MathML-OpenMath Interface for REDUCE

Luis Alvarez Sobreviela

May 2, 2000

MATH0082 Double Unit Project

An OpenMath to MathML translator.

Candidate:	Alvarez,L.
Supervisor:	JHD
Checker:	
Review date:	3 March 2000
Final submission date:	2 May 2000
Equipment required:	own Linux Computer; BUCS

Description

OpenMath and MathML are two ways of representing mathematical objects. Semantically, OpenMath is a superset of (content) MathML. The aim is to build a translator from OpenMath to content MathML, using presentation MathML where necessary as in the example of rank in Section 5.3 on MathML `http://www.w3c.org/TR/REC-MathML/chapter5.html`. Since OpenMath is extensible, the translator will need to be. There is no a priori choice of implementation language. A *viva voce* examination will be held.

The project report should be no more than 40 pages.

Marking Scheme

Background research	15	α
Analysis of MathML/OpenMath translation	10	α
Design	20	2α
Implementation	25	α
Testing	10	
Report and Documentation	15	α
Viva and Demonstration	5	
Total	100	6α

Contents

1 Introduction **2**

2 Literature Review **4**
2.1 Mathematical Publishing 5
2.2 Mathematics and the Internet Challenge 5
2.2.1 Html and Mathematics 5
2.2.2 Embedded Graphics 6
2.2.3 Graphical Page Display 7
2.3 OpenMath and MathML 7
2.4 Current Support 8
2.5 The future 10

3 OpenMath/MathML Translation **11**
3.1 Constructing Objects 12
3.2 Elements and Functions 13
3.2.1 The Scope of Symbols 14
3.3 Differences in Structure 15
3.3.1 Selector functions and Matrices 15
3.3.2 Bound Variables 16
3.3.3 Intervals 18
3.3.4 MathML attributes 18
3.3.5 MathML constants 19
3.3.6 `partialdiff` and `diff` 20
3.4 Elements not Supported by both Standards 21
3.4.1 <`condition`> 21
3.4.2 <`declare`> 22
3.4.3 *matrixrow, matrixcolumn* 22
3.5 Extensibility 23
3.6 How to Handle the Translation problem 25
3.7 Conclusion 25

4 Program Design and Implementation **27**
4.1 System architecture 27
4.1.1 Module Requirements 28
4.2 The Intermediate Representation 29
4.3 Use of Tables in the Translation Process 30
4.4 XML Lexing and Parsing 32
4.4.1 The Lexer 32
4.4.2 The Parser 32
4.5 Possible Future Extensions 33

5 Testing **35**
5.1 Translation 35
5.1.1 Content Dictionaries 36
5.1.2 MathML Attributes 36
5.1.3 Extensibility 37
5.2 Standard Compliance 37
5.2.1 Parsing of Expressions 38
5.2.2 Generation of Expressions 38
5.3 Interface Limitations 39
5.4 Conclusion 40

A CDs and Symbols handled by the Interface **41**

Bibliography **46**

Chapter 1

Introduction

Nearly eight years after the appearance of the World Wide Web, it is still a difficult medium to use for the transmission of mathematics and scientific material in spite of its success in other areas. Sending mathematics via e-mail or reading mathematics into a software package from a web page is not a simple task, depriving the scientific community from a powerful communications tool which is the Internet. Likewise, displaying mathematics on the Internet in a way that allows editing and reuse has until now been impossible.

As the Internet continues to grow it is becoming ever more important to facilitate the exchange of mathematics amongst users and computer algebra software packages, offering automatic processing of expressions, searching, editing and reuse.

To overcome these difficulties, various companies and societies have joined together to produce standards for representing mathematics whilst preserving mathematical meaning. The World Wide Web Consortium [1] and the OpenMath society [2] have developed the two leading standards currently receiving most attention. These are MathML [3] and OpenMath [4] respectively.

The chief purpose of OpenMath is to facilitate consistent communication of mathematics between mathematical applications. MathML however, concentrates on displaying mathematics on the web whilst maintaining its meaning. Both standards are complementary and used together can provide the opportunity to expand our ability to represent, encode and successfully communicate mathematical ideas with one another across the Internet.

The primary aim of this project is to understand the differences and similarities between OpenMath and MathML, to assess their exchangeability and develop a way of mapping one standard to the other. The main objective will be to ultimately design and implement an interface running on REDUCE which will translate OpenMath into MathML and vice versa. This interface will provide REDUCE with the capability of exchanging mathematics with other applications as well as displaying output on the World Wide Web and reading from it, allowing REDUCE to join the MathML/OpenMath trend.

Chapter 2

Literature Review

The notation of mathematics has constantly evolved with the appearance of new concepts and ideas. Modern mathematical notation is the result of centuries of refinement. As a result of this, the sophisticated symbols with which we write mathematics pose certain problems when bringing them onto printed paper. Publishing mathematics is a difficult task simply because mathematics do not lend themselves easily to publication.

Recently, the advances in Internet publishing, following the Internet expansion, have added a new dimension to mathematical publishing. New problems as well as new requirements must be dealt with. We want the Internet not only to be a medium for displaying mathematics around the world, but also a communications tool for transmitting them.

How can we ensure that mathematics published on a web page are reusable? Editable? The outputs of one application should be displayed on the Internet in a way humans can understand and other applications can reuse. But because there is a distinction between presenting mathematical objects, and transmitting their content, merging both into one notation to achieve this duality is a non-trivial task.

In order to fully understand the motivations of this project, as well as appreciating its outcome, it is important to carefully illustrate any related issues. We will look into the development of mathematical publishing and how it has evolved with the growth of the Internet. This will permit us to better understand the need for mathematical representation standards such as MathML and OpenMath which we shall introduce. Finally we will talk about the relation between these standards, the existing software supporting them, and their future.

With such an overview of the current situation, the necessity of a MathML to OpenMath interface for REDUCE will become clear.

2.1 Mathematical Publishing

Before the foundation of the World Wide Web, encoding of mathematical documents was already a widespread practice. Back in the days when computers were starting to become popular, the ASCII character set (and encodings based on it) was the only widely available encoding scheme. The restrictions of such a limited symbol set were soon apparent.

In the mid seventies, Donald Knuth developed TeX, from which variants such as LaTeX stemmed. Layout and typesetting of mathematics is extremely demanding and until now, Donald Knuth's TeX had been able to address these difficulties in a successful way, appealing to the scientific community who has now made it a standard in scientific publishing. TeX has become the tool of choice for producing scientific and mathematical documents.

Despite its widespread use and ease with which it is authored, TeX does not preserve mathematical semantic value, making it unpractical for use in web documents and useless for transmission between applications. TeX is only concerned with describing the presentation of mathematics, not the content. Because people are interested in transmitting their ideas and research via e-mail or web pages it is fundamental that semantic value is kept.

