XML
and the
Communication of Mathematical Objects

by

Xuehong Li

Graduate Program in Computer Science

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
April 1999

© Xuehong Li 1999

Abstract

Software programs of various sorts must exchange mathematical formulas and objects as data. This thesis examines the emerging standards for this type of exchange, including MathML and OpenMath. Both of these standards are based on the Extensible Markup Language (XML) and address different aspects of this data exchange problem. One of the most significant gaps in these Mathematical Markup Languages is the lack of a macro mechanism to handle abbreviations and to abstract new concepts. In this thesis, we examine the suitability of XSL, a stylesheet language for XML, as a means of macro extension in these mathematical markup domains.

We have implemented an XSL processor and several trial stylesheets to convert MathML *content* markup to MathML *presentation* markup annotated with OpenMath *semantics*. This models the general process of extending the content tags of MathML via macros relying on OpenMath. On the basis of these experiments, we conclude that XSL draft recommendation of December 16, 1998, with minor extensions detailed here, could be a suitable basis for macro processing in MathML.

Key words: XML, MathML, OpenMath, XSL, communication of mathematical objects, macro mechanism.

Acknowledgements

I would like to thank my supervisor Dr. Stephen Watt for his encouragement and endless enthusiasm not only for this thesis but also all research activities, for his invaluable and enlightening advice, and for his relentless help.

Thanks as well to faculty, staff and fellow graduate students in this department for all the constructive discussions and challenging questions as well as their encouragement and help, especially to the friends of the Symbolic Computation Lab for their endless help.

Finally I would also like to thank my family, especially my wife, for her love and support.

Without the help of these people, completing this thesis would not have been possible.

Contents

Certificate of Examination ii

Abstract iii

Acknowledgements iv

Contents v

Nomenclature & Acronyms viii

1 Introduction 1

2 An Overview of XML 3
- 2.1 What Is XML? 3
  - 2.1.1 Logical Structure and Physical Structure 4
  - 2.1.2 DTD, Valid Document and Well-formed Document 5
- 2.2 The Origins of XML 8
- 2.3 The Applications of XML 10
  - 2.3.1 Data Exchange Applications 10
  - 2.3.2 Document Publishing Applications 10

3 OpenMath 12
- 3.1 What Is OpenMath? 12
- 3.2 Why OpenMath? 12
- 3.3 How Was OpenMath Developed? 13
  - 3.3.1 The History of OpenMath 13
  - 3.3.2 Design Goals of OpenMath 14
  - 3.3.3 Mathematical Object Representations in OpenMath 16

4 MathML 24
- 4.1 What is MathML? . . . 24
- 4.2 Why MathML? . . . 24
- 4.3 How Does MathML Work? . . . 26
  - 4.3.1 Mathematical Notations and Content . . . 26
  - 4.3.2 MathML Presentation Elements . . . 27
  - 4.3.3 MathML Content Elements . . . 32
  - 4.3.4 The Top-level Interface Element . . . 36

5 The Relationship Between MathML and OpenMath 38
- 5.1 General Concepts . . . 38
- 5.2 Using OpenMath Annotation in MathML . . . 38
- 5.3 Using MathML Annotation in OpenMath . . . 41

6 An Overview of XSL 43
- 6.1 The Structure of A Style Sheet . . . 44
- 6.2 How to Form a Template? . . . 45
- 6.3 What Are the Patterns in XSL? . . . 46
- 6.4 Examples . . . 49

7 A Macro Mechanism for MathML 56
- 7.1 Macros for Abstracting Different Notational Styles . . . 57
- 7.2 Macros for Expanding Abbreviations . . . 58
- 7.3 Macros for Combined Presentation-Semantics Markup . . . 60
- 7.4 Realization of the Macro Mechanism . . . 62
- 7.5 Realization of Macros for Abstracting Different Notational Style . . . 64
- 7.6 Realization of Macros for Expanding Abbreviations . . . 64
- 7.7 Realization of the Macros for Combined Markup . . . 66
- 7.8 Remarks in Writing Stylesheets . . . 70
- 7.9 Extension of XSL . . . 73
- 7.10 More Examples . . . 76
- 7.11 Future Work . . . 82

8 Conclusion 89

Appendix 90

A Stylesheets for OpenMath CDs **90**
A.1 Limit.xsl 90
A.2 Sumprod.xsl 93
A.3 Calculus.xsl 96
A.4 Transc.xsl 100
A.5 Quant.xsl 102
A.6 Interval.xsl 103
A.7 Arith.xsl 105
A.8 List.xsl 106
A.9 Logic.xsl 107
A.10 Integer.xsl 108
A.11 Set.xsl 108
A.12 Relation.xsl 110
A.13 Stat.xsl 113
A.14 Fns.xsl 114
A.15 La-mml.xsl 114
A.16 A Stylesheet for Common Elements 115

Bibliography **117**

Nomenclature & Acronyms

XML	Extensible Markup Language
DTD	Document Type Definition
SGML	Standard General Markup Language
MathML	Mathematical Markup Language
OpenMath	A platform-independent standard for the representation of semantically-rich mathematical objects so that they may be exchanged in a meaningful way between various software tools.
OM	OpenMath
CD	Content Dictionary
XSL	Extensible Stylesheet Language

Chapter 1

Introduction

The need for communication of mathematical objects has appeared in several communities including the computer algebra field, mathematics, teaching and commercial publishing.

Mathematicians want to make their research works available on the World-Wide Web in an efficient way. Similarly, students and teachers may want to place scientific curriculum materials on the Web. Furthermore, commercial publishers are also involved with mathematics on the Web at all levels — from electronic versions of print books, to interactive textbooks, to academic journals. For example, the reader of a digital mathematical document may want to transfer the content from the document into a mathematical program such as Maple or Mathematica to further investigate examples. Users of computer algebra systems may want their examples to run in several computer algebra systems, and to make use of the advantages of each system. An underlying issue in all these examples is the need for communication of mathematical objects.

How can we communicate mathematical objects? Or, more accurately, how can we *effectively* communicate mathematical objects? Mathematical objects generally have a complex structure with two independent aspects: the *presentation notation* and the *semantic content*. Dealing with both of these equally or emphasizing one of them so that special purposes are satisfied leads to different designs. Much work has been done in this area in recent years:

The OpenMath Society has designed OpenMath [1], a platform-independent standard for the representation of semantically-rich mathematical objects so that they may be exchanged in a meaningful way between various software tools. The core of this standard is to supply a collection of CDs (Content Dictionaries) and data exchange formats. A CD defines a set of related mathematical concepts and operations

as a set of symbols, each symbol has a name and is attached some information. One of the data transfer encoding in OpenMath is XML (Extensible Markup Language) [2].

The W3C (World Wide Web Consortium) Math Working Group has released a formal recommendation of MathML (Mathematical Markup Language) [3]. This is an application of XML intended to facilitate the use of mathematical and scientific content on the Web. MathML supplies *presentation* elements and *content* elements, aiming to capture both the mathematical notation and the meaning.

These two standards' emphases are different: OpenMath is primarily for semantics and MathML is primarily for presentation. They address different aspects of the problem of representing mathematical objects and transmitting.

One of the most significant gaps in these two Mathematical Markup Languages is the lack of a macro mechanism to handle abbreviation and to abstract new concepts. Therefore, it is necessary to develop such a macro mechanism. This macro mechanism could make the communication of mathematical objects more effective.

XSL [4],[5], a stylesheet language for XML, is designed to construct trees of formatting objects for display. It can also be used for general XML transformation. We have implemented an XSL processor and an XML parser. Using these tools, we have experimented with many stylesheets to examine and experiment with XSL as a means of the macro extension. We find that XSL draft recommendation of Dec. 16, 1998 [5], with minor extensions, could be a suitable basis for mathematical macro processing.

In this thesis, we first give an overview of XML, and then we systematically describe the ideas and methods of OpenMath and MathML in dealing with the communication of mathematical objects. Then we discuss the relationship between them. Finally, and most importantly, after introducing XSL, we discuss our development of a macro mechanism for MathML, from presenting the problems to the implementation of a solution.

Chapter 2

An Overview of XML

2.1 What Is XML?

XML, Extensible Markup Language, is a publishing and document interchange format developed by the World Wide Web Consortium(W3C) [2]. Since it was released on 10 February 1998, it has gained widespread acceptance.

XML is a data format for storing structured and semi-structured text intended for communication or publication in a variety of media. An XML document contains special instructions, called tags, which usually enclose identifiable parts of the document. The sample below is a fragment of an XML document:

```
<COURSE>
  <TITLE>Software Engineering</TITLE>
  <ID>CS307</ID>
  <CREDIT>3</CREDIT>
</COURSE>
```

As a simplification of SGML (Standard General Markup Language), the XML format is similar to HTML (Hypertext Markup Language). The difference, however, is that HTML has a fixed set of tags, whereas XML allows the creation and utilization of user-defined tags. This gives individuals the ability to describe the information in a document with meaningful tags. This extensibility of self-description makes XML suitable for not only the world of publishing but also for describing complex data structures.

XML is a simplified subset of SGML that inherits SGML's three main features, which are that it is:

1. Generalized and descriptive;
2. Extensible;

3. Validated.

These features make XML quite flexible and powerful. Its simplicity compared to SGML makes XML easier to learn and process. We will discuss the relationship of XML, HTML and SGML in more detail in Section 2.

To get a basic understanding of XML quickly, we quote the design goals for XML here from the recommendation [2]:

"The design goals for XML are:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall be supporting a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features is to be kept to the absolute minimum, ideally zero.
6. XML documents shall be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance."

We note that a consequence of these goals is that XML markup tends to be rather verbose, but it compresses very efficiently.

2.1.1 Logical Structure and Physical Structure

An XML-based document has both a logical and a physical structure.

The logical structure means that a document can be broken down into named units and sub-units, called *elements.* An element can contain other elements and it can also contain data, e.g. the words and sentences that are the text of a document. For example, a book is broken down into chapters, each chapter is an element which may contain further elements such as paragraph, table and image.

In other words, the logical structure is actually a tree structure of a document. The root is the element that contains all of the others (e.g. book). The other elements are either branches or leaves: the branches are the elements that do contain other elements; the leaves are the elements that do not contain other elements.

Elements can also have extra information attached to them called *attributes*. Attributes describe properties of elements. For example, we can store the condition of a product as an attribute in a <PRODUCT> tag:

```
<PRODUCT CONDITION = "Excellent">
```

The physical structure means that components of the document, which are called *entities*, can be named and stored separately, sometimes in other data files so that information can be re-used and non-XML data (such as image data) can be included by reference. One may insert an reference somewhere in a document to make use of an "entity". The processor replaces the reference with the entity itself, which is called the replacement of text. For example, in a book on XML the name "Extensible Markup Language" may appear often. An entity, perhaps named "XML", may be created to hold this text. This entity would then be referred to using the notation "&XML;"

Generally, we may wish to use an entity in the following cases:

1. The same information is used in several places, and duplication would be both error-prone and time-consuming.

2. The information may be represented differently by an incompatible system.

3. The information is part of a large document that requires splitting into manageable units.

4. The information involves data that conforms to a format other than XML.

2.1.2 DTD, Valid Document and Well-formed Document

As we mentioned above, a structured document is composed of a tree of elements. Each element can contain other elements or data. Taking a book as an example: the entire document (book) is a single element containing smaller elements (chapters) and they contain even smaller ones, until the individual leaves (book's text).

Obviously, those elements cannot be put in the tree randomly: chapters cannot contain books and footnotes can not contain chapters. Determining whether or not

a document is a valid book requires rules stating which elements can be contained in other element. Those rules form the Book type. The elements and the rules restricting them define the document type. So if two documents have very different elements or allow elements to be combined in very different ways, then they may probably do not conform to the same document type.

Formally, a document type definition (or DTD) is a series of definitions for element types, attributes, entities and notations. It declares which of these are legal within the document and in what places they are legal. A document can claim to conform to a particular DTD in its document type declaration.

Here is a DTD for a document of student registration records:

```
<?XML version = "1.0" ?>
<!DOCTYPE DOCUMENT  [
<!ELEMENT DOCUMENT  (STUDENT)*>
<!ELEMENT STUDENT   (NAME,ID,REGISTRATION)>
<!ELEMENT NAME  (LASTNAME,FIRSTNAME)>
<!ELEMENT LASTNAME  (#PCDATA)>
<!ELEMENT FIRSTNAME (#PCDATA)>
<!ELEMENT ID  (#PCDATA)>
<!ELEMENT REGISTRATION  (COURSE)*>
<!ELEMENT COURSE (TITLE,NUMBER,CREDIT)>
<!ELEMENT TITLE  (#PCDATA)>
<!ELEMENT NUMBER (#PCDATA)>
<!ELEMENT CREDIT (#PCDATA)>
]>
```

where `#PCDATA` m means character data.

Having a DTD in a document is optional. For a small document, one may just remember the simple DTD and write the document directly according to the DTD. For a large, complex document, however, it is very difficult to concentrate on every required rule, and therefore a DTD becomes crucial since the computer (validated parser) can check the rule requirement automatically. It is worthwhile to indicate that, by applying the same reasoning, some XML documents do not have a document type declaration. That does not mean that they do not conform to a document type — it just means that they do not claim to conform to some formally defined document type definition.

The XML recommendation describes two levels of conformance: *valid* and *well-formed*. A valid XML document must have a DTD and adhere to it. A valid XML document must also be well-formed. A well-formed document can be used without

a DTD, but it must follow some simple rules to enable a browser to parse the file correctly.

These simple rules are specified in Section 2.1 of the XML recommendation[1]. The most important one among them is that an element cannot be partially enclosed by another element. For example, a section may not belong to two chapters. So, roughly speaking, any XML document that consists of properly nested elements is called a well-formed document.

As an example, the following XML document is both valid and well-formed:

```
<?XML version = "1.0" ?>
<!DOCTYPE DOCUMENT  [
<!ELEMENT DOCUMENT  (STUDENT)*>
<!ELEMENT STUDENT  (NAME,ID,REGISTRATION)>
<!ELEMENT NAME  (LASTNAME,FIRSTNAME)>
<!ELEMENT LASTNAME  (#PCDATA)>
<!ELEMENT FIRSTNAME  (#PCDATA)>
<!ELEMENT ID  (#PCDATA)>
<!ELEMENT REGISTRATION  (COURSE)*>
<!ELEMENT COURSE  (TITLE,NUMBER,CREDIT)>
<!ELEMENT TITLE  (#PCDATA)>
<!ELEMENT NUMBER (#PCDATA)>
<!ELEMENT CREDIT  (#PCDATA)>
]>
<DOCUMENT>
  <STUDENT>
    <NAME>
      <LASTNAME>Edwards</LASTNAME>
      <FIRSTNAME>Britta</FIRSTNAME>
    </NAME>
    <ID>3171998</ID>
    <REGISTRATION>
      <COURSE>
        <TITLE>Software Engineering</TITLE>
        <NUMBER>CS307</NUMBER>
        <CREDIT>3</CREDIT>
      </COURSE>
      <COURSE>
        <TITLE>Computer Network</TITLE>
        <NUMBER>cs357</NUMBER>
```

```
        <CREDIT>3</CREDIT>
      <COURSE>
    </REGISTRATION>
  </STUDENT>
</DOCUMENT>
```

Next we show a document that is well-formed but not valid (no DTD):

```
<?XML version = "1.0"?>
<greeting>Hello, world!</greeting>
```

Finally, here is a document that contains a nesting error and so is neither well formed nor valid:

```
<?XML version = "1.0" ?>
<greeting>
 Hello, world!
<farewell>
</greeting>
 Bye, world!
</farewell>
```

2.2 The Origins of XML

After introducing the basic concepts of XML in the first section, now we will discuss the relationship of XML with other two existing languages SGML (Standard General Markup Language) and HTML (Hypertext Markup Language). Simply speaking, XML is primarily based on SGML, but inherits some characteristics from HTML and so contains additional features that are aimed at its use on the Internet.

The General Markup concept appeared in early 1960s. When IBM asked Charles Goldfarb to build a system for storing, finding, managing, and publishing legal documents, Goldfarb found that many systems in IBM could not talk to each other and the main problem was that they had different representations for the information. In the late sixties, Goldfarb had a team to solve this problem. They realized three key issues to be addressed: first, support a common representation; second, the representation should be specific to legal documents; third, markup is rule-based. In addressing the first issue, they got the idea to abstract the formatted rendition to a "General Markup" for an alternative to either the formatting markup or WYSIWYG (What

You See Is What You Get). The idea of General Markup is to allow the authors to say what they mean not just what they look like. When the format is needed, a style sheet can be attached to the document to tell the computer how to generate a formatted rendition from the General Markup. The second issue "specific to" derived the concept of "extensibility". That is, it can be extensible to specific problem domain to markup different documents. Third issue is validation according to the DTD. The General Markup Language evolved nearly for two decades, until 1986, when the Standard General Markup Language (SGML) was set by ISO. However, with many other features, SGML is too big and complex, or perhaps too advanced, to become fully utilized and widespread.

HTML was invented by Tim Berners-Lee about 1990. Along with URLs and HTTP, HTML allowed CERN scientists to write physics papers and link them together, and today it has evolved as the most diverse, popular hypertext information system in existence. Its simplicity is widely believed to be an important part of its success. The simplicity and the other Web specifications allowed programmers around the world to quickly build systems and tools to work with the Web.

