]=mml:degree])">
      <mfrac>
        <mo>d</mo>
        <mrow>
          <mo>d</mo>
          <xsl:apply-templates select = "mml:bvar/*[1]"
                               mode = "semantics"/>
        </mrow>
      </mfrac>
    </xsl:if>
    <xsl:apply-templates select = "*[3]" mode = "semantics"/>
  </mrow>
</xsl:template>
```

```
<xsl:template match = "mml:apply[mml:partialdiff[1]]">
  <mrow>'
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:for-each select = "mml:bvar">
      <xsl:if test="*[last()]=mml:degree">
        <mfrac>
          <msup>
             <mo>d</mo>
             <xsl:apply-templates select = "mml:degree"
                                  mode = "semantics"/>
          </msup>
          <mrow>
            <mo>d</mo>
            <msup>
              <xsl:apply-templates select = "*[1]" mode = "semantics"/>
              <xsl:apply-templates select = "mml:degree"
                                   mode = "semantics"/>
            </msup>
          </mrow>
        </mfrac>
      </xsl:if>
      <xsl:if test="not(*[last()]=mml:degree)">
        <mfrac>
          <mo>d</mo>
          <mrow>
            <mo>d</mo>
            <xsl:apply-templates select = "*[1]" mode = "semantics"/>
          </mrow>
        </mfrac>
      </xsl:if>
    </xsl:for-each>
    <xsl:apply-templates select = "*[last()]" mode = "semantics"/>
  </mrow>
</xsl:template>

<xsl:template match = "mml:lowlimit | mml:uplimit | mml:bvar |
                       mml:degree | mml:logbase">
  <xsl:apply-templates select="*" mode = "semantics"/>
</xsl:template>

<xsl:template match = "mml:apply[mml:divergence[1] | mml:grad[1] |
                       mml:curl[1]]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
```

```
                                                                    @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <mo>
      <xsl:if test="*[1]=mml:divergence">
        <xsl:value-of select="'div'"/>
      </xsl:if>
      <xsl:if test="*[1]=mml:grad">
        <xsl:value-of select="'grad'"/>
      </xsl:if>
      <xsl:if test="*[1]=mml:curl">
        <xsl:value-of select="'curl'"/>
      </xsl:if>
    </mo>
    <xsl:choose>
      <xsl:when test="*[2]=mml:ci">
        <xsl:apply-templates select = "*[2]" mode = "semantics"/>
      </xsl:when>
      <xsl:otherwise>
        <mfenced separators="">
          <xsl:apply-templates select = "*[2]" mode = "semantics"/>
        </mfenced>
      </xsl:otherwise>
    </xsl:choose>
  </mrow>
</xsl:template>

<xsl:template match = "mml:apply[mml:laplacian[1]]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                    @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <msup>
      <mo> <mchar name="Delta"/> </mo>
      <mn>2</mn>
    </msup>
    <xsl:apply-templates select = "*[2]" mode = "semantics"/>
  </mrow>
</xsl:template>


<!-- ***************** SET THEORY ***************** -->

<xsl:template match = "mml:set | mml:list">
  <mfenced open="{{" close="}}" separators="">
```

```
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                          @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="*[1]=mml:bvar and *[2]=mml:condition">
      <xsl:apply-templates select="mml:bvar" mode = "semantics"/>
      <mo>|</mo>
      <xsl:apply-templates select="mml:condition" mode = "semantics"/>
    </xsl:if>
    <xsl:if test="*[1]=mml:condition and not(child::mml:bvar)">
      <mfenced open="" close="" separators=",">
        <xsl:for-each select = "*[not(self::mml:condition) and
                                  not(self::mml:bvar)]">
          <xsl:apply-templates select = "." mode="semantics"/>
        </xsl:for-each>
      </mfenced>
      <mo>|</mo>
      <xsl:apply-templates select="mml:condition" mode = "semantics"/>
    </xsl:if>
    <xsl:if test="not(child::mml:bvar) and not(child::mml:condition)">
      <mfenced open="" close="" separators=",">
        <xsl:for-each select = "*">
          <xsl:apply-templates select = "." mode="semantics"/>
        </xsl:for-each>
      </mfenced>
    </xsl:if>
  </mfenced>
</xsl:template>