While TeX is mainly a UNIX based application, PC applications dealing with mathematical encoding have also emerged. Generally these are equipped with a graphical user interface making them easier to use: Design Science's MS Word Equation Editor, FrameMaker, WordPerfect or ScientificWord are a few to name examples. All these applications[1] just deal with displaying mathematics and ignore semantic value. They are usually vendor specific making them unpractical for use in mathematical web publishing.

2.2 Mathematics and the Internet Challenge

2.2.1 Html and Mathematics

In the early 1990's, The World Wide Web Consortium's Html became the standard markup language for publishing on the World Wide Web. It has since evolved and has become an extensible and very powerful means of representing interactive Internet documents. In terms of representing mathematics however, Html has little support.

In the first versions of Html , no support for mathematics was included. It was not until 1993 that the first intent of embedding mathematics within Internet documents was attempted in the Html+ draft [5] presented by the World Wide Web Consortium. Equations were represented directly as

[1]It is worth noting that PC applications have not had the same success as TeX.

Html+ using an SGML [8] based notation, inspired by LaTeX's approach.

In 1994, the World Wide Web Consortium went further in mathematics Internet publishing by presenting the Html 3.0 draft [6] (which later was officially published as the Html 3.2 [7] specification with a few modifications) which offered a more comprehensive support. They claimed *"Html math is powerful enough to describe the range of math expressions you can create in common word processing packages, as well as being suitable for rendering to speech."*

Nonetheless, both drafts failed because of lack of interest from popular browser vendors. But even though the mathematical ideas in the Html 3.2 specification were never fully deployed, people started thinking more carefully about mathematics, and how they could be represented on the WWW.

In the meantime, while the World Wide Web Consortium and other societies continued working on developing mathematical support for Internet documents, other solutions to transmitting mathematics on the web arose. The lack of a standard approach to uniformly represent mathematics on the Internet pushed mathematicians and scientists to use a variety of different techniques to achieve this purpose. Let us give a brief overview of the main ones.

2.2.2 Embedded Graphics

One way of displaying mathematics on the web is by the use of embedded graphics inside Html documents. Mathematical equations are represented by graphical images (e.g. gifs) which all browsers display without difficulties. Formulae can be viewed in their original rendering, without the browser requiring additional fonts or external viewing programs.

Nevertheless, these images display low resolutions and printing them results in poor quality documents. There are also problems with alignment and sizing. Because graphical images are generally slow to download, documents might take more time than desired to be rendered. Since we are only dealing with images, the equations are not editable. No modifications can be done on them. For the same reasons, they are not reusable, because semantic value is completely lost.

This method is widespread but not very appreciated. In the Html 3.0 draft, the World Wide Web Consortium specifically states its intention of helping users avoid the use of inline images to display equations.

This is the approach used by programs such as LaTeX2Html [9] or TeX4ht [13] which can convert LaTeX and TeX documents to Html format for direct insertion into the Internet. LaTeX markup is translated into Html while mathematical equations are converted into graphical images. It is worth noting however, that there exist programs such as TtM [14] which translate the mathematical sections directly into MathML presentation markup .

2.2.3 Graphical Page Display

Another way of approaching the problem is by using graphical page displays. The page is rendered into a page-description language such as postscript or PDF. Internet browsers, aided by an external viewer or plug-in can then display the page in its integrity, including any mathematical formulae within it. When using this method, documents are displayed with exactly the same layout as the original documents, which could be TEX documents for instance. The printing resolution is also maintained at a high quality level.

But using an external viewer or plug-in involves everyone possessing a copy. A viewer also requires a verbose and large file format including all the non-standard fonts used. Just in the same way as the embedded graphics display, any mathematics contained within these documents looses its semantic value, as well as the possibility to edit it or modify it.

2.3 OpenMath and MathML

These interim solutions have only contributed to the problem by putting in evidence the need of a consistent standardized methodology for the transmission of mathematics via the World Wide Web. In view of the failure of existing methods MathML and OpenMath's[2] significance and importance increased. Both standards are complementary yet serving different purposes.

The primary aim of OpenMath is to facilitate reliable communication of mathematical objects between mathematical applications. It ensures semantic content is preserved within the notation. The semantic scope of OpenMath is defined within its content dictionaries (CD) where all symbols used are described defining their semantic value. Related symbols and functions are grouped into CD groups. It is expected that applications using OpenMath declare which CD groups they understand.

MathML however is World Wide Web oriented in that it seeks to display mathematics on web pages. MathML has two combinable versions, one encoding mathematical objects (presentation markup) and the other encoding mathematical meaning (content markup). Both versions allow authors to encode both the notation which represents a mathematical object and the mathematical structure of the object itself. Moreover, authors can mix both kinds of encoding in order to specify both the presentation and content of a mathematical idea.

In fact there are strong links between both recommendations. The communities developing both standards are closely related, with some members

[2]Describing these standards in detail is not in the scope of this report. We do encourage the reader to have a careful read through both standard specifications [2][3] in order to better understand this report and its implications.

belonging to both groups. This has resulted in both standards superceding each other in some areas.

The *core* OpenMath CD group is the principal CD group. The *core* CD group was designed based on MathML 1.0, extending the set of symbols covered by MathML 1.0. Its intention is not to be very specific, only covering everyday and K-12 (kindergarden to high school level) mathematics just as MathML does.

For completeness, a MathML CD group was introduced in the OpenMath standard. It is a subset of the *core* CD group and has the same semantic scope as do the content elements of MathML. It is expected that most applications will understand the *core* CD group, automatically understanding the MathML CD group.

The recently published MathML 2.0 version has incorporated elements of the *core* OpenMath CD group which weren't before in MathML 1.0. But in order to keep the scope of content markup down to a reasonable size, the designers of MathML have restricted the mathematics that it attempts to cover to high school level mathematics limiting MathML's ability to convey mathematical meaning. Because OpenMath is more powerful in this respect, the designers of MathML have introduced means allowing for extensibility. It is possible to encode semantic information inside MathML by embeding OpenMath objects within MathML code.

This demonstrates the close ties existing between both the World Wide Web Consortium and the OpenMath society. In the MathML 2.0 specification one can read: *"The MathML content elements are heavily indebted to the OpenMath project ..."*

2.4 Current Support

Both standards have received considerable attention, and have mobilized many developers. Support for MathML[3] and OpenMath is being introduced in many areas now that a future seems to profile itself.

The dominance of Java on the Internet today has made it a good candidate for offering a solution to the problem of publishing mathematics. The flexibility and power of Java applets can be used in conjunction with MathML or OpenMath to display mathematical formulae.

This approach is currently best represented by WebEQ [15]. WebEQ is a collection of programs and Java programming libraries dealing with all aspects of putting math on the Web. Because WebEQ is based on MathML, WebEQ tools can easily be combined with each other and with other MathML software to accomplish a wide range of tasks. The applet takes a representation of an equation as input, and displays it. The representation has to be some markup language which the applet supports

[3]For a comprehensive list of software supporting MathML look at the W3C web site [1]

(MathML or WebTEX). Another Java application is ICEBrowser [16]. A browser component written in Java which renders MathML.