HTML inherited some important strength from SGML. Most of its element types were generalized and descriptive, not formatting constructs like Tex. HTML document uses bracketing in the same way as SGML. However, HTML only uses a fixed set of element types, which makes it simple and easy to learn, but the cost is, on the other hand, that its application is severely limited. This limitation manifests itself mainly in three respects, as pointed out by Jon Bosak in [6]:

> 1) Extensibility: HTML dos not allows users to specify their own tags or attributes in order to parameterize or otherwise semantically qualify their data.
>
> 2) Structure: HTML does not support the specification of deep structures needed to represent database schemas or objected-oriented hierarchies.
>
> 3) Validation. HTML does not support the kind of language specification that allows consuming applications to check data for structural validation on importation.

As the World Wide Web went worldwide and was used by more people, more and more different flavors of documents began to appear on the Web. Some of these had deep complex data structure; some may be a long, needing a check of the requirement for validation. Therefore, HTML, with the above stated limitations, cannot serve for all these document types. The process of extending HTML tags may save this situation

partially but cannot completely solve this problem, as new applications may force this extension in an endless process.

XML is intended to rescue this problem. Instead of having a fixed set of tags, XML inherits all three important features of SGML: Generalized and Descriptive element type, Self-description and Validation, and more importantly, Extensibility. By removing many of the more complex and less-used features of SGML, XML is a simplified subset of SGML. The simplicity makes XML's tool implementation easier than SGML's and the three good features give XML more power than HTML.

With XML, you can write any flavor of document with your own descriptive element type and attribute name. If printing or publishing is desired, you can design a style sheet and attach to the document and tell the computer whatever format you want.

With XML, you can describe any deep level and/or nested structure of data.

With XML, you could give an optional structure grammar of the document and check that the document conforms to the structural requirements.

It is these beneficial features together with the simplicity that makes XML effective, powerful and increasingly popular.

2.3 The Applications of XML

2.3.1 Data Exchange Applications

When complex data must be exchanged between two programs, XML is a suitable format for making the data self-describing. XML has certain potential in a number of already identified domains, such as EDI (Electronic Data Interchange) and general meta-data (XML-Data and RDF (Resource Description Framework)). Also, in this context, OpenMath can be viewed as such an application. XML is used to encode OpenMath CDs and OpenMath objects so that they may be exchanged between various software tools. We will address this in detail in Chapter 3.

2.3.2 Document Publishing Applications

XML is a data language for markup of semi-structured documents. The examples of such semi-structured documents include reference works, training guides, technical manuals, academic journals and reports. XML tags can represent all the features of a typical document and data stream.

Markup with XML can result in a long-term cost benefit. Once the core data is held in a controlled and structured format, a high level of automation can be achievable; in some cases, even allowing fully automated publication, in appropriate formats, to paper, CDROM and the Internet. When published on the Internet, different formats are possible by attaching different style sheets.

Some areas, like mathematics and chemistry, have more complexly structured data and documents. To publish such documents on the Web definitely needs more work, and specific markup languages based on XML are perhaps needed for these domains. MathML is an application of XML to be used in publishing mathematical documents on the Web. In Chapter 4, we will address this work in detail.

More and more applications shall appear in the future. As far as XML acceptance is concerned, Microsoft has embraced XML for future releases of Internet Explorer, and Netscape is considering doing the same for its browser. XML will most likely gain widespread acceptance as the power of XML-enabled applications is realized.

According to Jon Bosak of the XML Working Group [6], the applications that will drive the acceptance of XML can be divided into four broad categories:

1. Applications that require the Web client to mediate between two or more heterogeneous databases.
2. Applications that attempt to distribute a significant proportion of the processing load from the Web server to the Web client.
3. Applications that require the Web client to present different views of the dame data to different users.
4. Applications in which intelligent Web agents attempt to tailor information discovery to the needs of individual users.

Chapter 3

OpenMath

3.1 What Is OpenMath?

OpenMath defines a platform-independent standard for the representation of mathematical objects so that they may be communicated in a meaningful way, between various software tools, through various media [1]. Some examples of such tools are general purpose or specialized computer algebra systems, document preparation systems, Web browsers displaying formulas, equation editors, and databases containing mathematical information. Some examples of the communication media are regular files, electronic mail, cut and paste, internet sockets, and CORBA applications.

3.2 Why OpenMath?

With the increase of the number of programs that do symbolic or numeric computation, and programs that display or typeset mathematical information, the need for a standard computer representation for communicating mathematical objects has become apparent. No one computer application does everything and each one has its own virtues and deficiencies. For example, many general purpose computer algebra systems implement algorithms, limiting the complexity of the problems they can handle, or do not have the data types available to implement an algorithm in an efficient way. Special purpose systems, on the other hand, often implement selected algorithms in an efficient way but are restricted to a small mathematical field. Naturally, when doing some mathematical work, one may want to use a specific application for one particular task and use another application for other tasks. Thus, switching the job between the two applications and sharing the information becomes necessary. If information can be shared between two applications, the user can optimize

the strengths of both. Moreover, being able to share data between two applications supports modular problem solving by having specific applications performing specific subtasks. This eliminates the need to re-implement existing techniques and allows for integration of already existing software components. However, since each application may have different internal data structures, without a standard computer representation for communicating mathematical objects, we may need to write over 100 different protocols to make 15 different mathematics programs talk to each other. If with a standard, this situation improves to only require 15 programs to translate between their internal representation of mathematical data and the common representation of the corresponding mathematical data, the advantage becomes apparent. A possible question may be raised: "Why not use Tex as the standard?" The answer is that Tex is only for formatting and does not carry semantic meaning for mathematical objects. Since no existing program can do the job discussed above, a new design for a standard becomes necessary.

3.3 How Was OpenMath Developed?

3.3.1 The History of OpenMath

The origin of OpenMath dates to 1992, when Maple developers began to discuss how best to organize communication between mathematical software packages. At that time, Maple developers had already obtained some experience from fashioning bridges between Maple and Matlab, and between Maple and Mathcad. When they had further need to also communicate with other mathematical software packages, they felt that the mechanism of continuously building on special, proprietary software packages was not a good approach. This problem was also encountered by developers of various other computer algebra systems, and therefore it was realized that an open standard for communication of mathematical objects was needed. To this end, Professor Gaston Gonnet organized a workshop at the ETH Zurich in December 1993, seeking the participation of potentially interested members of the computer algebra community. He proposed defining a standard for the communication of mathematical objects: the OpenMath protocol. Since then many individuals have contributed their time and effort to this goal of achieving an open standard. The workshops continue to be held frequently, once every six months. Meanwhile, a group of European researchers has launched a collaborat on the design and implementation of OpenMath standards and tools. This group, the OpenMath consortium, intends that OpenMath become an international *de-facto* industrial standard.

Meanwhile, the North American OpenMath Initiative was lunched as a North American counterpart to the European group. The direction of OpenMath effort has been guided by OpenMath Steering Committee (renamed as Board of the OpenMath Society in 1998), consisting of A. Cohen, G. Gonnet, M. Seppala, R. Sutor and S. Watt.

Much work has been done in the past several years. The first proposal of the standard [7] has evolved into the first official version of OpenMath, released on December 15, 1996 [1]. After that, S. Dalmas, M. Gaetano and S. Watt presented the first implementation of the standard in the form of a C library [8], which has been embedded in experimental versions of two major computer algebra systems, Maple and Reduce, so that they can communicate each other. This demonstrates a proof of concept of OpenMath. Similar activities appeared in other forms, e.g. an API for Aldor to allow Aldor to communicate with other computer algebra systems [8], and a Java library for OpenMath created by the PolyMath group at Simon Fraser University [9].

3.3.2 Design Goals of OpenMath

The general goal of OpenMath is to define a platform-independent standard for the representation of mathematical objects so that they may be exchanged in a meaningful way between various software tools.

To effectively achieve this general goal, the OpenMath Steering Committee sets the following objectives:

• OpenMath must preserve all important or costly information during transmission. Here "important" means that the information is indispensable for correct interpretation and "costly" means expensive in terms of computation costs.

• OpenMath must be suitable for transmitting data from many different areas of mathematics. Ideally, any mathematical object that can be represented in a computer can be transmitted using OpenMath.

• OpenMath must be able to handle floating point numbers and other data. Examples of the latter are mathematical formulae, a variety of finite discrete mathematical objects, formal proofs, algorithms, and integers of unlimited size. In addition, it must be able to transmit large amounts of machine precision numerical data reasonably efficiently.

• The ability to extend the standardized core of OpenMath is vital since new areas of mathematics will continually be implemented and applied. Moreover, the mechanism for extending OpenMath should be simple and well documented. Extensions

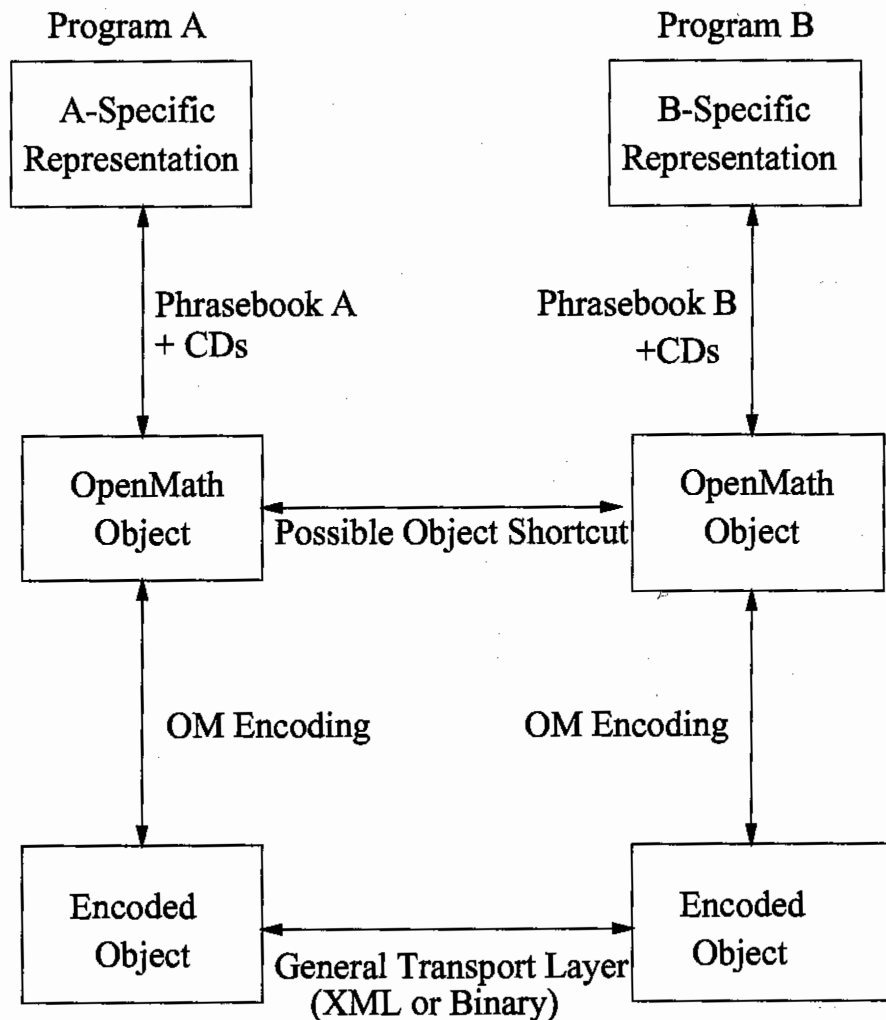
must be able to be produced by reasonably competent people outside the group of original OpenMath implementers. Efficiency is important but must be tempered with concerns for flexibility and extensibility.

• One must be able to use OpenMath to transmit data using at least these forms of transmission: email, cut-and-paste, local and non-local interprocess links (for example, UNIX sockets, OLE Automation, OLE Data Transfer), and saving to and retrieving from files.

• OpenMath must permit fairly straight-forward implementation of compliant senders and receivers, so that authors of mathematical (or other) packages can easily supply their own OpenMath interfaces.

3.3.3 Mathematical Object Representations in OpenMath

The architecture of OpenMath can be described in the following figure:



where OM means OpenMath and CD means Content Dictionary.

From the picture, we can see that there are three layers of representations for

a mathematical object. The first layer is a private layer that is the internal representation used by an application. The second layer is an abstract layer that is the representation as an OpenMath object. The third layer is a communication layer that translates the OpenMath object representation to a stream of bytes that is used as the external form (the low-level form that is actually exchanged). The hierarchy of these layers corresponds to the way OpenMath is normally integrated into an application. The application dependent program manipulates the mathematical objects using its internal representation. It can convert them to OpenMath objects and communicate them by using XML encoding of OpenMath objects.

The inverse process of the above conversions can be done after transmission.

The second layer is called the *abstract layer* because the OpenMath object can be considered abstract: it is not necessary for an OpenMath-compliant application to actually build the OpenMath object: it is sufficient to create a valid encoding from an internal form or vice-versa.

Phrasebook

The conversion from/to the internal representation of an application to/from OpenMath object representation is done via a Phrasebook corresponding to this application. A phrasebook is software that performs this conversion. This is the only software that should contain system-specific knowledge about representations and the implementation is generally up to the application programmer. The phrasebooks establish the correspondence to objects defined in CDs. A content dictionary (CD) maps the semantics level to the OpenMath level and describes which mathematical objects the corresponding OpenMath object denotes.

OpenMath Objects

An OpenMath object can be viewed as a tree and is also referred as a term.

The objects at the leaves of OpenMath trees are called basic mathematical objects. The basic mathematical objects supported by OpenMath are **integers, symbols, variables, floating-point numbers, character strings** and **bytearrays**.

Integers are integers in the mathematical sense, with no predefined range.

Symbols are at the heart of OpenMath. Each symbol has a name and is a member of a content dictionary. Each symbol has a prescribed, defined meaning. Each symbol has no more than one definition in a Content Dictionary. Other Content Dictionaries may define differently a symbol with the same name.

Variables are meant to denote parameters, variables or indeterminates. Variable names obey the same lexical rules as symbol names.

Floating-point numbers can be both single and double precision and follow IEEE formats.

Character strings are sequences of characters. These characters come form the Unicode standard.

Bytearrays are sequences of bytes. A bytearray is used to exchange arbitrary binary data.

The nodes of the trees representing OpenMath objects are made of **applications, error** and **attribution**. They are called constructor objects because they can be used to build compound OpenMath objects.

Application constructs an OpenMath object from a sequence of OpenMath objects. The first argument of application is called the "root" while the remaining objects will be called the arguments.

Error is made of an OpenMath symbol and a sequence of arguments (arbitrary OpenMath expression).

Attribution decorates an object with a sequence of pairs. Each pair consists of an OpenMath symbol as the attribute name and an associated OpenMath object as the attribute value.

Content Dictionaries

Content dictionaries play a central role in the OpenMath philosophy of communication of mathematical objects. It is the OpenMath content dictionary which actually holds the meanings of the mathematical objects to be communicated. For example, when two applications want to talk to each other about matrix multiplication, they must agree the same definition of matrix and matrix operations. These meanings are contained in a Content dictionary both agree upon.

The primary use of Content Dictionary is for a designer of phrasebooks. Therefore it should be as readable and precise as possible, to enable a phrasebook designer to effectively state which objects translate to which. Another possible use for OpenMath Content Dictionaries could rely on their automatic comprehension by a machine, in which case CD may have to be passed as data. In other words, it should be possible that CDs may passed in the same way as the OpenMath mathematical objects.

These goals guide the design of CDs, their structure and their relationships:

A content dictionary defines a set of related mathematical concepts and operations as a set of symbols. Each symbol has a name and has some information attached. Such

a symbol, for example, could be a constant *pi*, or an operation plus or a function *sin*. A symbol can also be used to define a representation for some mathematical objects. For example a polynomial can be built out of a set of constructors.

A mathematical object has a relatively complex structure. As we mentioned in Chapter 2, XML is suitable for describing complex data structures and is a powerful markup language for data communication. Therefore it naturally becomes the candidate in choosing the syntax for writing CDs.

Content Dictionaries are written in an XML encoding so that CDs can also be viewed as OpenMath objects and the structure of CDs can be described as precisely possible.

A special CD, called Meta, is written to specify the tags used in a CD, and a DTD has been written for validation of CDs.

Content Dictionaries have been designed to hold two types of information: that belonging to the whole CD, and that which is restricted to a particular symbol definition.

Information that belongs to the whole CD includes: the CD's name and description, expiry date, the status (official, experimental, private or obsolete), an optional URL and an optional list of CD's which this CD depends on.

Information that is restricted to a particular symbol includes: the symbol's name, description, an optional signature, optional examples, optional properties and default presentation information.

The following are two piece definitions for symbols **infinity** and **plus** respectively from the CD *Basic*:

```
<CDDefinition>
  <Name> infinity </Name>
  <Description>
     A value greater than any computable value.
     Infinity characterizes the positive real infinity.
     Negative infinity is represented as -infinity.
  </Description>
  <FunctorClass> Constant </FunctorClass>
   <CMP> a &it; infinity, "for any a which is not infinity itself"  </CMP>
   <CMP> a &it;= infinity, "for any a which is not infinity itself" </CMP>
</CDDefinition>

<CDDefinition>
  <Name> + plus </Name>
```

```
    <Description> The addition operator of any group </Description>
    <FunctorClass> Binary, Operator </FunctorClass>
    <CDAttributes> Associative, Commutative,
       Error( "invalid cancellation of infinity" )
    </CDAttributes>
    <Signature> (complex complex) -> complex </Signature>
    <Signature> (real real) -> real </Signature>
    <Signature> (rational rational) -> rational </Signature>
    <Signature> (integer integer) -> integer </Signature>
    <Signature> (symbolic symbolic) -> symbolic </Signature>
    <CMP> a+b=b+a, commutativity </CMP>
    <CMP> a+(b+c)=(a+b)+c, associativity </CMP>
    <CMP> a+0=a, identity </CMP>
    <CMP> 0+a=a, identity </CMP>
    <CDExamples>
    <TeX> a+(b+c)=(a+b)+c </TeX>
    <equal>
      <plus>a
        <Parenthesize>
         <plus> b c </plus>
        </Parenthesize>
      </plus>
      <plus>
        <Parenthesize>
         <plus> a b </plus>
        </Parenthesize>
        c
      </plus>
    </equal>
    </CDExamples>
  </CDDefinition>
```

Most element meanings can be conceived from the tag names themselves. The CMP element contains a "commented mathematical property", in the form of an equation or formula and an associated description.