<xsl:template match = "mml:apply[mml:union[1]]">
  <xsl:param name="IN_PREC" select="$NO_PREC"/>
  <xsl:param name="PARAM" select="$NO_PARAM"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:choose>
    <xsl:when test="$IN_PREC &gt; $UNION_PREC or $IN_PREC=$UNION_PREC
                    and $PARAM=$PAR_SAME">
      <mfenced separators="">
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                          @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="." mode="union">
          <xsl:with-param name="PARAM" select="$PARAM"/>
          <xsl:with-param name="PAREN" select="$PAREN"/>
          <xsl:with-param name="PAR_NO_IGNORE"
```

```
                        select="$PAR_NO_IGNORE"/>
        </xsl:apply-templates>
      </mfenced>
    </xsl:when>
    <xsl:when test="$IN_PREC &gt; $NO_PREC and $IN_PREC &lt;
                    $FUNCTION_PREC and not($SEM_SW=$SEM_ALL)
                    and not($SEM_SW=$SEM_XREF)
                    and not($SEM_SW=$SEM_XREF_EXT)">
      <xsl:apply-templates select="." mode="union">
        <xsl:with-param name="PARAM" select="$PARAM"/>
        <xsl:with-param name="PAREN" select="$PAREN"/>
        <xsl:with-param name="PAR_NO_IGNORE" select="$PAR_NO_IGNORE"/>
      </xsl:apply-templates>
    </xsl:when>
    <xsl:otherwise>
      <mrow>
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                    @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="." mode="union">
          <xsl:with-param name="PARAM" select="$PARAM"/>
          <xsl:with-param name="PAREN" select="$PAREN"/>
          <xsl:with-param name="PAR_NO_IGNORE"
                          select="$PAR_NO_IGNORE"/>
        </xsl:apply-templates>
      </mrow>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match = "mml:apply[mml:union[1]]" mode="union">
  <xsl:param name="PARAM" select="$NO_PARAM"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:apply-templates select = "*[2]" mode="semantics">
    <xsl:with-param name="IN_PREC" select="$UNION_PREC"/>
    <xsl:with-param name="PARAM" select="$PARAM"/>
    <xsl:with-param name="PAREN" select="$PAREN"/>
    <xsl:with-param name="PAR_NO_IGNORE" select="$PAR_NO_IGNORE"/>
  </xsl:apply-templates>
  <xsl:for-each select = "*[position()>2]">
    <mo> <mchar name="Union"/> </mo>
    <xsl:apply-templates select = "." mode="semantics">
      <xsl:with-param name="IN_PREC" select="$UNION_PREC"/>
      <xsl:with-param name="PAREN" select="$PAREN"/>
      <xsl:with-param name="PAR_NO_IGNORE" select="$NO"/>
    </xsl:apply-templates>
```

```
  </xsl:for-each>
</xsl:template>

<xsl:template match = "mml:apply[mml:intersect[1]]">
  <xsl:param name="IN_PREC" select="$NO_PREC"/>
  <xsl:param name="PARAM" select="$NO_PARAM"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:choose>
    <xsl:when test="$IN_PREC &gt; $INTERSECT_PREC">
      <mfenced separators="">
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                    @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="." mode="intersect">
          <xsl:with-param name="PARAM" select="$PARAM"/>
          <xsl:with-param name="PAREN" select="$PAREN"/>
          <xsl:with-param name="PAR_NO_IGNORE"
                          select="$PAR_NO_IGNORE"/>
        </xsl:apply-templates>
      </mfenced>
    </xsl:when>
    <xsl:when test="$IN_PREC &gt; $NO_PREC and $IN_PREC &lt;
                    $FUNCTION_PREC and not($SEM_SW=$SEM_ALL)
                    and not($SEM_SW=$SEM_XREF)
                    and not($SEM_SW=$SEM_XREF_EXT)">
      <xsl:apply-templates select="." mode="intersect">
        <xsl:with-param name="PARAM" select="$PARAM"/>
        <xsl:with-param name="PAREN" select="$PAREN"/>
        <xsl:with-param name="PAR_NO_IGNORE" select="$PAR_NO_IGNORE"/>
      </xsl:apply-templates>
    </xsl:when>
    <xsl:otherwise>
      <mrow>
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                    @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="." mode="intersect">
          <xsl:with-param name="PARAM" select="$PARAM"/>
          <xsl:with-param name="PAREN" select="$PAREN"/>
          <xsl:with-param name="PAR_NO_IGNORE"
                          select="$PAR_NO_IGNORE"/>
        </xsl:apply-templates>
      </mrow>
```