By using a Java applet we encounter the same difficulties as when using embedded graphics. In addition to this, Java applets have a larger initial download overhead, which can be disturbing to some users. Java applets usually offer good equation displays, but different vendors supply different solutions and markup languages.

Another set of applications currently offering MathML support are plug-ins. The main distinction in principle between using plug-ins or Java applets is that plug-ins need to be pre-installed on the Internet browser for any rendering to take place. IBM Techexplorer [17] is a representative example under development. It currently supports MathML encodings. IBM's approach to the problem is definetely bordering the solution the scientific community is hoping to see. Techexplorer can display MathML and the quality of display is acceptable. Hopefully, IBM's techexplorer initiative will push other browser vendors and companies to adopt MathML as the leading standard.

But as with the other temporary solutions, plug-ins also have their limitations. Plug-ins have trouble getting the current HTML document font size, changing the size of the window to fit the display, or getting the current HTML document background color. Plug-ins such as IBM's are not yet widespread, and most people are not familiar with plug-in download and installation.

In the area of computer algebra, soon many computer algebra packages should have interfaces to both standards. An example of this is the MathML to REDUCE interface available in REDUCE 3.7, or the MathML interface built in Mathematica Version 4.

Various programs convert LATEX documents into MathML. This is important because of the large amount of documents written in LaTeX until now. An example of a program accomplishing this task is TtM [14] for instance.

Various equation editors such as MathType or Design Science's MS equation editor also support MathML. They manipulate expressions and offer easy to use graphical user interfaces. It is possible to export equations to MathML format.

Until now however, both Explorer and Netscape have not yet incorporated support for MathML, although they have committed themselves in doing so in the near future. Because these are the most popular browsers, it is important that they soon provide MathML facilities in order to boost the use of MathML.

2.5 The future

> *"While many in the mathematical and scientific community have already adopted LaTeX as the standard for writing papers, it appears that MathML is the future of scientific and mathematical notation on the Web."* Bob Henshaw, UNC.

Regardless of how efficient MathML and OpenMath are in transmitting and displaying mathematics, it is clear that they will only be of any use if all communities adopt it. It is expected however that most popular software companies working on the Internet or on computer algebra packages will soon support MathML and OpenMath. It seems as if MathML and OpenMath will recieve the necessary support due to the commitment that various big companies have already shown (IBM, Netscape, Microsoft, Wolfram, Design Science, and many others).

At the moment some browsers have already implemented MathML rendering facilities (Amaya for instance), and soon other bigger browser vendors will join the trend. Mozilla has recently released its latest browser which does render MathML. Netscape should follow soon with Navigator5. MathType from Design Science has released a new version incorporating various tools for dealing with MathML and OpenMath. For those not familiar with Design Science, they also make MS Word's equation editor. Other companies (mainly Stilo) are developing equation editors with MathML and OpenMath facilities which will soon hit the market.

While substantial progress has been made, there are still areas in which more work is required before MathML can be incorporated easily into the Internet. Further improvement in coordination between browsers and embedded elements will be necessary. Furthermore, higher printing resolution must be achieved.

MathML and OpenMath are the first XML based markup language to appear on the Internet. They will show the power and limitations of XML. An example has been set for other specialist areas which also want to benefit from the Internet.; areas such as Chemical Engineering or Music are using XML to develop representation standards. Both standards have been recieved enthousiastically and it will surely not take long before they are used widely by the scientific community.

Chapter 3

OpenMath/MathML Translation

MathML and OpenMath are closely related, serving a similar purpose of conveying mathematics across different applications. The aim of this analysis is to relate MathML and OpenMath to illustrate their similarities and differences. We intend it to be application independent, highlighting the problems arising when developing programs translating MathML to OpenMath and vice versa.

As is stated in the OpenMath standard [4], OpenMath objects have the expressive power to cover all areas of computational mathematics. This is certainly not the case with MathML. However, MathML was designed to be displayed on any MathML compliant renderer. The possibility to translate between them would allow OpenMath objects to be displayed, and MathML objects to have a wider semantic scope. But is a translation possible?

OpenMath and MathML have many common aspects. Some features of the standards help facilitate the translation, mainly that the structure of both standards is very similar. They both use prefix operators and are XML [10] based. They both construct their objects by applying certain rules recursively. Such similarities facilitate mapping across both standards.

Because both standards are XML based, their syntax is governed by the rules of XML syntax. In other words, the details of using tags, attributes, entity references and so on are defined in the XML language specification. By complying with the XML standard it is possible to use generic XML generators and validators. These can be programmed for the application being developed, or existing ones can be used.

Finally, OpenMath has specific content dictionaries mirroring MathML's semantic scope, which permit a straightforward mapping between both recommendations. Since both standards are simply different ways of representing mathematics, designed with translation in mind, mapping one to the other is certainly possible.

We shall look at all the areas of both recommendations where differences occur and how they pose difficulties to designing a translator. It is important to understand how objects are constructed and what they represent. We will then discuss how functions and operators are applied on their arguments. There are various specific structural differences between both standards which need to be properly understood; we will attempt to explain these differences and offer a method of translation for each one. We will also discuss how MathML supports extensibility and to what extent it is possible to implement such extensibility to accept new OpenMath symbols. To finish we will give an explanation of how to handle the translation problem.

Before we start our analysis, it is important that we define a few terms related to our analysis. We also encourage the reader to have a look at the standards in order to better appreciate this analysis. MathML and OpenMath *objects* convey the meaning of a mathematical expression and are represented as labelled trees. An object can also be called an *expression*. A *symbol* in OpenMath is used to represent a mathematical concept. For instance *plus* or *max* are considered symbols. We call *elements* the words enclosed within <> such as `<apply>` or `<OMA>`. Elements enclose other XML data called their 'content' between a 'start tag' (sometimes called a 'begin tag') and an 'end tag', much like in HTML. There are also 'empty elements' such as `<plus/>`, whose start tag ends with /> to indicate that the element has no content or end tag.

3.1 Constructing Objects

Constructing objects in MathML and OpenMath is done in similar ways. MathML uses elements termed *containers* and OpenMath uses elements called *constructs*. They are both closely related, and most of them are easily interchangeable. The nature of the constructors in both standards is rather different, but their usage is the same.

OpenMath objects can be created by applying a symbol onto a series of arguments. These are the objects created by *application* and are surrounded by `<OMA>...</OMA>` elements. In MathML the approach is different. MathML possesses more constructors and they are more specific. It is important to note that OpenMath objects constructed with the `<OMA>` element may translate to various constructors in MathML.

In OpenMath for instance, defining a list or a matrix would be done by applying the application constructor on the *list* or *matrix* symbol followed by the contents of the list or matrix. In MathML however, a list would require the `<list>... </list>` constructor, and a matrix would need the `<matrix>...</matrix>` constructor.