Currently, there are three CD's with official status: they are Meta, Basic and Poly.

There are four CD's with experimental status: they are NonComm, Inert, LinAlg and Programming. With the increasing acceptance of OpenMath, there will be more CD's coming, and ideally one for each mathematical area.

Since the Basic CD covers basic concepts in many mathematical areas, a set of different CDs, see [10], have been formed by splitting the Basic CD. These are

```
alg.ocd         A cd of basic algebraic concepts
arith.ocd       A cd of arithmetic functions
calculus.ocd    A cd of calculus functions
comm.ocd        A cd of commutative arithmetic functions
fns.ocd         A cd of function functions
integer.ocd     A cd of integer functions
interval.ocd    A cd of interval functions
limit.ocd       A cd of limit functions
linalg.ocd      A cd of linear algebra functions
list.ocd        A cd of list functions
logic.ocd       A cd of logic functions
quant.ocd       A cd of quantifier functions
relation.ocd    A cd of relations
set.ocd         A cd of set functions
stats.ocd       A cd of basic statistical functions
sumprod.ocd     A cd of sum and product functions
transc.ocd      a cd of transcendental functions
```

The XML Encoding of OpenMath

XML is used in OpenMath in two places: one is for the encoding of CDs as we have seen above; the other is in mapping OpenMath objects to byte streams in the communication layer. Concretely, XML encodes OpenMath objects to byte streams. These byte streams constitute a low level representation that can be easily exchanged between processes via e-mail, cut-and-paste, sockets etc. or stored and retrieved from files.

Using XML encoding has the following two benefits:

1. Since the usual character set is used, it can be easily included in most documents and transport protocols. Also, it is both readable and writable by a human.

2. The text produced by this encoding can be included in XML document, which could employ widely existing XML process tools.

S. Dalmas *et al* have supplied a grammar for XML encoding in [11], with which the OpenMath objects can be easily encoded. The following simple example shows some encoded OpenMath objects:

```
<OMOBJ>
  <OMA>
   <OMS cd = "transc" name = "cos"/>
   <OMV name = "x"/>
  </OMA>
</OMOBJ>
```

Symbols are encoded using the **OMS** element. This element has two attributes: *cd* and *name*. The value of *cd* is the name of the Content Dictionary in which the symbol is defined and the value of *name* is the name of the symbol. For example, `<OMS cd = "transc" name = "cos"/>` is the encoding of the symbol named *cos* in the Content Dictionary named *transc*.

Variables are encoded using the **OMV** element, with only one attribute name whose value is the variable name. `<OMV name = "x"/>` is the encoding of a variable named x. If a variable is specified as bounded, the element `<OMBVAR>` is generally needed and the whole encoding will be:

```
<OMBVAR>
  <OMV name="x"/>
</OMBVAR>
```

Integers are encoded using the **OMI** element, e.g. `<OMI>1</OMI>`.

Applications are encoded using the **OMA** element. The application whose root is the OpenMath object e_0 and whose arguments are the OpenMath objects e_1 ... e_n is encoded as

`<OMA>` C_0 C_1 C_n `</OMA>`

where C_j is the encoding of e_j.

Binding is encoded using **OMBIND** element. The binding by the OpenMath object e_0 of the OpenMath variable e_1 in the object e_2 is encoded as

`<OMBIND>` C_0 C_1 C_2 `</OMBIND>`

where C_j is the encoding of e_j.

The following is somewhat complicated example:

```
<OMOBJ>
<OMBIND>
  <OMS cd="sumprod" name="sum"/>
  <OMBVAR>
```

```
    <OMV name="x"/>
  </OMBVAR>
  <OMA>
    <OMS cd="basic" name="tuple"/>
    <OMA>
      <OMS cd="interval" name="discrete-interval"/>
      <OMI> 1 </OMI>
      <OMV name="n"/>
    </OMA>
    <OMA>
      <OMS cd="arith" name="power"/>
      <OMV name="x"/>
      <OMI> 2 </OMI>
    </OMA>
  </OMA>
</OMBIND>
</OMOBJ>
```

Chapter 4

MathML

4.1 What is MathML?

MathML, or Mathematical Markup Language, is a low-level format for describing mathematics as a basis for machine-to-machine communication. It provides a much-needed solution to including mathematical expressions in Web pages.

MathML was released as a W3C (World Wide Web Consortium) recommendation on April 7th, 1998. MathML is the first application of XML to be issued as a W3C Recommendation [3].

MathML is intended to facilitate the use and re-use of mathematical and scientific content on the Web and for other applications such as computer algebra systems, print typesetters, and voice synthesisers. MathML can be used to encode both mathematical notations for high-quality visual display, and mathematical content for more semantic applications such as scientific software or voice synthesis.

4.2 Why MathML?

Encoding mathematics for computer processing or electronic communication is much older than the web. Several kinds of markup methods for mathematics, in particular *Tex*, had been widely used before the Web became a prominent communication medium. It was and remains very common for researchers to write papers containing encoded-form based on ASCII character set, and then e-mail the papers to each other.

Since the Web made the communication of information more effective it quickly became very popular. Naturally, mathematicians want to make their research work available on the Web. However, HTML, a language initially designed by scientists for scientists, has proven to be limited in helping mathematicians do their jobs. Most

complex mathematical notation or formula cannot be formatted simply by HTML tags and the mathematicians have to find a way to solve this problem.

One possibility is to include Tex in HTML to markup the mathematical notation. This idea may work in some contexts if the commercial browsers can support these Tex tags. The problem is that it will not satisfy many important requirements of publishing on the Web: it cannot markup mathematical contents and therefore manipulation of the mathematical materials interactively is impossible; and it will not allow reformatting when the window size is changed. Also, Tex normally groups expressions and subexpressions too coarsely. All these problems make the commercial browser companies lose interest in this idea.

Currently a common way to show math is to use GIF images of these notations and formulas and insert them into an appropriate tag. This "image-based" solusion work well in many cases. However, it has an obvious disadvantage: it is too image primitive. The primitive can result in poor document quality, be difficult to create, be slow in loading. Furthermore, mathematical information contained in images is not available for searching, indexing, or reuse in other applications.

For example we can use an image to hold the equation [$3^{3^{x^2}} = 20$] in a mathematical document. We can first give a suitable size to match the surrounding text. Later when a different system is used to display the document using different fonts, the image could be either too big or too small, destroying the professional appearance of the document. In addition, if we want to search all documents containing "= 20 ", because of the primitive of the image, this equation will be missed. Furthermore, if we want to copy the equation and paste it to a computer algebra system, the image property will not allow this action.

The World Wide Web Consortium observed the seriousness of these kinds of problems. Several proposals for HTML Math were then raised, and a working group (W3C Math Working group) was constituted aiming to address these problems.

As we have discussed in the previous chapter, similar problems have also appeared in other disciplines and may appear in the future disciplines. Ultimately the extension schema can not solve key weakness of HTML. XML, with "extensibility", will become popular for the Web. Considering the advantages of XML and its wide acceptance (which will make XML be supported by many tools) the W3C decided to base HTML Math on XML and call it MathML.

4.3 How Does MathML Work?

4.3.1 Mathematical Notations and Content

An important feature of mathematics is the use of a complex and highly evolved system of two-dimensional symbolic notations. Although mathematical ideas exist independently of the notations that represent them (some philosophers of mathematics may argue this point), part of the power of mathematics to describe and analyze derives from its ability to represent and manipulate ideas in symbolic form. So, the relation between meaning and notation is subtle. How to effectively capture both mathematical notation and the content on the Web is really a challenging job.

Marking up mathematical notation is relatively easy compared to marking up mathematical content since several typesetting languages, such as Tex and part of HTML, already exist. Parts or principles of these can be borrowed by MathML. However, marking up mathematical content on the Web is a completely new area and obviously more effort is needed on that part.

Furthermore, mathematical research continuously produces new notations to represent new mathematical ideas as well as existing ideas. Therefore an extension mechanism must also be taken into consideration in the design of MathML.

Several years since its inception, MathML has evolved from a "proposal" to a formal recommendation of the W3C. The MathML group proposed a language containing the following three groups of elements.

1. *Presentation elements*, which are concerned with layout and rendering of the mathematical notation.

2. *Content elements*, which encode the mathematical constructs or 'meaning' of an expression.

3. *Interface elements*, which provide a mechanism for embedding MathML expressions within an HTML page, and for passing the necessary information between an HTML browser and a MathML helper application doing the MathML processing.

In addition to these elements, MathML also has attributes for describing properties of some elements.

4.3.2 MathML Presentation Elements

Presentation elements describe the structure of mathematical notation structure. There are currently 28 presentation elements that accept over 50 attributes. Presentation elements are divided into two classes:

1. Tokens, which have only **#PCDATA** as content.
2. Layout schema, which have only other MathML elements as content, or are canonically empty.

Expressions in traditional math notation are recursively built out of layout schema. The recursion terminates with tokens or empty layout schema.

Tokens

All tokens (in the syntactic sense) in a mathematical expression are enclosed by token tags. The primary token types are identifiers (variables, function names, ...) numbers, operators, fences (e.g. parentheses), and string literals. There are also token elements for text or whitespace having no mathematical meaning. The following is the Token element table:

```
<mi>        identifier
<mn>        number
<mo>        operator, fence, or separator
<ms>        string literal
<mtext>     text
<mspace/>   space
```

Since *mi*, *mn* and *mo* are the most important presentation elements, we give more detailed uses here:
mi elements indicate that their contents should be displayed as identifiers. This means that single character identifiers like x and h should appear in italics, while multi-character identifiers like 'sin' and 'log' should be in an upright font. E.g. `<mi> x </mi>`

Attributes include font properties like fontweight, fontfamily and fontslant as well as general properties like fontcolor and background.
mn elements indicate that their contents should be rendered as numbers, which generally means in an upright font. E.g. `<mn>123</mn>`

Attributes are like those for mi.

mo elements are the most complex token schema. They indicate that their contents should be displayed as operators, but how operators are displayed is often quite complicated. For example, the spacing around different operators can vary. Operators like sums and products have special conventions for displaying limits as scripts. Still other operators such as vertical rules stretch to match the size of the expression that they enclose.

In MathML, rendering software is expected to contain an "operator dictionary", which contains information about how different operators are conventionally rendered. However, everything about how an operator should be displayed can be controlled directly by using attributes. Attributes include properties like lspace, rspace, stretchy, and movablelimits.

The **mo** element is also used to mark-up other symbols which are only operators in a very general sense, but whose layout properties are like those of an operator. Thus, mo elements are used to mark-up delimiter characters like parentheses (which stretch), punctuation (which has uneven spacing around it) and accents (which also stretch). One can use attributes to indicate that the contents of a mo should be treated as one of these related types.

The following list gives some examples of using `<mo>`:

```
<mo> + </mo>
<mo> &lt; </mo>
<mo> and </mo>
<mo> &InvisibleTimes; </mo>
<mo>(</mo>
```

Layout Schemata

The layout schemata fall into four classes:

1. General layout schemata

2. Script and limit schemata

3. Tables and matrices

4. Enlivening expressions

1. General layout schemata

These include

```
<mrow>     group any number of sub-expressions horizontally
<mfrac>    form a fraction from two sub-expressions
<msqrt>    form a square root sign (radical without an index)
<mroot>    form a radical with specified index
<mstyle>   style change
<merror>   enclose a syntax error message from a
           preprocessor
<mpadded>  adjust space around content
<mphantom> make content invisible but preserve its size
<mfenced>  surround content with a pair of fences
```

The most common and important general purpose layout schema is the mrow element. We will describe **mrow** and some other common elements in more detail:

```
<mrow> child1 ... </mrow>
```

An **mrow** element can contain any number of child elements. These child elements are displayed along the baseline in a horizontal row. In addition to positioning schemata in a row, the mrow can also be used to group together any number of terms so that this group can be viewed as a single unit, for example, for displaying as a subscript, or supscript.

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

The **mfrac** element expects exactly two children, the first of which will be positioned as the numerator of a fraction, and the second as the denominator. By setting the linethickness attribute to 0, the mfrac element can also be used for binomial coefficients.

```
<msqrt> child1 ... </msqrt>
```

The **msqrt** element accepts any number of children, and displays them under a radical sign.

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

The **mroot** element is nearly identical to the msqrt element, except it expects a second child, which is displayed above the radical in the location of the n in an nth root.

```
<mfenced> child ... </mfenced>
```

The **mfenced** element is like an mrow, except that it displays the enclosed in parentheses. Using attributes, one can set the beginning and ending delimiter character, as well as internal separator characters like commas.

```
<mstyle> child ... </mstyle>
```

The **mstyle** element is also like an mrow except that it handles attributes differently. The mrow element has almost no attributes of its own, while the mstyle elements can be used to set any MathML attribute.

2. Script and Limit Schemata

This includes the following element tags:

```
<msub>             attach a subscript to a base
<msup>             attach a superscript to a base
<msubsup>          attach a subscript-superscript
                   pair to a base
<munder>           attach an underscript to a base
<mover>            attach an overscript to a base
<munderover>       attach an underscript-overscript
                   pair to a base
<mmultiscripts>    attach prescripts and tensor
                   indices to a base
```

These tags are used to position one or more scripts around a base. Saying that mathematical notations are two-dimensional symbols mainly means sub- or superscript.

Their syntaxes are quite natural and the only thing to remember is the order of their augments. We give some of them here:

The syntax for **msub** is

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

The attribute is subscriptshift, which specifies the minimum amount to shift the baseline of subscript down.

The syntax for **msubsup** is

```
<msubsup> base   subscript   supscript</msubsup>
```

The attributes are subscriptshift and supscriptshift. Supscriptshift specifies the minimum amount to shift the baseline of superscript up.

The syntax for **mover** is

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

The attribute is *accent*, which controls whether *overscript* is drawn as an "accent" (diacritical mark) or as *limit*. The main difference between an *accent* and a *limit* is that the *limit* is reduced in size whereas an *accent* is the same size as the base.

3. Tables and Matrices

Table and matrix schemata include

```
<mtable>         table or matrix
<mtr>            row in a table or matrix
<mtd>            one entry in a table or matrix
<maligngroup/>   alignment group marker
<malignmark/>    alignment point marker
```

Matrices, arrays and other like mathematical notations are marked up using `<matable>`, `<mtr>` and `<mtd>` elements. These elements are similar to the `<table>`, `<tr>` and `<td>` elements in HTML, except that they provide specialized attributes, in order to provide the fine layout control necessary for, for example, commutative diagrams or block matrices.

4. Enlivening expressions

This category contains only one element tag:

```
<maction>
```

This element provides a mechanism for binding actions to expressions. For example, in lengthy mathematical expressions, a render might allow a reader to toggle between an ellipsis and a much longer expression that it represents.

4.3.3 MathML Content Elements

Content elements describe mathematical objects directly, as opposed to describing the notation that represents them.

Providing a specific encoding to describe mathematical objects directly is absolutely necessary. Even a disciplined and systematic use of presentation tags cannot properly capture the semantic information. This is because, without additional information, it is impossible to decide if a particular presentation was chosen deliberately to encode the mathematical structure or simply to achieve a particular visual or aural effect. Furthermore, an author using the same encoding to deal with both the presentation and mathematical structure might find a particular presentation encoding unavailable simply because convention had reserved it for a different semantic meaning. By encoding the underlying mathematical structure explicitly, without regard to how it is presented aurally or visually, we are able to interchange information more precisely with those systems that are able to manipulate the mathematics.

Mathematics consists of a large number of disciplines, however. Designing an encoding language to capture all of the meaning for each discipline will take too much work and is practically impossible. MathML takes care of only some commonly used disciplines and materials at a relatively simple level. The W3C Working group chose the following subject areas to be included in MathML:

Arithmetic, Algebra, Logic and Relations
Calculus
Set theory
Sequences and series
Trigonometry
Statistics
Linear Algebra

Also, as the specification said, "it is not claimed, or even suggested, that the proposed element set is complete for these areas, but the provision for author extensibility greatly alleviates any problem which omissions from this finite list might cause."

Content markup consists of about 75 elements accepting roughly a dozen attributes. These can be grouped into the following categories based on their usage:

Containers
Operators
Qualifiers
Relations

Conditions
Semantics

These are the building blocks from which MathML content expressions are constructed. We will give examples for each category.

Containers: containers provide a means for the construction of mathematical objects of a given type.

Tokens: **ci**, **cn** (corresponding to presentation tokens mi and mn)
ci is used to construct variables, or symbols. For example,

```
<ci>v</ci>
```

encodes a scalar symbol v.

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

encodes a vector variable.
cn is used to represent numbers. For example,

```
<cn>1234</cn>
```

encodes the number 1234.

```
<cn base = "8">1234</cn>
```

encodes the number with base "8".