```
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match = "mml:apply[mml:intersect[1]]" mode="intersect">
  <xsl:param name="PARAM" select="$NO_PARAM"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:apply-templates select = "*[2]" mode="semantics">
    <xsl:with-param name="IN_PREC" select="$INTERSECT_PREC"/>
    <xsl:with-param name="PARAM" select="$PARAM"/>
    <xsl:with-param name="PAREN" select="$PAREN"/>
    <xsl:with-param name="PAR_NO_IGNORE" select="$PAR_NO_IGNORE"/>
  </xsl:apply-templates>
  <xsl:for-each select = "*[position()>2]">
    <mo> <mchar name="Intersection"/> </mo>
    <xsl:apply-templates select = "." mode="semantics">
      <xsl:with-param name="IN_PREC" select="$INTERSECT_PREC"/>
      <xsl:with-param name="PAREN" select="$PAREN"/>
      <xsl:with-param name="PAR_NO_IGNORE" select="$NO"/>
    </xsl:apply-templates>
  </xsl:for-each>
</xsl:template>

<xsl:template match = "mml:apply[mml:setdiff[1]]">
  <xsl:param name="IN_PREC" select="$NO_PREC"/>
  <xsl:param name="PARAM" select="$NO_PARAM"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:choose>
    <xsl:when test="$IN_PREC &gt; $SETDIFF_PREC or
                    $IN_PREC=$SETDIFF_PREC
                    and $PARAM=$PAR_SAME">
      <mfenced separators="">
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                    @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="." mode="setdiff">
          <xsl:with-param name="PARAM" select="$PARAM"/>
          <xsl:with-param name="PAREN" select="$PAREN"/>
          <xsl:with-param name="PAR_NO_IGNORE"
                          select="$PAR_NO_IGNORE"/>
        </xsl:apply-templates>
      </mfenced>
    </xsl:when>
    <xsl:when test="$IN_PREC &gt; $NO_PREC and $IN_PREC &lt;
                    $FUNCTION_PREC and not($SEM_SW=$SEM_ALL)
```

```
                    and not($SEM_SW=$SEM_XREF)
                    and not($SEM_SW=$SEM_XREF_EXT)">
      <xsl:apply-templates select="." mode="setdiff">
        <xsl:with-param name="PARAM" select="$PARAM"/>
        <xsl:with-param name="PAREN" select="$PAREN"/>
        <xsl:with-param name="PAR_NO_IGNORE" select="$PAR_NO_IGNORE"/>
      </xsl:apply-templates>
    </xsl:when>
    <xsl:otherwise>
      <mrow>
        <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                    @id">
          <xsl:attribute name="xref">
            <xsl:value-of select="@id"/>
          </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates select="." mode="setdiff">
          <xsl:with-param name="PARAM" select="$PARAM"/>
          <xsl:with-param name="PAREN" select="$PAREN"/>
          <xsl:with-param name="PAR_NO_IGNORE"
                          select="$PAR_NO_IGNORE"/>
        </xsl:apply-templates>
      </mrow>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match = "mml:apply[mml:setdiff[1]]" mode="setdiff">
  <xsl:param name="PARAM" select="$NO_PARAM"/>
  <xsl:param name="PAREN" select="$PAR_NO"/>
  <xsl:param name="PAR_NO_IGNORE" select="$YES"/>
  <xsl:apply-templates select = "*[2]" mode = "semantics">
    <xsl:with-param name="IN_PREC" select="$SETDIFF_PREC"/>
    <xsl:with-param name="PARAM" select="$PARAM"/>
    <xsl:with-param name="PAREN" select="$PAREN"/>
    <xsl:with-param name="PAR_NO_IGNORE" select="$PAR_NO_IGNORE"/>
  </xsl:apply-templates>
  <mo>\</mo>
  <xsl:apply-templates select = "*[3]" mode = "semantics">
    <xsl:with-param name="IN_PREC" select="$SETDIFF_PREC"/>
    <xsl:with-param name="PARAM" select="$PAR_SAME"/>
    <xsl:with-param name="PAREN" select="$PAREN"/>
    <xsl:with-param name="PAR_NO_IGNORE" select="$NO"/>
  </xsl:apply-templates>
</xsl:template>

<xsl:template match = "mml:apply[mml:card[1]]">
  <mfenced open="|" close="|" separators=",">
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                          @id">
```

```
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:for-each select = "*[position()>1]">
      <xsl:apply-templates select = "." mode="semantics"/>
    </xsl:for-each>
  </mfenced>
</xsl:template>