Most OpenMath symbols constructed by application are constructed in MathML using the `<apply>` constructor. But there are exceptions which

OpenMath	**MathML**
`<OMA>`	`<interval>`, `<set>`, `<list>`, `<matrix>`, `<vector>`, `<apply>`, `<lambda>`, `<reln>`.
`<OMATTR>`	*attributes associated to a tag*
`<OMI>`, `<OMF>`	`<cn>`
`<OMV>`	`<ci>`
`<OMSTR>`	*not supported*
`<OMBIND>`	*not supported*
not supported	`<declare>`

Table 3.1: Relation between constructors

do not map to `<apply>` tags. It is important that all exceptions such as `matrix`, `list`, `set` and others are determined and that the appropriate MathML constructor is used when translating. Table 3.1 shows what possible MathML constructors `<OMA>` can map to.

OpenMath objects can also be constructed using the `<OMBIND>` element. This consists in binding a symbol to a function with zero or more bound variables. MathML does not have an equivalent, and so symbols which use the *binding* construct in OpenMath, like `lambda` or `forall`, may have different ways of being constructed in MathML. `lambda` uses a specific constructor in MathML, whereas `forall` uses the `<apply>` construct. It is very important in order to ensure proper translation, to determine which OpenMath symbols use the binding constructor and what their MathML equivalent is.

There are objects constructed by attributing a value to an object. These are objects constructed by *attribution* and employ the `<OMATTR>` elements. MathML also allows objects to possess attributed values called attributes. The translation is straightforward.

There are other constructors which we do not mention in more detail because there exists a direct mapping between both standards. This is the case of `<OMI>`, `<OMF>`, `<OMV>` `<cn>` and `<ci>`. Table 3.1 shows the relation between them.

3.2 Elements and Functions

MathML has a classification[1] which categorises elements according to the number of arguments they accept and the types of these arguments. This classification can be summarised for our purpose into the following:

unary elements accepting 1 argument

binary elements accepting 2 arguments

[1]MathML standard section 4.2.3

nary elements accepting 3 or more arguments

operators: elements whose arguments are given following a specific syntax. This includes symbols such as `int, sum, diff, limit, forall` and a few others.

This classification is not explicitly stated in the OpenMath standard but can also be used since OpenMath symbols fit well into these categories. By gathering OpenMath and MathML symbols into these defined groups according to their syntax, it is possible to define specific translating procedures which deal with all symbols in one group in the same way.

For instance one procedure could parse through any unary function by reading in the symbol and then the one argument. Printing out unary functions would be done by one procedure which would output the symbol in MathML or OpenMath followed by that one argument.

The advantages of this classification are that it greatly simplifies the translation. Parsing and generation of all symbols would then be the task of a few generic procedures. However, symbols contained in the *operators* group require more attention, since they have different ways of reading in arguments. Specific procedures need to be implemented for such cases. We will discuss these in more detail later.

3.2.1 The Scope of Symbols

When dealing with a function or an operator in mathematics, it is important that its scope is well defined. MathML and OpenMath both specify the scope of an operator by enclosing it with its arguments inside opening and closing tags. In MathML, the opening and closing tags `<apply>` are employed, and in OpenMath one uses the opening and closing tags `<OMA>`.

However, OpenMath's grammar as it is defined in the OpenMath standard in section 4.1.2 can produce OpenMath objects where the scope of an operator is ambiguous, in which case a parser would have great difficulties validating the syntax for translation. Let us illustrate this problem with the two OpenMath expressions in figure 3.1 which are grammatically correct.

Example 2 demonstrates how the use of `<OMA>` tags help define clearly the scope of each operator. A parser can then without difficulty interpret the expression and translate it correctly. Example 1, on the other side, shows how insufficient use of `<OMA>` tags can lead to ambiguous expressions both for automatic parsers and humans.

MathML is stricter when defining the scopes of operators. Every operator must be enclosed with its own `<apply>` tags. This difference between both standards is source of problems. The expression in Example 1 does not allow the scopes of the operators to be determined with accuracy, and so an equivalent MathML expression cannot be produced.

Example 1

```
<OMOBJ>
 <OMA>
   <OMS cd=arith1 name=plus/>
   <OMS cd=arith1 name=times/>
   <OMV name=x/>
   <OMV name=y/>
   <OMV name=z/>
   <OMI>6</OMI>
 </OMA>
</OMOBJ>
```

Example 2

```
<OMOBJ>
 <OMA>
   <OMS cd=arith1 name=plus/>
   <OMA>
     <OMS cd=arith1 name=times/>
     <OMV name=x/>
     <OMV name=y/>
   </OMA>
   <OMV name=y/>
   <OMI>6</OMI>
 </OMA>
</OMOBJ>
```

Figure 3.1: The importance of defining scopes

When developing an OpenMath/MathML translator, it is important to specify that operator scopes in OpenMath must be accurately defined, or else translation to MathML is not possible. The use of `<OMA>` tags must be imposed.

3.3 Differences in Structure

There are MathML and OpenMath elements which require special attention. Mainly because there are elements constructed differently in MathML as they are in OpenMath and because some elements have no equivalent in both standards. Such cases must be well understood before starting to implement any translator. We shall look at these cases and propose a reliable method for overcoming the differences and implementing an efficient solution. We will mention bound variables, element attributes and constants representation.

There exist elements in both standards which represent the same mathematical concept, but where the syntactical structure is different. The following list shows these elements: *matrices, limits, integrals, definite integrals, differentiation, partial differentiation, sums, products, intervals, selection from a vector* and *selection from a matrix.*

3.3.1 Selector functions and Matrices

Let us first look at: *matrices, selection from a matrix* and *selection from a vector.* These elements exist in both recommendations, but differ syntactically.

Selection from a matrix and from a vector is done by the `<selector/>` element in MathML and by the symbols *vector_selector* and *matrix_selector* in OpenMath. Because MathML uses the same element to deal both with matrices and vectors, it is necessary for the parser to determine what the arguments of the expression are before finding the correct equivalent OpenMath. If the expression has a matrix as argument, then *matrix_selector* is the correct corresponding symbol. If the argument is a vector then the corresponding symbol is *vector_selector*.

It is important to note as well the order of arguments. The MathML `<selector/>` tag first takes the vector or matrix object, and then the indices of selection. In OpenMath it is the other way around. First the indices of selection are given and then the object.

Another element where differences in structure are important is the matrix element. OpenMath has two ways of representing matrices. One representation defined in the `"linalg1"` CD and the other defined in the `"linalg2"` CD. A matrix is defined as a series of matrixrows in `"linalg1"`, exactly as in MathML. For such matrices, translation is straightforward.

However, `"linalg2"` defines a matrix as a series of matrix columns. This representation has no equivalent in MathML. It is important that a translator is capable of understanding both representations in order to offer correct translation.