Constructors: constructors produce a new type by combining elements into familiar compound objects, e.g. intervals and lists. Some of these constructors are **interval**, **list**, **matrix vector**, **apply**, **fn** and **reln**. For example,
to produce an "open-closed" interval, the encoding is

```
<interval closure = "open-closed">
   <ci>a</ci>
   <ci>b</ci>
</interval>
```

The **apply** element is used to apply a function or operator to its arguments to produce an expression representing an element of the range of the function. This is one of the commonly used elements. As an example, $sin(x)$ is encoded as

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

The **fn** element is used to identify an expression as a user-defined function or operator. e.g. if f and g are two functions, the new function f+g is encoded as

```
<fn>
  <apply>
    <plus/>
    <ci>f</ci>
    <ci>g</ci>
  </apply>
</fn>
```

Operators : We have already seen some examples of this category, such as `<sin/>` and `<plus/>` from the above.

Some examples of other operators are:

```
minus, times, over, power, min, max:  mathematical operators
                                      and constructors.
ln, log, int, diff:                   logarithms, integral
                                      and differentials.
mean, median, mode:                   statistics
```

There are two features in this category:

1. From the point of view of usage, MathML regards functions (e.g. sin, cos) and operators (e.g. plus, minus) in the same way.

2. MathML predefined functions and operators are all canonically empty elements.

Qualifier: qualifiers are used to specify some operator's meaning more fully.

Qualifiers: **lowlimit, uplimit, bvar, degree, logbase, interval**, and **condition**.

Operators taking qualifiers: **int, sum, product, diff, limit, log**, and **moment**.

A typical example is:

```
<apply>
  <int/>
  <bvar><ci>x</ci></bvar>
  <lowlimit><cn>0</cn></lowlimit>
  <uplimt><cn>1</cn></uplimit>
  <apply>
    <power/>
    <ci>x</ci>
    <cn>2</cn>
  </apply>
</apply>
```

for $\int_0^1 x^2\, dx$

Relations: The content elements dealing with relations are:

```
neq                          binary relation
implies                      binary logical relation
in, notin,
notsubset, notpresubset      binary set relation
tendsto                      binary series relation
eq, leq, lt, geq, gt         n-ary relation
subset, presubset            n-ary set relation
```

Relations are characterised by the fact that, if an external application were to evaluate them, they would typically return a truth value that is true or false.

A typical example is

```
<reln>
  <lt/>
  <ci>a</ci>
  <ci>b</ci>
</reln>
```

By using the container **reln**, we construct an expression $a < b$.

Conditions: this category contains one element: **condition** which is used to define the "such that" construct in mathematical expression: for example to encode "x such that $x^2 > 4$", we write

```
<bvar><ci>x</ci></bvar>
<condition>
```

```
    <reln><gt/>
      <apply><power/>
        <ci>x</ci>
        <cn>2</cn>
      </apply>
      <cn>4</cn>
    </reln>
  </condition>
```

Semantics: this category contains three elements: **semantics, annotations** and **annotation-xml**.

Sometimes the mathematical structure described by content elements may not be sufficient for specific applications. These content elements are used to encapsulate additional information required for some applications such as computer algebra systems.

Semantics is the container element for a MathML construct together with its semantic mapping information,

Annotation encapsulates this information in non-XML form.

Annotation-xml encapsulates this information in well-formed XML.

4.3.4 The Top-level Interface Element

The third category contains a single element: **math**. MathML specifies this element with two points of view.

With "inward looking", the **math** element is the top-level element. All other MathML content must be contained in a **math** element; equivalently, every valid, complete MathML expression must be contained in `<math>` tags. The **math** element must always be the outermost element in a MathML expression. It is an error for one math element to contain another and so, applications which return sub-expressions of other MathML expressions, (for example as the result of a cut-and-paste operation) should always wrap them up in `<math>` tags. Similarly, applications which insert MathML expressions into other MathML expressions must take care to remove the `<math>` tags from the inner expressions.

When embedding MathML in HTML, the **math** element encapsulates each instance of MathML markup within an HTML page. As such, the **math** element provides an attachment point for information, which affects a MathML expression as a whole. For example, the **math** element is the logical place to attach a style sheet by using the corresponding attributes.

From an outward looking perspective, the **math** element may serve as an interface for embedding MathML in HTML. This means it must be aware of its surrounding environment, and provide a mechanism for assigning information between the browser and the MathML renderer so that MathML can be integrated properly in the HTML document. Obviously this problem is actually the same as embedding XML into HTML and designing a solution is a cooperative activity of several W3C groups.

Chapter 5

The Relationship Between MathML and OpenMath

5.1 General Concepts

As we discussed in the previous chapters, we know that MathML will be the standard for representing mathematics on the Web, while OpenMath defines a platform-independent standard for the representation of mathematical objects so that they may be exchanged in a semantically meaningful way between various computational tools. Due to these different purposes, MathML is primarily for presentation, with less semantics than OpenMath, and OpenMath is primarily for semantics, with less presentation than MathML. They basically represent two different aspects of mathematical objects: they are not opposite but complementary. Each of them has the provision to incorporate adding elements into the other. When there is such a need to represent a mathematical object with both the presentation and semantics, it is possible to add elements in one aspect to another aspect.

5.2 Using OpenMath Annotation in MathML

MathML contains Semantic Mapping Elements: `<annotation>`, `<annotation-xml>` and `<semantics>` for adding semantic meaning. For example, these can be used to represent $\sin x$ with both presentation and semantics:

```
<semantics>
  <mrow>
    <mi>sin</mi>
```

```
      <mo> &ApplyFunction;</mo>
      <mi>x</mi>
    </mrow>
    <annotation-xml encoding = "OpenMath">
      <OMA>
        <OMS cd = "Basic" name = "sin"/>
        <OMV name = "x"/>
      </OMA>
    </annotation-xml>
  </semantics>
```

This procedure can be used to represent any composite mathematical object. That is, a mathematical object expression tree with any depth, so long as the corresponding semantic unit can be found in OpenMath.

For example, to use OpenMath annotation in MathML for the expression $\sum_{i=0}^{n} \sin(ih)$, we can have:

```
<mrow>
  <mover>
    <munder>
      <mo> &sum;</mo>
      <mrow>
        <mi>i</mi>
        <mo> &equal;</mo>
        <mn>0</mn>
      <mrow>
    </munder>
    <mi>n</mi>
  </mover>
  <semantics>
    <mi>sin</mi>
    <mfenced>
      <mrow>
        <mi>i</mi>
        <mo>&times;</mo>
        <mi>h</mi>
      </mrow>
    </mfenced>
    <annotation-xml encoding = "OpenMath">
      <OMA>
```

```
          <OMS cd = "Basic" name="sin"/>
          <OMA>
            <OMS cd = "Basic" name="times"/>
            <OMV name = "i"/>
            <OMV name = "h"/>
          </OMA>
        </OMA>
      </annotation-xml>
    </semantics>
  </mrow>
```

Since <OMS cd = "sumprod" name = "sum"/> in OpenMath needs the function to be summed as its argument, we can not give the semantic meaning for the summation part of <mover>...</mover>

If we want to add semantic meaning to the whole object, this would become the first case, that is

```
<semantics>
  <mrow>
    <mover>
      <munder>
        <mo>&sum;</mo>
        <mrow>
          <mi>i</mi>
          <mo>&equal;</mo>
          <mn>0</mn>
        </mrow>
      </munder>
      <mi>n</mi>
    </mover>
    <mtext>
      <mi>sin</mi>
      <mfenced>
        <mrow>
          <mi>i</mi>
          <mo>&times;</mo>
          <mi>h</mi>
        </mrow>
      </mfenced>
     </mtext>
```

```
    </mrow>
    <annotation-xml encoding = "OpenMath">
      <OMBIND>
        <OMS cd = "sumprod" name = "sum"/>
        <OMBVAR>
          <OMV name = "i"/>
        </OMBVAR>
        <OMA>
          <OMS cd="basic" name="tuple"/>
           <OMA>
              <OMS cd="interval" name="discrete-interval"/>
              <OMI> 0 </OMI>
              <OMV name= "n">
           </OMA>
        <OMA>
          <OMS cd="basic" name="sin"/>
            <OMA>
              <OMS cd="basic" name="times"/>
              <OMV name = "i"/>
              <OMV name = "h"/>
            </OMA>
        </OMA>
      </OMBIND>
    </annotation-xml>
  </semantics>
```

5.3 Using MathML Annotation in OpenMath

What we discussed in the previous section was how to add semantic meaning from OpenMath in a MathML document. It is possible to introduce similar Presentation Mapping Elements in OpenMath, say <OMPRESENTATION> and <OMANNOTATION> for adding presentation in OpenMath. Then we can have

```
<OMPRESENTATION>
   <OMA>
     <OMS cd = "Basic" name = "sin"/>
     <OMV name = "x"/>
   </OMA>
   <OMANNOTATION encoding = "MathML">
```

```
        <mrow>
          <mi>sin</mi>
          <mo> &ApplyFunction;</mo>
          <mi>x</mi>
        </mrow>
      </OMANNOTATION>
    </OMPRESENTATION>
```

It is worthwhile to indicate that even though MathML has its content markup elements and currently they are similar to OpenMath, OpenMath will be easily extended to include much more semantic mathematical objects than MathML content markup. The ideal situation will be that every math areas will have its own CD, e.g. topology, geometry and stochastic processes, etc. The complementary relationship between MathML and OpenMath will thereby be kept, at all times.

Chapter 6

An Overview of XSL

As we have already discussed in the previous chapters, HTML is used to write web pages with each tag's meaning well-defined and understood: `<H1>` makes a heading, `<p>` makes a paragraph, `<IMG>` loads a graphic and so on. The browser can directly understand each tag's meaning and give the corresponding presentation, but the deficiency is that HTML has a fixed set of tags and is not extensible. With XML, we can write whatever tags we want and therefore we could create meaningful tags according to the context we want to describe. The browsers or other media can not, however, directly understand what meanings the tags want to express in a XML document and so they do not know how to give suitable presentations. XSL fills this gap. It translates the tags in a general XML document to the vocabularies that those devices can understand.

XSL, or Extensible Stylesheet Language, is a language for expressing stylesheets for XML [4],[5]. Since there is no underlying semantics to augment for XML, XSL must specify how each element should be presented and what the element is. For this reason, XSL defines not only a language for expressing the style sheets, but also a vocabulary of "formatting objects" that have the necessary base semantics.

We informally describe the presentation process:

Two concepts will be used very often:

The *source tree*: parsed by an XML parser from the source document

The *result tree*: the product from the XSL processor.

There are two parts to the presentation process. First, the result tree is constructed from the source tree. Second, the result tree is interpreted to produce the formatted output on a display, on paper, in speech or other medium.

The first part, constructing the result tree, is achieved by associating patterns with

templates. A pattern is matched against elements in the source tree. A template is instantiated to create part of the result tree and guide the processor to the further activity, e.g., to choose another element to process. The result tree is separate from the source tree. The structure of the result tree can be completely different from the structure of the source tree. In constructing the result tree, the source tree can be filtered and reordered, and arbitrary structure can be added.

The second part, formatting, is achieved by using the formatting vocabulary specified in XSL to construct the result tree.

However, XSL does not require the result trees to use the formatting vocabulary and thus can be used for general XML transformations. This feature is what we want in the realization of our macro mechanism in Chapter 7. So, for our purpose, we will not care about formatting vocabulary and just focus on how XSL specifies the rule for a style sheet to do transformations.

To the end of transformation, we need to know three things:

1. The structure of a style sheet.
2. How to form a template in XSL?
3. What are the patterns in XSL?

6.1 The Structure of A Style Sheet

A style sheet is an XML document, and so it begins with the XML version information

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

After that is the `xsl:stylesheet` element, which may contain the following types of XSL elements:

1. `xsl:import`
2. `xsl:include`
3. `xsl:id`
4. `xsl:strip-space`
5. `xsl:preserve-space`
6. `xsl:macro`
7. `xsl:attribute-set`
8. `xsl:constant`
9. `xsl:template`

These elements may appear zero or more times. Most stylesheets contain `xsl:template` in the `xsl:stylesheet`. Here is a simple example:

```
<?xml verson="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match = "apply">
 <xsl:apply-template select="*[first-of-any()]"/>
</xsl:template>
<xsl:template match = "apply/sin">
 <mrow>
  <mi>sin</mi>
   <mo>&InvisibleApply;</mo>
    <mrow>
     <xsl:apply-templates select = "ancestor(apply)/*[not(first-of-any())]"/>
    </mrow>
 </mrow>
</xsl:template>

<xsl:template match = "ci">
 <mi>
  <xsl:apply-templates/>
 </mi>
</xsl:template>

<xsl:template match = "/">
 <xsl:apply-templates/>
</xsl:template>

</xsl:stylesheet>
```

6.2 How to Form a Template?

An XSL template is defined using a small set of XSL elements and a set of literal result elements.

XSL elements are those with the `xsl` namespace prefix. Result elements are the others.

The XSL elements used most often in our work are as follows :

1. **xsl:template**

   Specify a template rule for a specific matched pattern. The content of the **xsl:template** element is the template.

2. **xsl:apply-templates**

   Guides the XSL processor to find the appropriate templates to apply based on the results of the pattern. If no pattern is supplied, the results will be all child elements of the current node

3. **xsl:attribute**

   Used to add attributes to result elements

4. **xsl:if**

   provides a simple conditional processing within a template.

5. **xsl:choose**

   Provides multiple conditional testing; used with the xsl:otherwise and **xsl:when** elements.

6. **xsl:copy**

   Copies a node from the source to the result.

7. **xsl:otherwise**

   Provides multiple conditional testing; used with the **xsl:choose** and **xsl:when**

The literal result elements in our work are, naturally, those elements in MathML and OpenMath.

A template contains two parts: the matching part and the processing part. Matching part is the attribute part and processing part is the content of a template.

6.3 What Are the Patterns in XSL?

A pattern is a string, which selects or matches a set of nodes in a source document. XSL divides patterns into two kinds by their functionality: select patterns and match patterns.

Patterns generally appear in templates. As we have stated in the above section, a template contains two parts: the matching part and the processing part.

The match patterns are the patterns appearing in the matching part. If a node matches a match pattern, the corresponding template is the candidate chosen.

The select pattern appears in the processing part of the template. The selection is relative to the current node, which means with the current node as context. The select patterns are used to guide the processor to further processing in a template, based on the result of the selection.

In other words, they are used in different directions: match patterns are used for a node to find a matched template, while select patterns are used to find the corresponding nodes or test their existence for further processing. Their syntaxes, however, have little difference.

We will next have a look at the patterns used most often in our work.

Matching on name

```
<xsl:template match = "apply">
```

matches all **apply** elements in the source document.

Matching on ancestry

```
<xsl:template match = "apply/sin">
```

matches all **sin** elements with the parent element **apply**.

```
<xsl:apply-templates select = "ancestor(apply)/ci"/>
```

selects all **ci** elements with **apply** element as parent and this **apply** element is also the ancestor of the current node.

Matching the root

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

would select the root pattern

Wildcard matches

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

would match every node in the source document.

```
<xsl:template match = "apply/*>...</xsl:template>
```

would match every element with **apply** element as parent

Built in template rule

There is a built-in template rule to allow processing to continue in the absence of a successful pattern match by an explicit rule in the stylesheet. This rule applies to both element nodes and the root node:

```
<xsl:template match = "*|/">
  <xsl:apply-templates/>
</xsl:template>
```

A built-in rule can be overridden by an explicit rule.

Matching on attributes

```
<xsl:template match = "cn[@type = "constant"]">
```

would match the **cn** element with "type" attribute with value "constant". Here @ represents attribute.

Matching on child

```
<xsl:template match="apply[sum]">
```

would match the apply element with a child sum.

Matching on position

The position of a node relative to its siblings can be tested. XSL supplies the following position qualifiers:

first-of-any()
succeeds if the node being tested is the first element child
last-of-any()
succeeds if the node being tested is the last element child
first-of-type()
succeeds if the node being tested is the first element of its element type
last-of-type()
succeeds if the node being tested is the last element of its element type
not()
used to negate a test
e.g.

```
<xsl:apply-templates select = "*[first-of-any()]">
```

would select the first child of the current element.

```
<xsl:apply-template select = "ci[not(last-of-type())]">
```

would select all child elements of ci except the last one for the current element.

A pattern "." selects the current node and ".." selects the parent of the current node.

```
<xsl:apply-templates select = "../bvar">
```

would select the **bvar** sibling elements of the current node.

Specificity of patterns

When a source element is matched against patterns, it is possible for it to match more than one distinct pattern. In this situation, XSL defines which pattern or patterns are the most specific.

We just give a simple example to show the specificity in decreasing order:

1. employee[@type= 'contract' @country= 'USA']
2. employee[@type= 'contract']
3. employee
4. *

6.4 Examples

In this section, we will give several examples. These examples will show that, for the same source document, we can obtain different result trees (for different presentations) by attaching different stylesheets.

Example 6.1.

Consider the following source fragment:

```
<?XML version = "1.0" ?>
<math>
  <combination>
    <all>    10 </all>
    <choose> 3  </choose>
  </combination>
</math>
```

With the stylesheet shown in Figure 6.1, the result is

```
<mfenced>
  <mfrac thickness="0">
     <mn> 10 </mn>
     <mn>  3 </mn>
  </mfrac>
</mfenced>
```

which will be rendered as

$$\binom{10}{3}$$

.

(Note: this is LaTex simulating an XML rendering. The same for other mathematical formulas appeared hereafter.)