<!-- ***************** SEQUENCES AND SERIES ***************** -->

<xsl:template match = "mml:apply[mml:sum[1] | mml:product[1]]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                              @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:choose>
      <xsl:when test="*[2]=mml:bvar and mml:lowlimit and mml:uplimit">
        <munderover>
          <mo>
            <xsl:if test="*[1]=mml:sum">
              <mchar name="Sum"/>
            </xsl:if>
            <xsl:if test="*[1]=mml:product">
              <mchar name="Product"/>
            </xsl:if>
          </mo>
          <mrow>
            <xsl:apply-templates select = "*[2]" mode = "semantics"/>
            <mo>=</mo>
            <xsl:apply-templates select = "mml:lowlimit"
                                 mode = "semantics"/>
          </mrow>
          <xsl:apply-templates select = "mml:uplimit"
                               mode = "semantics"/>
        </munderover>
        <xsl:apply-templates select = "*[5]" mode = "semantics"/>
      </xsl:when>
      <xsl:when test="*[2]=mml:bvar and *[3]=mml:condition">
        <munder>
          <mo>
            <xsl:if test="*[1]=mml:sum">
              <mchar name="Sum"/>
            </xsl:if>
            <xsl:if test="*[1]=mml:product">
```

```
            <mchar name="Product"/>
          </xsl:if>
        </mo>
        <xsl:apply-templates select = "*[3]" mode = "semantics"/>
      </munder>
      <xsl:apply-templates select = "*[4]" mode = "semantics"/>
    </xsl:when>
  </xsl:choose>
</mrow>
</xsl:template>

<xsl:template match = "mml:apply[mml:limit[1]]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                          @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <munder>
      <mo>lim</mo>
      <mrow>
        <xsl:if test="*[2]=mml:bvar and *[3]=mml:lowlimit">
            <xsl:apply-templates select = "*[2]" mode = "semantics"/>
            <mo> <mchar name="RightArrow"/> </mo>
            <xsl:apply-templates select = "*[3]" mode = "semantics"/>
        </xsl:if>
        <xsl:if test="*[2]=mml:bvar and *[3]=mml:condition">
          <xsl:apply-templates select = "*[3]" mode = "semantics"/>
        </xsl:if>
      </mrow>
    </munder>
    <xsl:apply-templates select = "*[4]" mode = "semantics"/>
  </mrow>
</xsl:template>


<!-- ***************** TRIGONOMETRY ***************** -->

<xsl:template match="mml:apply[*[1][self::mml:sin | self::mml:cos |
                     self::mml:tan | self::mml:sec | self::mml:csc |
                     self::mml:cot | self::mml:sinh | self::mml:cosh |
                     self::mml:tanh | self::mml:sech | self::mml:csch |
                     self::mml:coth | self::mml:arcsin |
                     self::mml:arccos | self::mml:arctan]]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                          @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
```

```
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="not(parent::mml:apply[mml:power[1]])">
      <xsl:apply-templates select = "*[1]" mode = "trigonometry"/>
    </xsl:if>
    <xsl:if test="parent::mml:apply[mml:power[1]]">
      <msup>
        <xsl:apply-templates select = "*[1]" mode = "trigonometry"/>
        <xsl:apply-templates select = "../*[3]" mode = "semantics"/>
      </msup>
    </xsl:if>
    <mo> <mchar name='ApplyFunction'/> </mo>
    <xsl:apply-templates select = "*[2]" mode = "semantics">
      <xsl:with-param name="IN_PREC" select="$FUNCTION_PREC"/>
      <xsl:with-param name="PAR_NO_IGNORE" select="$NO"/>
    </xsl:apply-templates>
  </mrow>
</xsl:template>

<xsl:template match = "*" mode="trigonometry">
  <mo>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                              @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:choose>
      <xsl:when test="self::mml:sin">
        <xsl:value-of select="'sin'"/>
      </xsl:when>
      <xsl:when test="self::mml:cos">
        <xsl:value-of select="'cos'"/>
      </xsl:when>
      <xsl:when test="self::mml:tan">
        <xsl:value-of select="'tan'"/>
      </xsl:when>
      <xsl:when test="self::mml:sec">
        <xsl:value-of select="'sec'"/>
      </xsl:when>
      <xsl:when test="self::mml:csc">
        <xsl:value-of select="'csc'"/>
      </xsl:when>
      <xsl:when test="self::mml:cot">
        <xsl:value-of select="'cot'"/>
      </xsl:when>
      <xsl:when test="self::mml:sinh">
        <xsl:value-of select="'sinh'"/>
      </xsl:when>
      <xsl:when test="self::mml:cosh">
```