When dealing with a `"linalg2"` matrix, a procedure can be implemented which given the matrix columns of a matrix, returns a series of matrix rows representing the same matrix. From these matrix rows, a MathML expression can be generated.

3.3.2 Bound Variables

The remaining elements *limit, integrals, definite integrals, differentiation, partial differentiation, sums, and products* have a similar structure and can be treated in a similar way when translating. Following the classification in section 3.2 these elements go in the *operators* group.

What characterises these elements is that in MathML they all specify their bound variables explicitly using the `<bvar>` construct. However, in OpenMath, the bound variables are not explicitly stated. OpenMath expressions are the result of applying the symbol on a lambda expression. In order to determine the bound variable the parser must retrieve it from the lambda expression. Let us illustrate this problem by contrasting two equivalent expressions on figure 3.2

In MathML, the index variable is explicitly stated within the `<bvar>` tags. It is part of the `<sum/>` syntax and is obligatory. In OpenMath, the *sum* symbol takes as arguments an interval giving the range of summation and a function. Specifying the bound variable is not part of the syntax. It is contained inside the lambda expression. This same difference in structure exists with the other operators mentioned above.

OpenMath

```
<OMOBJ>
 <OMA>
  <OMS cd="arith1" name="sum"/>
  <OMA>
   <OMS cd="interval1" name="interval"/>
   <OMI> 1 </OMI>
   <OMI> 10 </OMI>
  </OMA>
  <OMBIND>
   <OMS cd="fns1" name="lambda"/>
   <OMBVAR>
     <OMV name="x"/>
   </OMBVAR>
   <OMA>
    <OMS cd="arith1" name="divide"/>
    <OMI> 1 </OMI>
    <OMV name="x"/>
   </OMA>
  </OMBIND>
 </OMA>
</OMOBJ>
```

MathML

```
<math>
 <apply><sum/>
   <bvar>
     <ci>i</ci>
   </bvar>
   <lowlimit>
     <cn>0</cn>
   </lowlimit>
   <uplimit>
     <cn>100</cn>
   </uplimit>
   <apply><power/>
       <ci>x</ci>
       <ci>i</ci>
   </apply>
 </apply>
</math>
```

Figure 3.2: Use of bound variables

When translating any of these elements, it is necessary to support automatic generation and decoding of lambda expressions. Thus when going from OpenMath to MathML, the bound variable and the function described by the lambda expression need to be extracted to generate valid MathML.

When passing from MathML to OpenMath, the variable contained inside the `<bvar>` tags and the function given as argument would have to be encoded as a lambda expression. This is possible for all MathML expressions of this type, and correct OpenMath is simple to produce.

Thus by retrieving bound variable information from OpenMath lambda expressions, it is possible to translate to MathML. But OpenMath grammar does not impose the use of lambda expressions to define bound variables. Because of this flexibility, it is possible to construct OpenMath expressions which cannot be translated to MathML by an automatic translator. If one looks at the `"calculus1"` CD, the OpenMath examples of *int* and *defint* do not specify their variable of integration. A parser would not determine the variables of integration and an equivalent MathML expression would not be possible.

This is a problem for an OpenMath/MathML translator with no easy solution. A parser intelligent enough to extract the correct bound variables of an expression is very difficult to implement. We recommend that OpenMath expressions which do not specify all the necessary information for translation are ignored. The use of lambda expressions should be required.

3.3.3 Intervals

Some operators require an interval to be given specifying the range within which a variable ranges. The *sum* or *product* operator are some good examples. They both take as argument the interval giving the range of summation or multiplication. Other operators accepting intervals in some cases are *int* and *condition*.

Both in MathML and OpenMath these operators define ranges with intervals, but differently. OpenMath defines intervals using specific interval defining symbols found in the `interval1` CD. MathML can use either the interval element or the tags `<lowlimit>` and `<upperlimit>`. These two tags do not have an OpenMath equivalent and so when encountered must be transformed into an interval. This is not difficult since one must simply merge the lower and upper limits into the edges of an interval.

3.3.4 MathML attributes

There are OpenMath symbols which map to the same MathML element, and are only distinguished by the attributes characterising the MathML element. A MathML element which illustrates this is `<interval>`. The interval element in MathML has a `closure` attribute which specifies the type of interval being represented. This attribute takes the following values: `open`, `closed`, `open_closed`, `closed_open`. Depending on the attribute value, a different OpenMath symbol will be used in the translation. The following example illustrates how one element with different attribute values maps to different OpenMath symbols.

```
<interval type="closed">

<OMS cd="interval1" name="interval_cc"/>
```

are equivalent and so are

```
<interval type="open">

<OMS cd="interval1" name="interval_oo"/>
```

When a translator encounters such elements, it is necessary that the MathML elements generated posses these attributes, or else semantic value is lost. Table 3.3.4 shows the relation between all MathML elements whose attributes are of importance and their equivalent OpenMath symbols.

MathML element	Attribute values	OpenMath symbol
`<interval>`	*default*	*interval*
	`closure="open_closed"`	*interval_oc*
	`closure="closed_open"`	*interval_co*
	`closure="closed"`	*interval_cc*
	`closure="open"`	*interval_oo*
`<tendsto>`	*default*	*above*
	`type="above"`	*above*
	`type="below"`	*below*
	`type="both_sides"`	*null*
`<set>`	*default*	*set*
	`type="normal"`	*set*
	`type="multiset"`	*multiset*

Table 3.2: Equivalent OpenMath symbols to the different attribute values of MathML elements

3.3.5 MathML constants

In MathML, constants are defined as being any of the following: `e`, `i`, `pi`, `gamma`, `infinity`, `true`, `false` or `not a number (NaN)`. They appear within `<cn>` tags when the attribute `type` is set to `constant`. For instance π would be represented in MathML as:

```
<cn type="constant">pi</cn>
```

In OpenMath, these constants all appear as different symbols and from different CDs. Hence, we face a similar problem as we did with MathML attributes. The `<cn>` tag with the attribute set to `constant` can map to different OpenMath symbols.

It is important that the translator detects the use of the `constant` attribute value and maps the constant expressed to the correct OpenMath symbol.

MathML also allows to define Cartesian complex numbers and polar complex numbers. A complex number is of the form two real point numbers separated by the `<sep/>` tag. For instance $3 + 4i$ is represented as:

```
<cn type="cartesian_complex"> 3 <sep/> 4 </cn>
```

OpenMath is more flexible in its definition of complex numbers. The real and imaginary parts, or the magnitude and argument of a complex number do not have to be only real numbers. They may be variables. This allows OpenMath to represent numbers such as $x+iy$ or $re^{i\theta}$ which cannot be done in MathML.