With another stylesheet shown in Figure 6.2, the result is:

```
<mmultiscripts>
    <mi>C</mi>
    <mn> 3 </mn>
    <none/>
    <mprescripts/>
    <mn> 10 </mn>
    <none/>
</mmultiscripts>
```

which will be rendered as

$${}_{10}C_3$$

.

Example 6.2.

Consider the following source fragment:

```
<continued-fraction>
  <cf-elem> 3 </cf-elem>
  <cf-elem> 5 </cf-elem>
  <cf-elem> 6 </cf-elem>
  <cf-elem> 7 </cf-elem>
</continued-fraction>
```

With the stylesheet shown in Figure 6.3, the result is:

```
<mfenced open="[" close="]">
  <mn> 3 </mn>
  <mn> 5 </mn>
  <mn> 6 </mn>
  <mn> 7 </mn>
</mfenced>
```

which will be rendered as

[3,5,6,7].

With another stylesheet shown in Figure 6.4, the result is:

```
<mrow>
   <mn> 3 </mn>
   <mo> + </mo>
   <mfrac>
     <mn> 1 </mn>
     <mrow>
       <mn> 5 </mn>
       <mo> + </mo>
       <mfrac>
        <mn> 1 </mn>
        <mrow>
          <mn> 6 </mn>
          <mo> + </mo>
          <mfrac>
            <mn> 1 </mn>
            <mn> 7 </mn>
          </mfrac>
        </mrow>
     </mrow>
   </mfrac>
</mrow>
```

This will be rendered as

$$3 + \frac{1}{5+\frac{1}{6+\frac{1}{7}}}$$

```
<?xml verson="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

<xsl:template match = "combination">
  <mfenced>
    <mfrac thickness="0">
      <xsl:apply-templates/>
    </mfrac>
  </mfenced>
</xsl:template>

<xsl:template match = "all|choose">
  <mn>
    <xsl:apply-templates/>
  </mn>
</xsl:template>

<xsl:template match = "/">
 <xsl:apply-templates/>
</xsl:template>

</xsl:stylesheet>
```

Figure 6.1: Stylesheet 1 in Example 6.1

```
<?xml verson="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

<xsl:template match = "combination">
  <mmultiscripts>
      <mi> C </mi>
      <xsl:apply-templates select="*[last-of-any()]"/>
      <none/>
      <mprescripts/>
      <xsl:apply-templates select="*[first-of-any()]"/>
      <none/>
  </mmultiscripts>
</xsl:template>

<xsl:template match = "all|choose">
  <mn>
    <xsl:apply-templates/>
  </mn>
</xsl:template>

<xsl:template match = "/">
 <xsl:apply-templates/>
</xsl:template>

</xsl:stylesheet>
```

Figure 6.2: Stylesheet 2 for Example 6.1

```
<?xml verson="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

<xsl:template match = "continued-fraction">
  <mfenced open="[" close="]">
    <xsl:apply-templates/>
  </mfenced>
</xsl:template>

<xsl:template match = "cf-elem">
  <mn>
   <xsl:apply-templates/>
  </mn>
</xsl:template>

<xsl:template match = "/">
 <xsl:apply-templates/>
</xsl:template>

</xsl:stylesheet>
```

Figure 6.3: Stylesheet 1 in Example 6.2

```
<?xml verson="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

<xsl:template match = "continued-fraction">
    <xsl:apply-templates select="*[first-of-any()]"/>
</xsl:template>

<xsl:template match = "cf-elem">
  <xsl:choose>
    <xsl:when test=".[last-of-any()]"
      <xsl:apply-templates/>
    </xsl:when>
    <xsl:otherwise>
      <mrow>
       <mn>
         <xsl:apply-templates/>
       </mn>
       <mo> + </mo>
       <mfrac>
        <mn>1</mn>
        <xsl:apply-templates select="next()"/>
       </mfrac>
      </mrow>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match = "/">
  <xsl:apply-templates/>
</xsl:template>

</xsl:stylesheet>
```

where **next()** is a pattern extension to XSL, see section 8.3.2.

Figure 6.4: Stylesheet 2 in Example 6.2

Chapter 7

A Macro Mechanism for MathML

From the discussion of the previous chapters, we have seen that OpenMath and MathML are playing important roles in the communication of mathematical objects. Each of them emphasizes a different aspect: OpenMath is primarily for semantic meaning. MathML is primarily for presentation. They are complementary.

One of the most significant gaps with MathML is the lack of a macro mechanism to handle abbreviations and to abstract new concepts. Writing a MathML document becomes, sometimes, a verbose job: authors may have the occasion to repeat some complicated but constant notation. They may also, sometimes, find that MathML lacks pre-defined content elements for encoding their objects and operators. In particular, when a semantic meaning from OpenMath is needed to bind their objects, the authors may have to write it themselves.

All these situations seriously impair efficiency in the communication of mathematical objects. Therefore, it is necessary to develop a macro mechanism for MathML. Basically, this macro mechanism should include the following three applications:

1. Macros for abstracting different notational styles.
2. Macros for expanding abbreviations.
3. Macros for combined presentation-semantics markup.

In this chapter, we will present a prototype of such a macro mechanism. First we will describe the three macro application areas by examples. Then we will examine the suitability of XSL for the realization of the macro mechanism. Finally, we will give more examples to show that this macro mechanism is meaningful and powerful.

7.1 Macros for Abstracting Different Notational Styles

Many concepts in mathematics can be presented in different notational styles. For example, the number of combinations for choosing n elements from m elements can be expressed as $\binom{m}{n}$, C_n^m or ${}_mC_n$; a continued fraction can be expressed as $[n_1,...n_i]$ or $n_1 + \frac{1}{n_2+\frac{1}{\ddots}}$; in some literature $tg(x)$ is used for the tangent of x, while in other literature $\tan(x)$ is used.

In MathML, we need to use different presentation markups to give the different renderings.

For example, we have

```
<mfenced>
  <mfrac thickness="0">
     <mn> m </mn>
     <mn> n </mn>
  </mfrac>
</mfenced>
```

for $\binom{m}{n}$, and we have

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

for ${}_mC_3$. These two different presentation markups have the same correspondence in the content markup, i.e.

```
<?XML version = "1.0" ?>
<math>
  <combination>
    <all>    m  </all>
    <choose> n  </choose>
```

```
  </combination>
</math>
```

Macros for abstracting different notational styles means to obtain the different presentation markups from the same content markup.

7.2 Macros for Expanding Abbreviations

In order to markup $rank(U^T V) = 1$ in content elements, we have the following MathML fragment:

```
<reln>
 <eq/>
 <apply>
  <fn>
   <semantics>
    <ci><mo>rank</mo></ci>
    <annotation-xml encoding="OpenMath">
     <OMS cd = "linalg", name = "matrix-rank"/>
    </annotation-xml>
   </semantics>
  </fn>
  <apply>
   <times/>
   <apply>
    <transpose/>
    <ci>U</ci>
   </apply>
   <ci>V</ci>
  </apply>
 </apply>
<cn>1</cn>
</reln>
```

Since the concept "rank" from the linear algebra was not included in MathML as a pre-defined content element, we use the element **mo** to contain this symbol and insert it to **ci** with binding semantic meaning from OpenMath.

Suppose that the part

```
<fn>
 <semantics>
  <ci><mo>rank</mo></ci>
  <annotation-xml encoding="OpenMath">
   <OMS cd = "linalg", name = "matrix-rank"/>
  </annotation-xml>
 </semantics>
</fn>
```

can be represented by a macro <rank/>, and the part

```
<apply>
 <transpose/>
 <ci>U</ci>
</apply>
```

can be represented by the macro tags <tr>u</tr>.

Then the whole document would be condensed to

```
<reln>
  <eq/>
  <apply>
   <rank/>
   <apply>
    <times/>
    <tr>U</tr>
    <ci>V</ci>
  </apply>
  <cn>1</cn>
</reln>
```

which is much simpler and easier to write.

We call this kind of macro application Macros for Expanding Abbreviations. It is clear that this macro not only handles abbreviations but also provides a way to abstract new concepts. Once we have a macro tag library, we can use the macro tags to write relatively simpler condensed documents and we can abstract new concepts from other sources. With the help of some tools and the library we can expand the condensed document to the desired document.

7.3 Macros for Combined Presentation-Semantics Markup

As we have seen in Chapter 5, the combination of presentation-semantics markup could appear at different levels: it can be partially in some branches of the document tree or can be fully combined at the root of the document tree. Generally, it is preferable to have combinations at each node, rather than just at the root so that selection of subexpression retains semantic meanings.

General speaking, when a new concept is abstracted from OpenMath, the combination appeares as a part in an expression. For example, "rank" in the previous section is this case. We give more examples here:

Example 7.1 inverse matrix

To abstract the new concept "inverse matrix" from OpenMath to MathML content markup, we use the following binding:

```
<fn>
 <semantics>
  <ci><mo>mtrx-inverse</mo></ci>
  <annotation-xml encoding = " OpenMath ">
   <OMS cd="linalg" name="matrix-inverse"/>
  </annotation-xml>
 </semantics>
</fn>
```

and abbreviated it as a macro: `<mtrx-inverse/>`.

Example 7.2 group

To abstract the new concept "group" to MathML content markup, we could use the binding:

```
<semantics>
  <mi fontweight="bold"> G </mi>
  <annotation-xml encoding = " OpenMath ">
    <OMS cd="absalg.ocd" name="group"/>
  </annotation-xml>
</semantics>
```

and abbreviated it as a macro: `<Group> G </Group>`

When we want a mathematical expression on the Web to be able to be transferred to different computer algebra systems to be evaluated, we may need the full combination. This section addresses the macro application for this case.

If we want to write a MathML document to markup a mathematical object with both presentation and semantic meaning added from OpenMath, we will have to write both parts. For example, to represent `sin(x)` with both presentation and semantic meaning from OpenMath, we have:

```
<semantics>
 <mrow>
  <mi>sin</mi>
  <mo>&ApplyFunction;</mo>
  <mi>x</mi>
 </mrow>
 <annotation-xml encoding="OpenMath">
  <OMA>
   <OMS cd = "basic" name="sin"/>
   <OMV name = "x"/>
  </OMA>
 </annotation-xml>
</semantics>
```

However these two parts are just different aspects of the same mathematical object. In other words, they are images of different mappings from the same source. The source is

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

So, if we would have such mappings previously, then we could just need to write the source. And with the help of some tools we can obtain the images of the mappings from the source. This is the idea of Macro for combined presentation-semantics markup.

7.4 Realization of the Macro Mechanism

From the discussion of the above chapter we see that the realization of the macro mechanism for MathML is essentially to find a transformation from an XML source tree to an XML result tree. Since XSL can be used for general XML transformations, therefore it is possible to form the desired transformations with XSL. To do this, we need an XML parser to parse XML documents to XML trees and we need an XSL processor to do the transformations. We have written a validating XML parser and an XSL processor, in which we have implemented most of the specifications based on our needs. These tools are heavily used in our program "macroProcessor". This is the main program that is used to test and experiment with macro extension.

In the following we first give a simple description about our XML parser and XSL processor, then we investigate how to construct the desired mapping in our macro mechanism with XSL and whether or not the current XSL is powerful enough for the macro extension.

The XML Parser

Our XML parser accepts one argument, the filename of the XML document, and parses the data to an XML tree. The tree can be as an input to the XSL processor and could also be displayed with the proper indentation.

The parser has two options: it can either first parse the document completely and then check the validation against the requirements of the DTD, or check the validation during parsing.

Since this parser is for an experimental purpose, namely parsing MathML documents (documents comforming to the MathML DTD), we have made some assumptions in order to simplify the implementation:

1. Elements and #PCDATA cannot be siblings in the document type.
2. The number of appearance allowed for an element's children are the same.
3. The entities in the document can not be replaced.

Our XML parser is implemented with C++. We have designed 5 basic classes, which represent the concepts of **element, entity, attribute, symbol** and **parser**. With a template **list**, we have obtained four aggregation classes: **element list, entity list, attribute list** and **symbol list**. In addition, we have a class **tree**. These lists and tree hold the data in class **parser**. The total code is about 3500 lines.

The XSL Processor

Our XSL processor accepts one or two arguments. If only one argument appears, the transformation is an identity transformation for this argument. Otherwise, the first argument is the filename for the source tree and the second is the filename for the stylesheet (also an XML tree). The result tree can be transformed to the corresponding document or displayed with proper indentation.

We have implemented most of the patterns of the XSL specification.

The matching patterns include:

matching on name
matching on parent (not any ancestor)
matching on root
wildcard match
matching on an attribute
matching on child
matching on position
matching on sibling (not in the specification)

The select patterns include:

specific type child with position qualifier.
any child with position qualifier.
any node with the specific ancestor.

We have implemented most of the instructions in a template such as xsl:apply-templates, xsl:if, xsl:copy, xsl:choose, and so on.

It consists of a new class **transformer** and several other classes built in the XML parser. The code, not including the classes of the XML parser, is about 1500 lines. It is worthwhile to mention that we started our implementation in late October 1998, based on the XSL W3C Working Draft 18-August-1998 [4]. Some of the features that we found needed to be added to XSL for our work were later supplied in the new Draft of 16-December-1998. In January 1999, we modified our XSL processor according to the new draft.

7.5 Realization of Macros for Abstracting Different Notational Style

This case is straightforward — we simply use different stylesheets for the different notations. We have seen this situation in Section 6.4.

7.6 Realization of Macros for Expanding Abbreviations

What we need to do for expanding abbreviations is to write a stylesheet containing the templates to expand the macro tags and the template to copy the other general elements.

Example 7.3.

In order to markup $rank(U^T V) = 1$, we write the source document

```
 <reln>
   <eq/>
   <apply>
     <rank/>
     <apply>
       <times/>
       <tr>U</tr>
       <ci>V</ci>
     </apply>
   </apply>
   <cn>1</cn>
</reln>
```

where the "rank" tag and "tr" tag are macro tags and the others are general tags.

Now the stylesheet for expansion transformation would be

```
<?XML version = "1.0" ?>
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WD-xsl" >

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

```
</xsl:template>

<xsl:template match = "rank" >
  <fn>
    <semantics>
      <ci><mo>rank</mo></ci>
      <annotation-xml encoding = "OpenMath">
        <OMS cd="linalg" name="matrix-rank"/>
      </annotation-xml>
    </semantics>
  </fn>
</xsl:template>

<xsl:template match = "tr" >
  <apply>
    <transpose/>
    <ci>
      <xsl:apply-templates />
    </ci>
  </apply>
</xsl:template>

<xsl:template match = "*|/">
  <xsl:copy>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

With this stylesheet, the XSL processor will first find the template for the root node. By initiating this template, the processor is directed by xml:apply-templates to find all the children of the root node which, in this case, has just one **reln** . Since **reln** does not have explicit match, the built-in rule is chosen, and first, **reln** is copied to the result tree; after that the processor is directed by xml:apply-template to find **reln**'s children; this procedure goes recursively until the **rank** element is chosen. Since **rank** has an explicit match, the matched template is initiated and the content (all are literal result elements) are copied to the result tree. The processor keeps on following the stylesheet until all guided things are processed. So we have the following

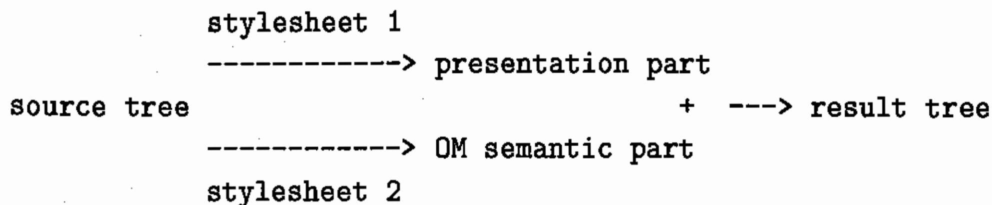
result document:

```
<reln>
  <eq/>
  <apply>
    <fn>
      <semantics>
        <ci><mo>rank</mo></ci>
        <annotation-xml encoding = "OpenMath">
          <OMS cd="linalg" name="matrix-rank"/>
        </annotation-xml>
      </semantics>
    </fn>
    <apply>
      <times/>
      <apply>
        <transpose/>
        <ci>U</ci>
      </apply>
      <ci>V</ci>
    </apply>
  </apply>
  <cn>1</cn>
</reln>
```

From the above examples, we can see that once we have a library for this kind of macro, that is, stylesheets containing suitable template rules for the macro tags, then we can write a simpler document. This shows the power of the macro mechanism. More examples can be seen in section 7.10.