```
        <xsl:value-of select="'cosh'"/>
      </xsl:when>
      <xsl:when test="self::mml:tanh">
        <xsl:value-of select="'tanh'"/>
      </xsl:when>
      <xsl:when test="self::mml:sech">
        <xsl:value-of select="'sech'"/>
      </xsl:when>
      <xsl:when test="self::mml:csch">
        <xsl:value-of select="'csch'"/>
      </xsl:when>
      <xsl:when test="self::mml:coth">
        <xsl:value-of select="'coth'"/>
      </xsl:when>
      <xsl:when test="self::mml:arcsin">
        <xsl:value-of select="'arcsin'"/>
      </xsl:when>
      <xsl:when test="self::mml:arccos">
        <xsl:value-of select="'arccos'"/>
      </xsl:when>
      <xsl:when test="self::mml:arctan">
        <xsl:value-of select="'arctan'"/>
      </xsl:when>
    </xsl:choose>
  </mo>
</xsl:template>


<!-- ***************** STATISTICS ***************** -->

<xsl:template match = "mml:apply[mml:mean[1]]">
  <mfenced open="&lt;" close="&gt;" separators=",">
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                          @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:for-each select = "*[position()>1]">
      <xsl:apply-templates select = "." mode="semantics"/>
    </xsl:for-each>
  </mfenced>
</xsl:template>

<xsl:template match = "mml:apply[mml:sdev[1]]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                          @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
```

```
      </xsl:attribute>
    </xsl:if>
    <mo> <mchar name="sigma"/> </mo>
    <mfenced separators=",">
      <xsl:for-each select = "*[position()>1]">
        <xsl:apply-templates select = "." mode="semantics"/>
      </xsl:for-each>
    </mfenced>
  </mrow>
</xsl:template>

<xsl:template match = "mml:apply[mml:variance[1]]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <mo> <mchar name="sigma"/> </mo>
    <msup>
      <mfenced separators=",">
        <xsl:for-each select = "*[position()>1]">
          <xsl:apply-templates select = "." mode="semantics"/>
        </xsl:for-each>
      </mfenced>
      <mn>2</mn>
    </msup>
  </mrow>
</xsl:template>

<xsl:template match = "mml:apply[mml:median[1]]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <mo>median</mo>
    <mfenced separators=",">
      <xsl:for-each select = "*[position()>1]">
        <xsl:apply-templates select = "." mode="semantics"/>
      </xsl:for-each>
    </mfenced>
  </mrow>
</xsl:template>

<xsl:template match = "mml:apply[mml:mode[1]]">
  <mrow>
```

```
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                              @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <mo>mode</mo>
    <mfenced separators=",">
      <xsl:for-each select = "*[position()>1]">
        <xsl:apply-templates select = "." mode="semantics"/>
      </xsl:for-each>
    </mfenced>
  </mrow>
</xsl:template>

<xsl:template match = "mml:apply[mml:moment[1]]">
  <mfenced open="&lt;" close="&gt;" separators="">
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                              @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="*[2]=mml:degree">
      <msup>
        <xsl:apply-templates select="*[3]" mode = "semantics"/>
        <xsl:apply-templates select="*[2]" mode = "semantics"/>
      </msup>
    </xsl:if>
    <xsl:if test="not(*[2]=mml:degree)">
      <xsl:for-each select = "*[position()>1]">
        <xsl:apply-templates select = "." mode="semantics"/>
      </xsl:for-each>
    </xsl:if>
  </mfenced>
</xsl:template>


<!-- ***************** LINEAR ALGEBRA ***************** -->

<xsl:template match="mml:vector">
  <mfenced separators="">
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                              @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <mtable>
      <xsl:for-each select="*">
```

```
        <mtd>
          <xsl:apply-templates select="." mode = "semantics"/>
        </mtd>
      </xsl:for-each>
    </mtable>
  </mfenced>
</xsl:template>

<xsl:template match = "mml:matrix">
  <mfenced separators="">
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <mtable>
      <xsl:apply-templates select="*" mode = "semantics"/>
    </mtable>
  </mfenced>
</xsl:template>

<xsl:template match = "mml:matrixrow">
  <mtr>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:for-each select="*">
      <mtd>
        <xsl:apply-templates select="." mode = "semantics"/>
      </mtd>
    </xsl:for-each>
  </mtr>
</xsl:template>

<xsl:template match = "mml:apply[mml:determinant[1]]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                                @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <mo>det</mo>
    <xsl:apply-templates select = "*[2]" mode = "semantics"/>
  </mrow>
</xsl:template>
```