So how should one map such an OpenMath expression to MathML? Because there is no specific construct for such complex numbers, the easiest way is to generate a MathML representation using simple operators. The two expressions in figure 3.3 are equivalent and illustrate how a translator should perform:

```
<OMOBJ>
  <OMA>
    <OMS cd="nums1" name="complex_polar"/>
    <OMV name="x"/>
    <OMV name="y"/>
  </OMA>
</OMOBJ>

 <math>
    <apply><times/>
       <ci> x </ci>
       <apply><exp/>
          <apply><times/>
             <ci> y </ci>
             <cn type="constant"> &imaginaryi; </cn>
          </apply>
       </apply>
    </apply>
 </math>
```

Figure 3.3: How to translate complex numbers

The problem is the same when representing rationals, since OpenMath allows variables to be used as elements of a rational number, whereas MathML only allows real numbers.

3.3.6 `partialdiff` and `diff`

In both standards it is possible to represent normal and partial differentiations. But the structures are different. Let us first look at `diff`. In MathML, it is possible to specify the order of the derivative. In OpenMath, differentiation is always of first order. The trouble here is translating MathML expressions where the order of derivation is higher than one. There is no equivalent representation in OpenMath.

What can be done to overcome this discrepancy is to construct an OpenMath expression differentiated as many times as is specified by the MathML derivation order. For instance, when dealing with a MathML second order derivative, the equivalent OpenMath expression could be a first order derivative of a first order derivative. This will surely generate very verbose OpenMath in cases where the order of derivation is high, but at least will convey the same semantic meaning and surmounts OpenMath's limitation.

The case of partial differentiation is complicated. The representations in both standards are very different. In MathML one specifies all the variables of integration and the order of derivation of each variable. In OpenMath one specifies a list of integers which index the variables of the function. Suppose a function has bound variables x, y and z. If we give as argument the integer list $\{1, 3\}$ then we are differentiating with respect to x and z. The differentiation is of first order for each variable.

Translating partial differentials from OpenMath to MathML is simple, because the information conveyed by the OpenMath expression can be represented without difficulty by MathML syntax. However the other way around is difficult. Given OpenMath's limitation of only allowing first order differentiation for each variable, many MathML expressions which differentiate with respect to various variables and each at a different degree cannot be translated. We recommend that such MathML expressions are discarded by the translator.

3.4 Elements not Supported by both Standards

There are some elements which have no equivalent in both standards. These are mainly the MathML elements `<condition>` and `<declare>`and the OpenMath *matrixrow* and *matrixcolumn* symbols.

3.4.1 <condition>

The `<condition>` element is used often throughout MathML and is necessary to convey certain mathematical concepts. There is no direct equivalent in OpenMath, making translation impossible for certain expressions.

The `<condition>` element is used to define the 'such that' construct in mathematical expressions. Condition elements are used in a number of contexts in MathML. They are used to construct objects like sets and lists by rule instead of by enumeration. They can be used with the `forall` and `exists` operators to form logical expressions. And finally, they can be used in various ways in conjunction with certain operators. For example, they can be used with an int element to specify domains of integration, or to specify argument lists for operators like min and max.

The example in figure 3.4 represents $\{\forall x | x < 9 : x < 10\}$ and shows how the `<condition>` tags can be used in a MathML expression. This MathML expression has no OpenMath equivalent because OpenMath does not allow to specify any conditions on bound variables.

The `<condition>` tags are used in the following MathML elements: *set, forall, exists, int, sum, product, limit, min* and *max*. In all of these elements

```
<math>
  <apply><forall/>
    <bvar>
      <ci> x </ci>
    </bvar>
    <condition>
      <apply><lt/>
        <ci> x </ci>
        <cn> 9 </cn>
      </apply>
    </condition>
    <apply><lt/>
      <ci> x </ci>
      <cn> 10 </cn>
    </apply>
  </apply>
</math>
```

Figure 3.4: Use of <condition>

except *limit*, the use of <condition> tags makes translation impossible.

The case of *limit* is different because OpenMath does allow constraints to be placed on the bound variable; mainly to define the limit point and the direction from which the limit point is approached.

3.4.2 <declare>

The <declare> construct is used to associate specific properties or meanings with an object. It was designed with computer algebra packages in mind. OpenMath's philosophy is to leave the application deal with the object once it has received it. It is not intended to be a query or programming language. This is why such a construct was not defined. A translator should deny such MathML expressions.

3.4.3 *matrixrow, matrixcolumn*

In the MathML specification it is stated that *'The matrixrow elements must always be contained inside of a matrix'*. This is not the case in OpenMath where the *matrixrow* symbol can appear on its own. A matrix row encountered on its own has no MathML equivalent. However, when it is encountered within a matrix object, then translation is possible.

As we mentioned earlier, it is possible to translate a matrix defined with matrixcolumns to MathML. However, if a matrixcolumn is found on its own it does not have a MathML equivalent.

3.5 Extensibility

OpenMath already possesses a set of CDs covering all of MathML's semantic scope. These CDs belong to the MathML CD Group. It is clear that these CDs must be understood by an OpenMath/MathML interface. There are as well a few other symbols from other CDs which are not in the MathML CD Group but can be mapped such as matrices defined in `"linalg2"`.

But OpenMath has the capability of extending its semantic scope by defining new symbols within new content dictionaries. This facility affects the design of any OpenMath compliant application. When it comes to translating to MathML, it is necessary that newly defined symbols are properly dealt with. A translator should have the ability to recognise any symbol with no mapping to MathML.

But how do we deal with most symbols outside the MathML CD Group? Or with new symbols which will continue to appear as OpenMath evolves? How do we map them to MathML?

MathML, as any system of content markup, requires an extension mechanism which combines notation with semantics. Extensibility in MathML is not as efficient as in OpenMath, but it is possible to define and use functions which are not part of the MathML specification. MathML content markup specifies several ways of attaching an external semantic definition to content objects.

Because OpenMath contains many elements which have no equivalent in MathML, and because OpenMath can have new CDs amended to it, we will need to use these mechanisms of extension. The `<semantic>` element is used in MathML to bind a semantic definition with a symbol. An example taken from the MathML specification [3] section 5.2.1[2] shows how the OpenMath 'rank' operator (non existent in MathML) can be encoded using MathML. The MathML encoding of rank is shown in figure 3.5:

It shows that an OpenMath operator without MathML equivalent is easily contained within `<semantic>` tags and can be applied on any number of arguments.

This method works well when dealing with operators constructed by *application* (between `<OMA>` tags), because MathML also constructs expressions by application (between `<apply>` tags). It is assumed they take any number of arguments. However, OpenMath can also construct expressions by binding symbols to their arguments. As we described earlier (section 3.1), this method has no equivalent in MathML.

So what happens when a new symbol is encountered which is constructed by binding in OpenMath? Enveloping the new symbol inside `<semantic>` tags will produce an incorrect translation.