7.7 Realization of the Macros for Combined Markup

There is a little bit more work to do for full combined markup. Since combined markup contains two parts: the presentation part and the semantic part (in OpenMath), we need to write two stylesheets to transform the source document to its presentation part and OpenMath semantic part respectively. The final result tree can be formed as the concatenation of the two parts with some "decoration". This procedure can be expressed by the following figure:



Example 7.4

In order to markup $\sin(x)$ with presentation and OpenMath semantic meaning, we first write the source document

```
<?XML version = "1.0" ?>
<math>
  <apply>
    <sin/>
    <ci>x</ci>
  </apply>
</math>
```

and stylesheet 1 for presentation transformation

```
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WO-xsl">

<xsl:template match = "apply">
  <xsl:apply-templates select="*[first-of-any()]"/>
</xsl:template>

<xsl:template match = "apply/sin">
 <mrow>
  <mi>sin</mi>
  <mo>&InvisibleApply;</mo>
  <mrow>
   <xsl:apply-templates select="ancestor(apply)/*[not(first-of-any())]"/>
  </mrow>
 </mrow>
</xsl:template>

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

```
    <mi>
      <xsl:apply-templates/>
    </mi>
  </xsl:template>

  <xsl:template match = "/">
    <xsl:apply-templates/>
  </xsl:template>

  </xsl:stylesheet>
```

This gives the presentation part:

```
<mrow>
 <mi>sin</mi>
 <mo>&ApplyFunction;</mo>
 <mi>x</mi>
</mrow>
```

Stylesheet 2 for OpenMath semantic part transformation:

```
<?XML version = "1.0" ?>
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WO-xsl">

<xsl:template match = "apply">
  <OMA>
    <xsl:apply-templates select="*[first-of-any()]"/>
  </OMA>
</xsl:template>

<xsl:template match = "sin">
  <OMS cd = "transc" name = "sin"/>
    <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "ci">
 <OMV name={.}/>
</xsl:template>

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

```
    <annotation-xml encoding = "OpenMath">
      <xsl:apply-templates/>
    </annotation-xml>
  </xsl:template>

  </xsl:stylesheet>
```

This gives the OpenMath semantic part:

```
  <annotation-xml encoding = "OpenMath">
    <OMA>
      <OMS cd = "transc" name="sin"/>
      <OMV name = "x"/>
    </OMA>
  </annotation-xml>
```

And therefore, the final result tree is

```
    <semantics>
      <mrow>
        <mi>sin</mi>
        <mo>&ApplyFunction;</mo>
        <mi>x</mi>
      </mrow>
      <annotation-xml encoding = "OpenMath">
        <OMA>
          <OMS cd = "transc" name = "sin"/>
          <OMV name = "x"/>
        </OMA>
      </annotation-xml>
    </semantics>
```

where `<semantics>` and `</semantics>` are decorated to the beginning and the end.

Remark: At the first glance, we probably have the conclusion that the macro mechanism does not save time for writing MathML document since writing a stylesheet takes more time than writing the desired document directly. However, when we have previously written many stylesheets and carefully classified them based on their content to form a library for reuse, we could just include the desired existing stylesheets to make a simple stylesheet. This technique can no doubt save a lot of time. And that is the power of the macro mechanism.

7.8 Remarks in Writing Stylesheets

From the above discussion, we see that an important issue for macro mechanism is to write the stylesheets. And, to examine if XSL is powerful enough for the macro extension, we need many stylesheets to test different cases.

We have written many stylesheets, especially stylesheets for transforming from MathML content to OM semantics symbols. For each OpenMath CD we have developed a corresponding stylesheet. These stylesheets are collected in the Appendix.

Writing a stylesheet for the abbreviation case is not too hard. Writing a stylesheet for the combined markup case, however, is not a trivial task since MathML content structure and presentation structure are generally different, and MathML content structure and OpenMath semantic structure may also have significant differences. Some differences even make the transformation from one structure to another impossible within the current XSL working draft. Some extensions are therefore necessary in order to get the job done or to do it in a more natural way.

In this section we discuss some considerations we took, and some valuable experience and techniques in writing our stylesheets. In the next section, we will describe extensions we introduced in our work.

Considerations, Experience and Skills

1. For some mathematical objects, their encoding structures of MathML content and OpenMath or MathML presentation have no difference and therefore the stylesheet can be written in the way of one-to-one correspondence, e.g.

– MathML–	–OpenMath–
`<apply>`	`<OMA>`
`  <sin/>`	`  <OMS cd="transc" name="sin"/>`
`  <cn>5</cn>`	`  <OMI>5</OMI>`
`</apply>`	`</OMA>`

– template rules –

```
<xsl:template match="apply">
 <OMA>
  <xsl:apply-templates/>
 </OMA>
</xsl:template>
```

```
<xsl:template match="sin">
  <OMS cd = "transc" name = "sin"/>
</xsl:template>

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

However, the stylesheet can be also written in the second way of "selection":

– template rules –

```
<xsl:template match="apply">
  <OMA>
    <xsl:apply-templates select="*[first-of-any()]"/>
  </OMA>
</xsl:template>

<xsl:template match="sin">
 <OMS cd = "transc" name = "sin"/>
  <xsl:apply-templates select="../*[not(first-of-any()]"/>
</xsl:template>

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

where the rule for apply just picks off the first child and the process instruction from the rule for sin directs the processor to process *cn*.

Which way should we choose? Well, both are adequate if they are used alone. However, there are many mathematical objects, which do have different encoding structures in MathML and OpenMath. For them, we can only use the second way, that is, use the rule for the first child to control the process for other children.

When these two ways are mixed in one stylesheet, conflicts will result in wrong results. So our first decision is to choose the second method.

2. For the elements "forall" and "exists", there is a big difference between MathML and OpenMath encoding structures. In OpenMath, these two quantifiers take only two arguments. The first argument is the bound variable (placed within OMBVAR), and the second is an expression. In MathML they may also have an optional condition argument, in addition to the same two arguments as in OpenMath. Our treatment when the condition argument appears in MathML is to transform it to be within the second argument in OpenMath, which is then a tuple with a combination of condition and expression. See Quant.xsl in the Appendix.

Similar situations appear in "list" and "set" in the opposite direction. In OpenMath, three arguments are needed: the first is a bound variable, the second is a range for this variable, and the third is an expression to be evaluated over this range. In MathML, the third argument is optional if the expression is the variable itself. Fortunately, XSL allows us to repeat processing any part in the source tree. So when this happens we can process that variable a second time as the expression in OpenMath. See List.xsl and Set.xsl in the Appendix.

3. For elements "partialdiff" in MathML and "diff" in OpenMath, the variables and the degree are structured in different ways:

– MathML –

```
<apply>
  <partialdiff/>
  <bvar>
    <ci>x</ci>
    <degree>2</degree>
  </bvar>
  <bvar>
    <ci>y</ci>
  </bvar>
  ...
```

– OpenMath –

```
<OMA>
  <OMS cd="calculus" name="diff"/>
  <OMA>
    <OMS cd="list" name="list"
    <OMV name="x"/>
    <OMV name="y"/>
  </OMA>
  <OMA>
    <OMS cd="list" name="list"/>
    <OMI>2</OMI>
    <OMI>1</OMI>
  </OMA>
   ...
```

XSL is sufficiently powerful to allow us to change the order. To perform this transformation, the template rule for partialdiff contains

```
...
<OMA>
```

```
 <OMS cd="list" name="list"/>
  <xsl:apply-templates select="../bvar/ci"/>
 </OMA>
 <OMA>
  <OMS cd="list" name="list"/>
  <xsl:apply-templates select="../bvar"/>
 </OMA>
...
```

and the template rule for **bvar** contains

```
...
<xsl:choose>
 <xsl:when test="degree">
  <xsl:apply-templates/>
 </xsl:when>
 <xsl:otherwise>
  <OMI>1</OMI>
 </xsl:otherwise>
</xsl:choose>
...
```

7.9 Extension of XSL

Here we summarize the extensions we made to XSL to allow us to write the stylesheets for the set of OpenMath CDs.

pattern extensions

1. fromto(n,m), fromto(n)
XSL supplies position qualifiers to test the position of a node relative to siblings:

first-of-any()
last-of-any()
first-of-type()
last-of-type()
not()

However these qualifiers and their combinations can not be used to select some range of siblings, for example from third to last or third to fifth. For our need, we add a qualifier

fromto(n,m) and fromto(n)

where $n \leq m \leq$ total (type or any) with any violation of the above inequality yielding the empty set.

These function-like qualifiers select siblings from n-th to m-th or to the final one

An example using these qualifiers can be found in Sumprod.xsl of the Appendix.

2. next()

We have mentioned in section 6.4 that an extension of next() is needed in order to select the next sibling of the current node. If the current node is the last sibling, this selection will return null. See Example 6.2.

3. Match on the number of children

There are some cases (e.g. integral, summation, product and limit) where unary functions and expressions are dealt with in different ways in OpenMath:

For example, in OpenMath, **int** is used in this way:

Indefinite integration of unary functions takes only one argument: the unary function. e.g.:

```
<OMOBJ>
  <OMA>
    <OMS cd="calculus" name="int"/>
    <OMS cd="transc" name="sin"/>
  </OMA>
</OMOBJ>
```

Indefinite integration of expressions takes two arguments, the first is the variable of integration, and the second the expression. For example:

```
<OMOBJ>
  <OMA>
    <OMS cd="calculus" name="int"/>
    <OMV name="x"/>
    <OMA>
      <OMS cd="arith" name="times"/>
      <OMV name="x"/>
      <OMV name="y"/>
    </OMA>
```

```
  </OMA>
</OMOBJ>
```

However, MathML content markup does not use different structures to distinguish this difference:

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

<apply>
  <int/>
  <apply>
    <times/>
    <ci>x</ci>
    <ci>y</ci>
  </apply>
</apply>
```

and so we have to enumerate **all** the unary functions in the select pattern in the template for "int" and then initialize different parts according to the test result.

```
<xsl:template match = "int">
  ...
<xsl:when test = "../apply[sin or cos or ln]">
  ...
<xsl:otherwise>
  ...
</xsl:template>
```

Since **apply** has 2 children in the unary functions case, and has more than 2 children in the expression case, we can add a match on the number of children #n and make the pattern simpler:

```
<xsl:when test ="../apply[#2]">
```

7.10 More Examples

This section gives some more complex examples

Example 7.5 (macro for expanding abbreviation)

— **xml-in** —

```
<?XML version = "1.0" ?>
<math>
  <reln>
    <eq/>
    <apply>
     <mtrx-inverse/>
     <apply>
       <id-mtrx/>
       <cn>n</cn>
     </apply>
    </apply>
    <apply>
      <id-mtrx/>
      <cn>n</cn>
    </apply>
  </reln>

  <reln>
    <eq/>
    <det>
      <apply>
        <mtrx-inverse/>
        <ci>A</ci>
      </apply>
    </det>
    <det>A</det>
 </reln>
</math>
```

— **xml-xsl** —

```
<?XML version = "1.0" ?>
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WD-xsl" >
```

```
<xsl:template match = "mtrx-inverse" >
  <fn>
    <semantics>
      <ci><mo>mtrx-inverse</mo></ci>
      <annotation-xml encoding = " OpenMath ">
        <OMS cd="linalg" name="matrix-inverse"/>
      </annotation-xml>
    </semantics>
  </fn>
</xsl:template>

<xsl:template match = "id-mtrx" >
  <fn>
    <semantics>
      <ci><mo>id-mtrx</mo></ci>
      <annotation-xml encoding = " OpenMath ">
        <OMS cd="linalg" name="identity-matrix"/>
      </annotation-xml>
    </semantics>
  </fn>
</xsl:template>

<xsl:template match = "det" >
  <apply>
    <determinant/>
    <ci type = "matrix">
      <xsl:apply-templates />
    </ci>
  </apply>
</xsl:template>

<xsl:template match = "/">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match = "*">
  <xsl:copy>
   <xsl:apply-templates/>
  </xsl:copy>
```

```
</xsl:template>

</xsl:stylesheet>
```

— xml-out —

```
<?XML version = "1.0" ?>
<reln>
  <eq/>
  <apply>
    <fn>
      <semantics>
        <ci><mo>matrix-inverse</mo></ci>
        <annotation-xml encoding = " OpenMath ">
          <OMS cd="linalg" name="matrix-inverse"/>
        </annotation-xml>
      </semantics>
    </fn>
    <apply>
      <fn>
        <semantics>
          <ci><mo>id-matrix</mo></ci>
          <annotation-xml encoding = " OpenMath ">
            <OMS cd="linalg" name="identity-matrix"/>
          </annotation-xml>
        </semantics>
      </fn>
      <cn>n</cn>
    </apply>
  </apply>
  <apply>
    <fn>
      <semantics>
        <ci><mo>id-matrix</mo></ci>
        <annotation-xml encoding = " OpenMath ">
          <OMS cd="linalg" name="identity-matrix"/>
        </annotation-xml>
      </semantics>
    </fn>
    <cn>n</cn>
```

```
    </apply>
  </reln>

  <reln>
    <eq/>
    <apply>
      <determinant/>
      <apply>
        <fn>
          <semantics>
            <ci><mo>matrix-inverse</mo></ci>
            <annotation-xml encoding = " OpenMath ">
              <OMS cd="linalg" name="matrix-inverse"/>
            </annotation-xml>
          </semantics>
        </fn>
        <ci type = "matrix"> A </ci>
      </apply>
    </apply>
    <apply>
      <determinant/>
      <ci type = "matrix"> A </ci>
    </apply>
  </reln>
```

Example 7.6 (combined markup)

— xml-in —

```
<math>
  <apply>
    <power/>
    <apply>
      <sin/>
      <ci>x</ci>
    </apply>
    <cn>2</cn>
  </apply>
</math>
```

— xml-xsl-1 —

```
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WO-xsl">

<xsl:template match = "apply">
  <xsl:apply-templates select = "*[first-of-any()]"/>
</xsl:template>

<xsl:template match = "apply/power">
 <mrow>
  <msup>
   <xsl:apply-templates select="ancestor(apply)/*[not(first-of-any())]"/>
  </msup>
  <xsl:if test = "ancestor(apply)/apply/sin">
    <mo>&ApplyFunction;</mo>
    <xsl:apply-templates select = "ancestor(apply)/ci"/>
  </xsl:if>
 </mrow>
</xsl:template>

<xsl:template match = "sin">
  <mi>sin</mi>
</xsl:template>

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

<xsl:template match = "ci">
  <mi>
   <xsl:apply-templates/>
  </mi>
</xsl:template>

<xsl:template match = "/">
  <xsl:apply-templates/>
</xsl:template>

</xsl:stylesheet>
```

— xml-xsl-2 —

```
<?XML version = "1.0" ?>
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WO-xsl">

<xsl:template match = "apply">
  <OMA>
   <xsl:apply-templates/>
  </OMA>
</xsl:template>

<xsl:template match = "power">
  <OMS cd = "Basic" name = "power"/>
</xsl:template>

<xsl:template match = "sin">
  <OMS cd="transc" name="sin"/>
</xsl:template>

<xsl:template match = "cn">
  <OMI>
    <xsl:apply-templates/>
  </OMI>
</xsl:template>

<xsl:template match = "ci">
  <OMV name={.}/>
</xsl:template>

<xsl:template match = "/">
  <annotation-xml encoding = "OpenMath">
   <xsl:apply-templates/>
  </annotation-xml>
</xsl:template>

</xsl:stylesheet>
```

— xml-out —

```
<semantics>
```

```
    <mrow>
      <msup>
        <mi>sin</mi>
        <mn>2</mn>
      </msup>
      <mo> &ApplyFunction;</mo>
      <mi>x</mi>
    </mrow>
    <annotation-xml encoding="OpenMath">
      <OMA>
       <OMS cd = "arith" name = "power"/>
       <OMA>
         <OMS cd = "transc" name = "sin"/>
         <OMV name="x"/>
       </OMA>
      </OMA>
    </annotation-xml>
  </semantics>
```

7.11 Future Work

In our realization of macros for combined markup, we have considered so far the combination of presentation markup and content markup at the root only. Generally, it is preferable to have combined markup at each node so that the selection of subexpressions retains semantic meaning. To realize this case, multiple passes are needed. We discuss a possible approach here, but leave implementation for future work.

Consider an example: to markup the expression

$$\sin\tan(x+y) - \tan\sin(x-y)$$

with both presentation markup from MathML and semantic markup from OpenMath at each node. The result tree is shown in figure 7.1. It is a relatively huge document, which would be obtained from the source document:

```
      <apply>
        <minus/>
        <apply>
          <sin/>
          <apply>
```

```
        <tan/>
        <apply>
          <plus/><ci>x</ci><ci>y</ci>
        </apply>
      </apply>
    </apply>
    <apply>
      <tan/>
      <apply>
        <sin/>
        <apply>
          <minus/><ci>x</ci><ci>y</ci>
        </apply>
      </apply>
    </apply>
  </apply>
```

One could obtain the result tree as follows: We transform the source document to the corresponding presentation markup, at each code, attached with the entire subtree from the source. These subtrees supply the information for a further pass to obtain the semantic meaning from OpenMath for each node.

Now we consider the intermediate file. Conceptually, we can use an "attachment" tree to represent this file (See Figure 7.2), where each rectangle represents the corresponding entire subtree.

Clearly, the attachment of the entire subtree takes a lot of storage space. We claim that if the source tree has n nodes, after the first pass stated above, the number of nodes in the "attachment" tree will be $O(n^2)$.

We explain the idea of the proof here:

If the tree is binary tree, the depth of the tree is roughly $\log_2(n)$. At each depth i, there are 2^i nodes; for each node, the attached entire subtree has $\sum_{j=i}^{\log n} 2^j$ and therefore the number of total nodes is $\sum_{i=0}^{\log n} 2^i \sum_{j=i}^{\log n} 2^j$, which is $O(n^2)$.

Since the intermediate document must be traversed in a further pass, we would have a time complexity of at least this order.

To achieve both efficiency and storage saving, we need to reduce $O(n^2)$ to $O(n)$.

From our experience in writing stylesheets for the transformation from MathML content markup to presentation markup, as well as OpenMath semantic meaning, we found that in most cases, the selection of which transformation to apply is based on a limited depth match against each tree node. Suppose that for an entire set of

stylesheets, the maximal depth required for matching is h. Then the initial transformation can place on each node a clipped control subtree which retains only the top h levels. This bounds the expansion at each node, and all intermediate documents are then of size $O(n)$.

A suitable value of h can be determined by examining the stylesheets. How to best write the stylesheets so as to minimize h without making any loss of the semantic meaning is a request for future work.

Figure 7.1: result tree