```
<xsl:template match = "mml:apply[mml:transpose[1]]">
  <msup>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                              @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:apply-templates select = "*[2]" mode = "semantics"/>
    <mo>T</mo>
  </msup>
</xsl:template>

<xsl:template match = "mml:apply[mml:selector[1]]">
  <xsl:if test="*[2]=mml:matrix and *[3]=mml:cn">
      <xsl:variable name="m" select = "*[3]"/>
      <xsl:choose>
        <xsl:when test="*[4]=mml:cn">
          <xsl:variable name="n" select = "*[4]"/>
          <xsl:copy-of select=
                    "mml:matrix/*[position()=$m]/*[position()=$n]"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:copy-of select = "mml:matrix/*[position()=$m]"/>
        </xsl:otherwise>
      </xsl:choose>
  </xsl:if>
  <xsl:if test="(*[2]=mml:vector or *[2]=mml:list) and *[3]=mml:cn">
    <xsl:variable name="m" select = "*[3]"/>
    <xsl:copy-of select = "*[2]/*[position()=$m]"/>
  </xsl:if>
</xsl:template>

<xsl:template match = "mml:apply[mml:vectorproduct[1] |
                                 mml:scalarproduct[1] |
                                 mml:outerproduct[1]]">
  <mrow>
    <xsl:if test="($SEM_SW=$SEM_XREF or $SEM_SW=$SEM_XREF_EXT) and
                                                              @id">
      <xsl:attribute name="xref">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:apply-templates select="*[2]" mode = "semantics"/>
    <mo>
      <xsl:if test="mml:vectorproduct[1]">
        <mchar name="Cross"/>
      </xsl:if>
      <xsl:if test="mml:scalarproduct[1] | mml:outerproduct[1]">
```

```
        <xsl:value-of select="'.'"/>
      </xsl:if>
    </mo>
    <xsl:apply-templates select="*[3]" mode = "semantics"/>
  </mrow>
</xsl:template>

</xsl:stylesheet>
```

Appendix B

The ID Generator for MathML Documents

```
<?xml version="1.0"?>

<!-- ******************************************************* -->
<!--           ID Generator for MathML Documents             -->
<!--    By Igor Rodionov, Computer Science Department of     -->
<!--   the University of Western Ontario, London, Canada     -->
<!--             Version 0.11 from Jan. 03, 2001             -->
<!--              Comments to:  igor@csd.uwo.ca              -->
<!-- ******************************************************* -->

<xsl:stylesheet id="http://www.scl.csd.uwo.ca/mmlidgen.xsl"
                version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:mml="http://www.w3.org/1998/Math/MathML"
                xmlns="http://www.w3.org/1998/Math/MathML">

<xsl:output method="xml" indent="yes"/>

<xsl:strip-space elements="apply semantics annotation-xml
        csymbol fn cn ci interval matrix matrixrow vector
        lambda bvar condition logbase degree set list
        lowlimit uplimit"/>


<!-- ************* THE TOPMOST ELEMENT: MATH ************** -->

<xsl:template match = "mml:math">
  <xsl:copy>
    <xsl:copy-of select="@*"/>
    <xsl:apply-templates mode="idgen"/>
  </xsl:copy>
</xsl:template>

<!-- ******************** ID GENERATOR ******************** -->

<xsl:template match = "mml:*" mode="idgen">
  <xsl:copy>
    <xsl:copy-of select="@*"/>
```

```
    <xsl:attribute name="id">
      <xsl:value-of select="generate-id()"/>
    </xsl:attribute>
    <xsl:apply-templates mode="idgen"/>
  </xsl:copy>
</xsl:template>

<xsl:template match = "text()" mode="idgen">
  <xsl:copy-of select="."/>
</xsl:template>

</xsl:stylesheet>
```

VITA

Name:	Igor V. Rodionov
Place of birth:	Moscow, Russia
Year of birth:	1971
Post-secondary Education and Degrees	The University of Western Ontario London, Canada 1999-2000 M.Sc.
	Vanderbilt University Nashville, TN, USA 1995-1997 M.A.
	Moscow State Technical University Moscow, Russia 1988-1994 B.Sc.
Honors and Awards:	Freedom Support Act Scholarship 1995-1997
Related work experience:	Teaching Assistant The University of Western Ontario 1998-2000
	Programmer Book Trading Firm 'Bibliopolis' Moscow, Russia 1994-1995