It is first necessary to determine if the new symbol encountered is con-

[2]CHECK!!!!`http://www.w3.org/WD-MathML2-19991222/chapter5.html#mixing:parallel`

```
<math>
  <apply><eq/>
    <apply><fn>
        <semantics>
          <ci><mo>rank</mo></ci>
          <annotation-xml encoding="OpenMath">
            <OMS cd="linalg3" name="rank"/>
          </annotation-xml>
        </semantics>
      </fn>
      <apply><times/>
        <apply><transpose/>
          <ci>u</ci>
        </apply>
        <ci>v</ci>
      </apply>
    </apply>
    <cn>1</cn>
  </apply>
</math>
```

Figure 3.5: Encoding of OpenMath symbol 'rank' in MathML

structed by binding or not. In order to do so, a file describing the new symbol specifying these details could be read in by the translator. This file could be the CD where the symbol is defined. But unfortunately CDs are written in a human readable way, and there is no way a program could determine the construction method of a particular symbol or the number and type of arguments it takes.

One would need to read in the STS file of a symbol. But the best way would be by checking the tag preceding the new symbol given by the OpenMath input. If it was `<OMBIND>` then we are sure this symbol is constructed by binding. Nonetheless, accurate mapping would be impossible. As we have seen before, MathML only offers extensibility constructing operators by application. It is not possible to define new containers, new types, or new operators constructed differently such as those constructed by binding.

While it is possible to define certain new symbols in MathML, the advantages of OpenMath extensibility would create problems for a translator to MathML. This is why it is stated in the OpenMath standard in section 2.5 that *'it is envisioned that a software application dealing with a specific area of mathematics declares which content dictionaries it understands'*. A MathML translator deals with the area of mathematics defined by MathML and should understand all CDs within the MathML CD Group. Any other symbols will be properly translated if they are enclosed inside `<OMA>` tags.

Extensibility is limited by the extension mechanisms offered by MathML.

3.6 How to Handle the Translation problem

Although there are surely many ways to tackle the translation problem, there are a few requirements which must be respected by any OpenMath/MathML translator. Mainly that content dictionaries and symbols are dealt with correctly during translation in both directions.

In OpenMath, symbols always appear next to the content dictionary they belong to. The `<OMS>` element always takes two attributes: the symbol's name and the symbol's corresponding CD. Two symbols with the same name coming from different CDs are considered to be different.

When parsing OpenMath, a translator must ensure that the symbols read belong to the correct CDs, if not it should conclude the symbol has a meaning it does not understand and deal with it accordingly. Because an OpenMath/MathML translator will understand all MathML related CDs, symbols encountered are considered valid if they come from this CD group. Symbols with the same name, but from unknown CDs should be enclosed within `<semantic>` tags when possible.

We face the same requirement when generating OpenMath. All OpenMath symbols output from the translator must appear next to their correct CDs. If we are translating the MathML element `<plus/>`, the corresponding OpenMath symbol *plus* must appear next to the `arith1` CD.

This requires a translator to keep a database relating each understood symbol with its CD. This database must allow the translator to detect unknown symbols, or to accept some symbols from different CDs with the same name which have MathML equivalents. This is the case of *matrix* which belongs to various CDs (`linalg1`, `linalg2`) as do the symbols *in*, *inverse*, *setdiff*, *vector*, *arcsinh* to name a few.

These symbols belonging to various CDs pose a problem when translating from MathML to OpenMath. Which CD do we choose? *inverse* for instance belongs to *fns1* and *arith2*. Priority should be given to the CD belonging to the MathML CD group. If both CDs belong to the MathML then common sense should guide which CD to place. It is up to the designer.

An OpenMath/MathML interface must be very rigorous when dealing with content dictionaries. Any mistake may produce invalid OpenMath or reject valid OpenMath expressions.

3.7 Conclusion

It is clear now that a translation is possible. Putting apart the difficulties described in this analysis, their are many similarities between both standards.

As we have seen, expressions are constructed similarly and the application of functions is practically identical.

However, the various differences of structure can limit the power of a translator in some situations. Mainly when translating partial differentiations or applying conditions to bound variables.

The design of any translator requires a good understanding of both standards and how they represent mathematical concepts. The information described in this document will guide the design of an OpenMath/MathML translator.

Chapter 4

Program Design and Implementation

The design of an OpenMath/MathML interface must aim to keep the structure simple, extensible if needed and easy to maintain. This document will attempt to describe the structure of the overall system and the individual modules which compose it. A common interface coordinating the separate components will be analysed and defined.

Furthermore we will explain why the system will be table based and what advantages this offers for our application. Because both OpenMath and MathML are XML languages, we must specify the requirements the translator's lexer and parser must follow. Finally we will see what new functionalities can be added to the interface in possible future extensions.

4.1 System architecture

The task of translating one language to another, as is the case of our OpenMath/MathML interface, can be compared to the task performed by a compiler when passing from a programming language to a computer executable representation.

We will need to lex and parse an expression, represent it in some intermediate language which allows a certain degree of freedom for manipulation, and then from there an expression can be generated in the target language.

Following this approach, the architecture of the REDUCE OpenMath to MathML interface is going to be composed of four independent modules. One for each of the following tasks:

- Passing MathML to the intermediate representation
- Passing OpenMath to the intermediate representation
- Passing from the intermediate representation to MathML

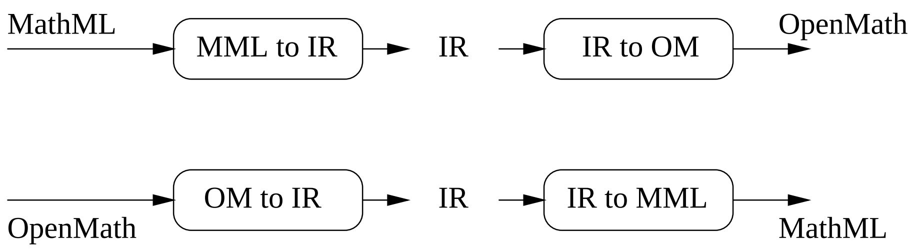

Figure 4.1: OpenMath/MathML Interface System Architecture

- Passing from the intermediate representation to OpenMath

Dividing the interface into these separate modules gives us the possibility to better understand the overall process of translation. It has the advantage of permitting efficient modifications to the system.

If MathML syntax were to change, for instance, it would only be necessary to modify two of the four modules. This separation also makes it easy to add extensions. By implementing a module going from the intermediate representation to LaTeX, it is possible to extend the interface's capabilities to offer OpenMath to LaTeX or MathML to LaTeX translations. Figure 4.1 illustrates the system architecture.

4.1.1 Module Requirements

Each of these modules has several requirements to respect. These requirements ensure the system is efficient and behaves satisfactorily. *(Here IR stands for intermediate representation)*

MathML to IR: This module parses through MathML and generates an equivalent expression in the intermediate representation. It should ensure that the input given is not lexically or syntactically incorrect. In which case the translation process is aborted. Incorrect or unimportant attribute values should be ignored unless they compromise the translation process. Both MathML 1.0 and MathML 2.0 expressions must be accepted as valid and parsed. It should be designed so any modification in MathML can be easily adapted to.