```
<semantics>
  <mrow>
    <semantics>
      <mrow>
        <mi>sin</mi>
        <mo>&ApplyFunction;</mo>
        <sematics>
          <mrow>
            <mi>tan</mi>
            <mo>&ApplyFunction;</mo>
            <semantics>
              <mfence>
                <mi>x</mi><mo>plus</mo><mi>y</mi>
              </mfence>
              <annotation-xml>
                <OMA>
                  <OMS cd ="arith" name="plus"/>
                  <OMV name="x"/>
                  <OMV name="y"/>
                </OMA>
              </annotation-xml>
            </semantics>
          </mrow>
          <annotation-xml>
            <OMA>
              <OMS cd = "transc" name="tan"/>
              <OMA>
```

```
              <OMS cd ="arith" name="plus"/>
              <OMV name="x"/>
              <OMV name="y"/>
            </OMA>
          </OMA>
        </annotation-xml>
      </semantics>
    </mrow>
    <annotation-xml>
      <OMA>
        <OMS cd = "transc" name="sin"/>
        <OMA>
          <OMS cd = "transc" name="tan"/>
          <OMA>
            <OMS cd ="arith" name="plus"/>
            <OMV name="x"/>
            <OMV name="y"/>
          </OMA>
        </OMA>
      </OMA>
    </annotation-sml>
  </semantics>
  <mo>minus</mo>
  <semantics>
    <mrow>
      <mi>tan</mi>
      <mo>&ApplyFunction;</mo>
      <semantics>
        <mrow>
          <mi>sin</mi>
          <mo>&ApplyFunction;</mo>
          <semantics>
            <mfence>
              <mi>x</mi><mo>minus</mo><mi>y</mi>
            </mfence>
            <annotation-xml>
              <OMA>
                <OMS cd ="arith" name="minus"/>
                <OMV name="x"/>
```

```
                <OMV name="y"/>
              </OMA>
            </annotation-xml>
          </semantics>
        </mrow>
        <annotation-xml>
          <OMA>
            <OMS cd = "transc" name="sin"/>
            <OMA>
              <OMS cd ="arith" name="minus"/>
              <OMV name="x"/>
              <OMV name="y"/>
            </OMA>
          </OMA>
        </annotation-xml>
      </semantics>
    </mrow>
    <annotation-xml>
      <OMA>
        <OMS cd = "transc" name="tan"/>
        <OMA>
          <OMS cd = "transc" name="sin"/>
          <OMA>
            <OMS cd ="arith" name="minus"/>
            <OMV name="x"/>
            <OMV name="y"/>
          </OMA>
        </OMA>
      </OMA>
    </annotation-sml>
  </semantics>
</mrow>
<annotation-xml encoding ="OpenMath">
  <OMA>
    <OMS cd = "arith" name="minus"/>
    <OMA>
      <OMS cd = "transc" name="sin"/>
      <OMA>
        <OMS cd = "transc" name="tan"/>
```

```
          <OMA>
            <OMS cd ="arith" name="plus"/>
            <OMV name="x"/>
            <OMV name="y"/>
          </OMA>
        </OMA>
      </OMA>
      <OMA>
        <OMS cd = "transc" name="tan"/>
        <OMA>
          <OMS cd = "transc" name="sin"/>
          <OMA>
            <OMS cd ="arith" name="minus"/>
            <OMV name="x"/>
            <OMV name="y"/>
          </OMA>
        </OMA>
      </OMA>
    </OMA>
  </annotation-xml>
</semantics>
```

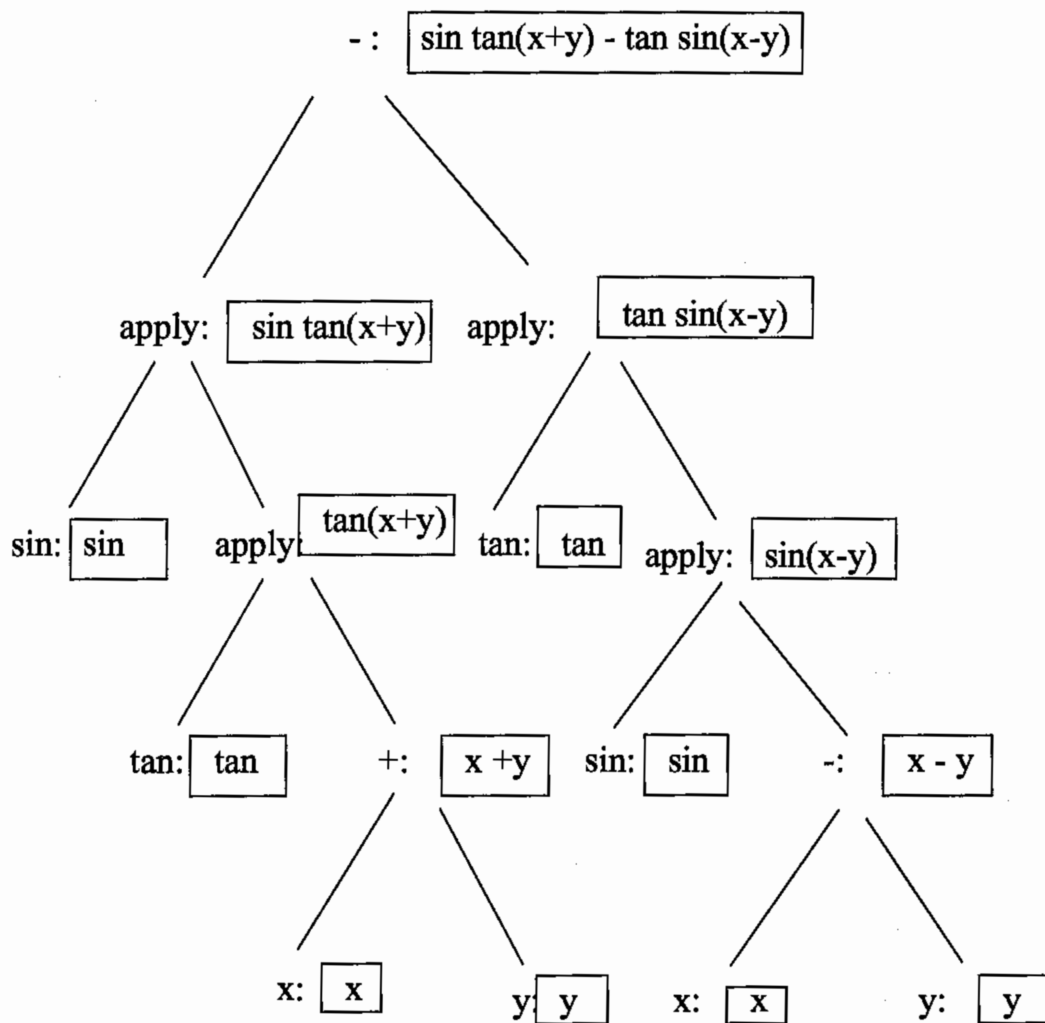


Figure 7.2: “attachment” tree

Chapter 8

Conclusion

OpenMath and MathML are playing increasingly important roles in the communication of mathematical objects. Each of them have a different emphasis. OpenMath is primarily for semantic meaning. MathML is primarily for presentation. They are complementary.

One of the most significant gaps is the lack of a macro mechanism for MathML to handle abbreviation and to abstract new concepts, especially when MathML needs to use semantic meanings from OpenMath. Therefore, it is necessary to develop a macro mechanism for MathML.

A prototype of such a macro mechanism for MathML has been developed in this thesis. We have experimented many examples, and the result shows the efficiency and power of this macro mechanism. On the basis of these experiments, we conclude that XSL draft recommendation of December 16, 1998, with minor extensions, could be suitable basis for mathematical macro processing. The extensions are detailed in Section 7.9.

We believe that the prototype we presented here is a meaningful step in the development of mathematical macro processing. The stylesheets we have written can provide meaningful guidance for a future library for a macro mechanism.

Appendix A

Stylesheets for OpenMath CDs

A.1 Limit.xsl

```
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WD-xsl">

<xsl:template match = "apply">
 <xsl:choose>
  <xsl:when test="apply[#2]">
   <OMA>
    <xsl:apply-templates select="*[first-of-any()]"/>
   </OMA>
  </xsl:when>
  <xsl:otherwise>
   <xsl:apply-templates select="*[first-of-any()]"/>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>

<xsl:template match = "apply/apply">
 <xsl:choose>
  <xsl:when test=".[#2]">
   <xsl:apply-templates select="*[first-of-any()]"/>
  </xsl:when>
  <xsl:otherwise>
   <OMA>
    <xsl:apply-templates select="*[first-of-any()]"/>
   </OMA>
```

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

<xsl:template match = "limit">
 <xsl:choose>
  <xsl:when test="../apply[#2]">
   <OMS cd = "limit" name = "limit"/>
   <xsl:choose>
    <xsl:when test="../lowlimit">
     <xsl:apply-templates select="../lowlimit/cn"/>
     <OMS cd="limit" name="above"/>
    </xsl:when>
    <xsl:when test="../condition">
     <xsl:apply-templates select="../condition/reln/tendsto"/>
    </xsl:when>
   </xsl:choose>
   <xsl:apply-templates select="../apply"/>
  </xsl:when>

  <xsl:otherwise>
   <OMBIND>
    <OMS cd="limit" name="limit"/>
    <OMBVAR>
     <xsl:apply-templates select="../bvar/ci"/>
    </OMBVAR>
    <OMA>
     <OMS cd="basic" name="tuple"/>
     <OMA>
      <xsl:choose>
       <xsl:when test="../lowlimit">
        <OMS cd="limit" name="tendsto"/>
        <xsl:apply-templates select="../bvar/ci"/>
        <xsl:apply-templates select="../lowlimit/cn"/>
        <OMS cd="limit" name="above"/>
       </xsl:when>
       <xsl:when test="../condition">
        <xsl:apply-templates select="../condition/reln/tendsto"/>
       </xsl:when>
```

```
        </xsl:choose>
       </OMA>
       <xsl:apply-templates select="../apply"/>
      </OMA>
     </OMBIND>
    </xsl:otherwise>
   </xsl:choose>
  </xsl:template>

  <xsl:template match = "tendsto[@type='above']">
   <xsl:choose>
    <xsl:when test="ancestor(apply)/apply[#2]">
     <xsl:apply-templates select="../*[last-of-any()]"/>
    </xsl:when>
    <xsl:otherwise>
     <OMS cd = "limit" name = "tendsto"/>
     <xsl:apply-templates select="../*[not(first-of-any())]"/>
    </xsl:otherwise>
   </xsl:choose>
   <OMS cd = "limit" name = "above"/>
  </xsl:template>

  <xsl:template match = "tendsto[@type='below']">
   <xsl:choose>
    <xsl:when test="ancestor(apply)/apply[#2]">
     <xsl:apply-templates select="../*[last-of-any()]"/>
    </xsl:when>
    <xsl:otherwise>
     <OMS cd = "limit" name = "tendsto"/>
     <xsl:apply-templates select="../*[not(first-of-any())]"/>
    </xsl:otherwise>
   </xsl:choose>
   <OMS cd = "limit" name = "below"/>
  </xsl:template>

  <xsl:template match = "tendsto[@type='two-sided']">
   <xsl:choose>
    <xsl:when test="ancestor(apply)/apply[#2]">
     <xsl:apply-templates select="../*[last-of-any()]"/>
```

```
    </xsl:when>
    <xsl:otherwise>
     <OMS cd = "limit" name = "tendsto"/>
     <xsl:apply-templates select="../*[not(first-of-any())]"/>
    </xsl:otherwise>
  </xsl:choose>
  <OMS cd = "limit" name = "both-sides"/>
 </xsl:template>

 <xsl:template match = "tendsto">
  <xsl:choose>
   <xsl:when test="ancestor(apply)/apply[#2]">
    <xsl:apply-templates select="../*[last-of-any()]"/>
   </xsl:when>
   <xsl:otherwise>
    <OMS cd = "limit" name = "tendsto"/>
    <xsl:apply-templates select="../*[not(first-of-any())]"/>
   </xsl:otherwise>
  </xsl:choose>
  <OMS cd = "limit" name = "above"/>
 </xsl:template>

 </xsl:stylesheet>
```

A.2 Sumprod.xsl

```
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WD-xsl">

<xsl:template match = "apply">
 <xsl:choose>
  <xsl:when test="apply[#2]">
   <OMA>
    <xsl:apply-templates select="*[first-of-any()]"/>
   </OMA>
  </xsl:when>
  <xsl:otherwise>
   <xsl:apply-templates select="*[first-of-any()]"/>
  </xsl:otherwise>
 </xsl:choose>
```

```
</xsl:template>

<xsl:template match = "apply/apply">
 <xsl:choose>
  <xsl:when test=".[#2]">
   <xsl:apply-templates select="*[first-of-any()]"/>
  </xsl:when>
  <xsl:otherwise>
   <OMA>
    <xsl:apply-templates select="*[first-of-any()]"/>
   </OMA>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>

<xsl:template match = "sum">
 <xsl:choose>
  <xsl:when test="../apply[#2]">
   <OMS cd = "sumprod" name = "sum"/>
   <xsl:choose>
    <xsl:when test="../lowlimit">
     <OMA>
      <OMS cd = "interval" name="discrete-interval"/>
      <xsl:apply-templates select="../*[fromto(3,4)]"/>
     </OMA>
     <xsl:apply-templates select="../apply[#2]"/>
    </xsl:when>
    <xsl:when test="../condition">
     <xsl:apply-templates select="../condition/reln"/>
    </xsl:when>
   </xsl:choose>
  </xsl:when>
  <xsl:otherwise>
   <OMBIND>
    <OMS cd="sumprod" name="sum"/>
    <OMBVAR>
     <xsl:apply-templates select="../bvar/ci"/>
    </OMBVAR>
    <OMA>
```

```
      <OMS cd="basic" name="tuple"/>
       <xsl:choose>
        <xsl:when test="../lowlimit">
         <OMA>
          <OMS cd="inerval" name="discrete-interval"/>
          <xsl:apply-templates select="../*[fromto(3,4)]"/>
         </OMA>
        </xsl:when>
        <xsl:when test="../condition">
         <xsl:apply-templates select="../condition"/>
        </xsl:when>
       </xsl:choose>
       <xsl:apply-templates select="../apply"/>
      </OMA>
     </OMBIND>
    </xsl:otherwise>
   </xsl:choose>
  </xsl:template>


  <xsl:template match = "product">
  <xsl:choose>
   <xsl:when test="../apply[#2]">
    <OMS cd = "sumprod" name = "product"/>
    <xsl:choose>
     <xsl:when test="../lowlimit">
     <OMA>
       <OMS cd = "interval" name="discrete-interval"/>
       <xsl:apply-templates select="../*[fromto(3,4)]"/>
     </OMA>
     </xsl:when>
     <xsl:when test="../condition">
      <xsl:apply-templates select="../condition/reln"/>
     </xsl:when>
    </xsl:choose>
    <xsl:apply-templates select="../apply[#2]"/>
   </xsl:when>

   <xsl:otherwise>
```

```
<OMBIND>
 <OMS cd="sumprod" name="product"/>
 <OMBVAR>
  <xsl:apply-templates select="../bvar/ci"/>
 </OMBVAR>
 <OMA>
  <OMS cd="basic" name="tuple"/>
   <xsl:choose>
    <xsl:when test="../lowlimit">
     <OMA>
      <OMS cd="inerval" name="discrete-interval"/>
      <xsl:apply-templates select="../*[fromto(3,4)]"/>
     </OMA>
    </xsl:when>
    <xsl:when test="../condition">
     <xsl:apply-templates select="../condition"/>
    </xsl:when>
   </xsl:choose>
   <xsl:apply-templates select="../apply"/>
  </OMA>
 </OMBIND>
</xsl:otherwise>
</xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

A.3 Calculus.xsl

```
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WD-xsl">

<xsl:template match = "apply">
 <OMA>
  <xsl:apply-templates select="*[first-of-any()]"/>
 </OMA>
</xsl:template>

<xsl:template match = "apply/apply[#2]">
```

```
  <xsl:apply-templates select="*[first-of-any()]"/>
 </xsl:template>

 <xsl:template match = "partialdiff|diff">
  <OMS cd = "calculus" name = "diff"/>
  <OMA>
   <OMS cd="list" name="list"/>
   <xsl:apply-templates select="../bvar/ci"/>
  </OMA>
  <OMA>
   <OMS cd="list" name="list"/>
   <xsl:apply-templates select="../bvar"/>
  </OMA>
  <xsl:apply-templates select="../*[last-of-any()]"/>
</xsl:template>

<xsl:template match = "int">
 <xsl:choose>
  <xsl:when test ="../apply[#2]">
   <xsl:choose>
    <xsl:when test = "../bvar">
     <OMS cd = "calculus" name = "int"/>
     <xsl:choose>
      <xsl:when test ="../lowlimit">
       <OMA>
        <OMS cd="interval" name="continuous-interval"/>
        <xsl:apply-templates select="../*[fromto(3,4)]"/>
       </OMA>
       <xsl:apply-templates select="../*[last-of-any()]"/>
      </xsl:when>
      <xsl:when test="../condition | ../interval">
       <xsl:apply-templates select="../condition|../interval"/>
       <xsl:apply-templates seclect="../*[last-of-any()]"/>
      </xsl:when>
      <xsl:otherwise>
       <xsl:apply-templates seclect="../*[last-of-any()]"/>
      </xsl:otherwise>
     </xsl:choose>
    </xsl:when>
```

```
     <xsl:otherwise>
      <OMS cd = "calculus" name = "int"/>
      <xsl:apply-templates select="../apply[#2]"/>
     </xsl:otherwise>
    </xsl:choose>
   </xsl:when>
   <xsl:otherwise>
    <xsl:choose>
     <xsl:when test = "../bvar">
      <OMBIND>
       <OMS cd = "calculus" name = "int"/>
       <OMBVAR>
        <xsl:apply-templates select="../bvar/ci"/>
       </OMBVAR>
       <xsl:choose>
        <xsl:when test="../lowlimit">
         <OMA>
          <OMS cd = "basic" name="tuple"/>
          <OMA>
           <OMS cd="interval" name="continuous-interval"/>
           <xsl:apply-templates select="../*[fromto(3,4)]"/>
          </OMA>
          <xsl:apply-templates select="../*[last-of-any()]"/>
         </OMA>
        </xsl:when>
        <xsl:when test="../condition| ../interval">
         <OMA>
          <OMS cd = "basic" name="tuple"/>
          <xsl:apply-templates select="../condition| ../interval"/>
          <xsl:apply-templates select="../*[last-of-any()]"/>
         </OMA>
        </xsl:when>
        <xsl:otherwise>
         <xsl:apply-templates seclect="../*[last-of-any()]"/>
        </xsl:otherwise>
       </xsl:choose>
      </OMBIND>
     </xsl:when>
     <xsl:otherwise>
```

```
     <OMS cd = "calculus" name = "int"/>
     <xsl:apply-templates select="..//ci"/>
     <xsl:apply-templates select="../*[not(first-of-any())]"/>
    </xsl:otherwise>
   </xsl:choose>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>

<xsl:template match = "interval">
 <OMA>
  <OMS cd = "interval" name = "continuous-interval-cl-cl"/>
  <xsl:apply-templates/>
 </OMA>
</xsl:template>

<xsl:template match = "reln">
 <OMA>
  <xsl:apply-templates select="*[first-of-any()]"/>
 </OMA>
</xsl:template>

<xsl:template match = "in">
 <OMS cd="set" name="in"/>
 <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match ="bvar">
 <xsl:choose>
  <xsl:when test="degree">
   <xsl:apply-templates select="degree"/>
  </xsl:when>
  <xsl:otherwise>
   <OMI>1</OMI>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>

<xsl:template match = "condition|lowlimit|uplimit|degree">
```

```
  <xsl:apply-templates/>
 </xsl:template>

 </xsl:stylesheet>
```

A.4 Transc.xsl

```
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WD-xsl">

<xsl:template match = "ln">
 <OMS cd = "transc" name = "ln"/>
 <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "log">
 <OMS cd = "transc" name = "log"/>
 <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "exp">
 <OMS cd = "transc" name = "exp"/>
 <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "sin">
 <OMS cd = "transc" name = "sin"/>
 <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "cos">
 <OMS cd = "transc" name = "cos"/>
 <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "tan">
 <OMS cd = "transc" name = "tan"/>
 <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>
```

```
<xsl:template match = "sec">
 <OMS cd = "transc" name = "sec"/>
 <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "csc">
 <OMS cd = "transc" name = "csc"/>
 <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "cot">
 <OMS cd = "transc" name = "cot"/>
 <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "sinh">
 <OMS cd = "transc" name = "sinh"/>
 <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "cosh">
 <OMS cd = "transc" name = "cosh"/>
 <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "tanh">
 <OMS cd = "transc" name = "tanh"/>
 <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "sech">
 <OMS cd = "transc" name = "sech"/>
 <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "csch">
 <OMS cd = "transc" name = "csch"/>
 <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>
```

```
<xsl:template match = "coth">
 <OMS cd = "transc" name = "coth"/>
 <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "arcsin">
 <OMS cd = "transc" name = "arcsin"/>
 <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "arccos">
 <OMS cd = "transc" name = "arccos"/>
 <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "arctan">
 <OMS cd = "transc" name = "arctan"/>
 <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

</xsl:stylesheet>
```

A.5 Quant.xsl

```
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WD-xsl">

<xsl:template match = "apply">
 <OMBIND>
  <xsl:apply-templates select="*[first-of-any()]"/>
 </OMBIND>
</xsl:template>

<xsl:template match = "forall">
 <OMS cd = "quant" name = "forall"/>
 <OMBVAR>
  <xsl:apply-templates select="../bvar/ci"/>
 </OMBVAR>
 <xsl:choose>
```

```
  <xsl:when test="../condition">
   <OMA>
    <OMS cd="basic" name="tuple"/>
    <xsl:apply-templates select="../condition"/>
    <xsl:apply-templates select="../*[last-of-any()]"/>
   </OMA>
  </xsl:when>
  <xsl:otherwise>
   <xsl:apply-templates select="../*[last-of-any()]"/>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>

<xsl:template match = "exist">
 <OMS cd = "quant" name = "exist"/>
 <OMBVAR>
  <xsl:apply-templates select="../bvar/ci"/>
 </OMBVAR>
 <xsl:choose>
  <xsl:when test="../condition">
   <OMA>
    <OMS cd="basic" name="tuple"/>
    <xsl:apply-templates select="../condition"/>
    <xsl:apply-templates select="../*[last-of-any()]"/>
   </OMA>
  </xsl:when>
  <xsl:otherwise>
   <xsl:apply-templates select="../*[last-of-any()]"/>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

A.6 Interval.xsl

```
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WD-xsl">

<xsl:template match = "apply">
```

```
<OMA>
<xsl:apply-templates/>
</OMA>
</xsl:template>

<xsl:template match = "interval[@closure='open']">
<OMA>
<OMS cd = "interval" name = "continuous-interval-op-op"/>
<xsl:apply-templates/>
</OMA>
</xsl:template>

<xsl:template match = "interval[@closure='closed']">
<OMA>
<OMS cd = "interval" name = "continuous-interval-cl-cl"/>
<xsl:apply-templates/>
</OMA>
</xsl:template>

<xsl:template match = "interval[@closure='open-closed']">
<OMA>
<OMS cd = "interval" name = "continuous-interval-op-cl"/>
<xsl:apply-templates/>
</OMA>
</xsl:template>

<xsl:template match = "interval[@closure='closed-open']">
<OMA>
<OMS cd = "interval" name = "continuous-interval-cl-op"/>
<xsl:apply-templates/>
</OMA>
</xsl:template>

<xsl:template match = "interval">
<OMA>
<OMS cd = "interval" name = "continuous-interval-cl-cl"/>
<xsl:apply-templates/>
</OMA>
</xsl:template>
```

```
</xsl:stylesheet>
```

A.7 Arith.xsl

```
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WD-xsl">

<xsl:template match = "apply">
<OMA>
<xsl:apply-templates select="*[first-of-any()]"/>
</OMA>
</xsl:template>

<xsl:template match = "power">
<OMS cd = "arith" name = "power"/>
<xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "minus">
<OMS cd = "arith" name = "minus"/>
<xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "plus">
<OMS cd = "arith" name = "plus"/>
<xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "times">
<OMS cd = "arith" name = "times"/>
<xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "devide">
<OMS cd = "arith" name = "devide"/>
<xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "abs">
```

```
<OMS cd = "arith" name = "abs"/>
<xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "root">
<OMS cd = "arith" name = "root"/>
<xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>


<xsl:template match = "conjugate">
<OMS cd = "arith" name = "conjugate"/>
<xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

</xsl:stylesheet>
```

A.8 List.xsl

```
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WD-xsl">

<xsl:template match = "list">
<OMA>
<OMS cd = "list" name = "list"/>
<xsl:apply-templates/>
</OMA>
</xsl:template>

<xsl:template match = "list/ci">
<OMSTR>
<xsl:apply-templates/>
</OMSTR>
</xsl:template>

<xsl:template match = "list[bvar]">
<OMBIND>
 <OMS cd = "list" name = "list"/>
 <xsl:apply-templates select="bvar"/>
 <OMA>
```

```
    <OMS cd = "basic" name="tuple"/>
    <xsl:apply-templates select="condition"/>
    <xsl:choose>
      <xsl:when test="apply">
        <xsl:apply-templates select="apply"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:apply-templates select="bvar"/>
      </xsl:otherwise>
    </xsl:choose>
  </OMA>
</OMBIND>
</xsl:template>

</xsl:stylesheet>
```

A.9 Logic.xsl

```
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WD-xsl">

<xsl:template match = "not">
<OMS cd = "logic" name = "not"/>
<xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "and">
<OMS cd = "logic" name = "and"/>
<xsl:apply-templates select ="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "or">
<OMS cd = "logic" name = "or"/>
<xsl:apply-templates select ="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "xor">
<OMS cd = "logic" name = "xor"/>
<xsl:apply-templates select ="../*[not(first-of-any())]"/>
</xsl:template>
```

```
<xsl:template match = "implies">
<OMS cd = "logic" name = "implies"/>
<xsl:apply-templates select ="../*[not(first-of-any())]"/>
</xsl:template>

</xsl:stylesheet>
```

A.10 Integer.xsl

```
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WD-xsl">

<xsl:template match = "gcd">
<OMS cd = "integer" name = "gcd"/>
</xsl:template>

<xsl:template match = "factorial">
<OMS cd = "integer" name = "factorial"/>
</xsl:template>

<xsl:template match = "quotient">
<OMS cd = "integer" name = "quotient"/>
</xsl:template>

<xsl:template match = "rem">
<OMS cd = "integer" name = "rem"/>
</xsl:template>

</xsl:stylesheet>
```

A.11 Set.xsl

```
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WD-xsl">

<xsl:template match = "set">
<OMA>
  <OMS cd = "set" name = "set"/>
  <xsl:apply-templates/>
</OMA>
```

```
</xsl:template>

<xsl:template match = "set[bvar]">
<OMA>
  <OMS cd = "set" name = "set"/>
  <OMBIND>
    <OMS cd = "basic" name="elements"/>
    <xsl:apply-templates select="bvar"/>
    <OMA>
      <OMS cd ="basic" name="tuple"/>
      <xsl:apply-templates select="condition"/>
      <xsl:choose>
        <xsl:when test="apply">
        <xsl:apply-templates select="../apply"/>
        </xsl:when>
        <xsl:otherwise>
        <xsl:apply-templates select="bvar"/>
        </xsl:otherwise>
      </xsl:choose>
    </OMA>
  </OMBIND>
</OMA>
</xsl:template>

<xsl:template match = "union">
  <OMS cd = "set" name = "union"/>
  <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "intersect">
  <OMS cd = "set" name = "intersect"/>
  <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "setdiff">
  <OMS cd = "set" name = "setdiff"/>
  <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>
```

```
<xsl:template match = "subset">
  <OMS cd = "set" name = "subset"/>
  <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "prsubset">
  <OMS cd = "set" name = "prsubset"/>
  <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "notsubset">
  <OMS cd = "set" name = "notsubset"/>
  <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "notprsubset">
  <OMS cd = "set" name = "notprsubset"/>
  <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "in">
  <OMS cd = "set" name = "in"/>
  <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "notin">
  <OMS cd = "set" name = "notin"/>
  <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

</xsl:stylesheet>
```

A.12 Relation.xsl

```
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WD-xsl">

<xsl:template match = "apply">
<OMA>
<xsl:apply-templates/>
```

```
</OMA>
</xsl:template>

<xsl:template match = "eq">
  <OMS cd = "relation" name = "eq"/>
  <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "lt">
  <OMS cd = "relation" name = "lt"/>
  <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "gt">
  <OMS cd = "relation" name = "gt"/>
  <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "neq">
  <OMS cd = "relation" name = "neq"/>
  <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "leq">
  <OMS cd = "relation" name = "leq"/>
  <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "geq">
  <OMS cd = "relation" name = "geq"/>
  <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "min|max">
<xsl:choose>
 <xsl:when test="../condition">
   <xsl:choose>
     <xsl:when test="../min">
      <OMS cd = "relation" name = "min"/>
```

```
      </xsl:when>
      <xsl:otherwise>
       <OMS cd = "relation" name = "max"/>
      </xsl:otherwise>
    </xsl:choose>
    <OMBIND>
     <OMS cd="basic" name="elements"/>
     <OMBVAR>
      <xsl:apply-templates select="..//ci"/>
     </OMBVAR>
     <OMA>
      <OMS cd="basic" name="tuple"/>
      <xsl:apply-templates select="..//ci[first-of-type()]"/>
      <OMA>
       <xsl:apply-templates select="../condition"/>
      </OMA>
      <xsl:choose>
        <xsl:when test="../apply">
          <xsl:apply-templates select="../apply"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:apply-templates select="..//ci[first-of-type()]"/>
        </xsl:otherwise>
      </xsl:choose>
    </OMA>
   </OMBIND>
  </xsl:when>

  <xsl:otherwise>
    <xsl:choose>
      <xsl:when test="../min">
       <OMS cd = "relation" name = "min"/>
      </xsl:when>
      <xsl:otherwise>
       <OMS cd = "relation" name = "max"/>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:apply-templates select="../*[not(first-of-any())]"/>
  </xsl:otherwise>
```

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

</xsl:stylesheet>
```

A.13 Stat.xsl

```
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WD-xsl">

<xsl:template match = "mean">
  <OMS cd = "stats" name = "mean"/>
  <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "sdev">
  <OMS cd = "stats" name = "sdev"/>
  <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "var">
  <OMS cd = "stats" name = "var"/>
  <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "median">
  <OMS cd = "stats" name = "median"/>
  <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "mode">
  <OMS cd = "stats" name = "mode"/>
  <xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "moment">
  <OMS cd = "stats" name = "moment"/>
  <xsl:apply-templates select="../degree"/>
  <xsl:apply-templates select="../*[not(first-of-any() or last-of-any())]"/>
</xsl:template>
```

```
<xsl:template match = "degree">
  <xsl:apply-templates/>
</xsl:template>

</xsl:stylesheet>
```

A.14 Fns.xsl

```
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WD-xsl">

<xsl:template match = "ident">
<OMS cd = "fns" name = "identity"/>
<xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "inverse">
<OMS cd = "fns" name = "inverse"/>
<xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "compose">
<OMS cd = "fns" name = "compose"/>
<xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "lambda">
<OMS cd = "fns" name = "lambda"/>
<xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

</xsl:stylesheet>
```

A.15 La-mml.xsl

```
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WD-xsl">

<xsl:template match = "matrix">
<OMA>
```

```
<OMS cd = "la-mml" name = "matrix"/>
<xsl:apply-templates/>
</OMA>
</xsl:template>

<xsl:template match = "matrixrow">
<OMA>
<OMS cd = "la-mml" name = "matrixrow"/>
<xsl:apply-templates/>
</OMA>
</xsl:template>

<xsl:template match = "vector">
<OMA>
<OMS cd = "la-mml" name = "vector"/>
<xsl:apply-templates/>
</OMA>
</xsl:template>

<xsl:template match = "transpose">
<OMS cd = "la-mml" name = "transpose"/>
<xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

<xsl:template match = "determinant">
<OMS cd = "la-mml" name = "determinant"/>
<xsl:apply-templates select="../*[not(first-of-any())]"/>
</xsl:template>

</xsl:stylesheet>
```

A.16 A Stylesheet for Common Elements

```
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/TR/WD-xsl">

<xsl:template match = "apply">
<OMA>
<xsl:apply-templates select="*[first-of-any()]"/>
</OMA>
```

```
</xsl:template>

<xsl:template match = "cn[@type='real']">
  <OMF dec="{.}"/>
</xsl:template>

<xsl:template match = "cn[@type='constant']">
  <OMF dec="{.}"/>
</xsl:template>

<xsl:template match = "cn">
  <OMI>
   <xsl:apply-templates>
  </OMI>
</xsl:template>

<xsl:template match = "ci">
   <OMV name="{.}"/>
</xsl:template>

<xsl:template match = "/">
<annotation-xml encoding = "OpenMath">
<xsl:apply-templates/>
</annotation-xml>
</xsl:template>

</xsl:stylesheet>
```

Bibliography

[1] The OpenMath Steering Committee OpenMath Version 1.0 released, Dec., 1996. http://www.openmath.org

[2] W3C Recommendation 10-February-1998 *Extensible Markup Language (XML) 1.0* http://www.w3.org/TR/1998/Rec-xml-19980210

[3] W3C Recommendation *Mathematical Markup Language (MathML) 1.0 Specification* 07-April-1998

[4] W3C Working Draft *Extensible Stylesheet Language (XSL) version 1.0* 18-August-1998

[5] W3C Working Draft *Extensible Stylesheet Language (XSL) version 1.0* 16-December-1998

[6] J. Bosak *XML, Java, and the future of the Web* Oct. 02, 1997 http://www.xml.com/xml/pub/w3j/s3.bosak.html

[7] S. Vorkoetter *Proposed OpenMath specification: Draft version 1.1*, july 1995. http://www.w3.org/OpenMath/History/reports/prototype0-spec.ps.gz.

[8] S. Dalmas, M. Gaetano, and S. Watt. *An OpenMath 1.0 Implementation.* page 241-248. ACM Press, 1997

[9] PolyMath group *Java OpenMath Library, version 0.5,* http://pdg.cecm.sfu.ca/openmath

[10] N. Howgrave-Graham *OpenMath CDs* http://www.bath.ac.uk/~mapnahg/om/cds

[11] S. Dalmas et al *A Draft of the OpenMath Standard* ESPRIT project 24969: OpenMath