IR to MathML: This module generates valid MathML from the intermediate representation of an expression. The user should have the option to generate either MathML 2.0 or MathML 1.0, since most applications today are only MathML 1.0 compliant . In order to embed MathML into a web page for rendering by a plug-in, there should also be an option outputting the MathML inside HTML `<embed>` tags.

OpenMath to IR: This module reads in OpenMath expressions and transforms them into the intermediate representation. It should ensure that the input given is not lexically or syntactically incorrect. In which case the translation process is aborted. Symbols must be checked to see if they have a MathML equivalent. This means checking each symbol against the CD it belongs to and then looking up in a table to see whether a mapping is possible. If there is no equivalent, this module must encode the OpenMath symbol into the intermediate representation as an unknown symbol for inclusion in MathML `<semantic>` tags.

IR to OpenMath: This module generates valid OpenMath from the intermediate representation. It is important that all symbols generated appear next to the correct CD to which they belong. This is done by consulting a table containing this information.

Because it is important to specify which OpenMath CDs an application handles, Appendix A gives a comprehensive list of all the OpenMath CDs and elements which are supported by the translator.

4.2 The Intermediate Representation

If the breakdown of the system into separate modules is to be effective, we need a clean interface between all parts. An intermediate representation representing expressions in a generic way accomplishes this task. For an intermediate representation to be useful, it is important that it conveys and preserves all the information MathML and OpenMath objects are capable of representing. Let us look at the requirements such an intermediate representation must satisfy for use in our OpenMath/MathML interface.

Both OpenMath and MathML build expressions by using prefix operators. REDUCE's symbolic representation of expressions also uses prefix operators to construct expressions. This connection motivates us to use prefix operators in our intermediate representation, thus allowing an uncomplicated mapping between the intermediate representation, OpenMath, MathML and REDUCE's representation of expressions. Subsequently the intermediate representation is closely related to the parse trees of each language.

Given that MathML elements may take attributes changing their semantic meaning, it is necessary that attribute values are represented by the intermediate representation. Thus permitting MathML elements mapping to different OpenMath symbols (depending on their attribute values) to be correctly translated from one standard to the other. The attributes conveyed by the intermediate representation are then interpreted differently by the various modules according to the context they appear in.

Considering that OpenMath extensibility is a key issue, our intermediate representation must be able to encode objects without MathML equivalent. The unsupported OpenMath symbol and its CD will be passed on from the **OpenMath to IR** module to the **IR to MathML** module so that the MathML extension mechanism is employed.

Moreover, the intermediate representation will need to be simple to manipulate. Since RLISP is the programming language in which this interface is written, we must keep in mind the possibilities and limitations this language offers. Therefore the intermediate representation expressions will be structured as lists. Lists are the basic data structures in RLISP and there exist many commands permitting very easy and efficient manipulation of them.

Because our intermediate representation is designed in terms of the syntactic structure of both OpenMath and MathML, and certain subroutines are attached to the MathML and OpenMath production rules to produce proper intermediate encoding, we can classify our methodology as *syntax-directed translation* [11]. Basically, the actions of the syntax analysis phase guide the translation. Thus the intermediate code is generated as syntax analysis takes place.

4.3 Use of Tables in the Translation Process

The complexity and diversity of MathML and OpenMath elements require that a translator has some way of keeping information concerning all elements. The parsing and generation of OpenMath requires a translator to have some way of knowing which content dictionaries symbols belong to. Similarly, the correct procedures must be employed upon each element encountered. This information must be stored in a readily accessible way. It is important to design these tables and that we understand how each module will use them to appropriately accomplish their tasks.

The information guiding the translator can be either hand coded into the program or gathered into tables. Hand coding is complex and useful only in situations where an element needs to be handled in a very precise way. Tables however can contain organized information related to each element useful when parsing and generating expressions.

We believe using a table-based system is more efficient for our application and can produce better and more compact code, thus improving code readability, extensibility and maintenance. Because a translator must deal with a variety of elements, most of similar structure, a table-based system permits the translator to relate an element to a set of functions and/or information. This way, any modifications of the MathML standard can be easily adapted to by modifying a table or adding a new entry to it.

The idea is to gather in a few tables all the necessary information for properly handling all MathML and OpenMath recognized elements. Let us describe the main tables[1] which are used by the interface. To better understand the system we will describe how they should be used by each module to accomplish the task.

MathML to IR Module

All MathML elements are stored in the tables `constructors!*`, `relations!*` and `functions!*`. These tables determine what functions must be called for each MathML element encountered and what the equivalent intermediate representation operator is.

When a MathML object is encountered, the first element will inform us of how the expression is constructed. We look this element up in the `constructors!*` table to call the proper function which deals with objects constructed in this manner.

If the expression constructor is the `<reln>` element then the `relations!*` table is used. This table will determine which function to call as well as containing the equivalent intermediate representation operator. The `functions!*` table is the same as the `relations!*` table only that it contains all operators appearing within `<apply>`...`</apply>` instead.

These tables together will inform the translator of how to deal with all MathML elements

New MathML elements can be added to these tables to modify the translator's scope. An existing procedure can be related to the new element, or a new procedure can be implemented and added to the table next to the element's entry. An equivalent intermediate operator must also be defined here.

IR to MathML Module

When an intermediate representation expression must be translated to MathML the table `ir2om_mml!*` specifies which function to call for the translation of each intermediate representation operator. As an intermediate expression is parsed, this table will ensure that proper production of MathML is achieved. This table also contains the function to call when producing OpenMath.

New operators are added to this table. The procedure name specifying how the new IR operator is translated to MathML is also added to the table.

OpenMath to IR Module

OpenMath objects must be thoroughly checked for various reasons. Firstly, not all OpenMath symbols have MathML equivalents. Table `mmleq!*` contains all OpenMath symbols which easily translate to MathML. If a symbol is

[1] These tables are defined in the file `tables.red`

not contained within this table then it is searched inside tables `special_cases!*` and `special_cases2!*`.

Table `special_cases!*` contains all OpenMath symbols which have a MathML equivalent but under a different name. It also deals with OpenMath symbols mapping to one MathML element but with different attribute values. This table will also specify where necessary the correct attribute types and values the MathML equivalent element must take.

Table `special_cases2!*` contains all OpenMath symbols which require careful translation. For each element, a specific function is associated. These functions are specially designed to deal with these elements efficiently.

If a symbol is not contained within any of these tables, then the elements is considered unknown and the MathML extension mechanism is used to produce a reasonable translation.

IR to OpenMath Module

Producing OpenMath from the intermediate representation follows a similar procedure as that described for generation of MathML. The table `ir2om_mml!*` contains the function to call for each intermediate representation operator to produce OpenMath.

4.4 XML Lexing and Parsing

Because there are no XML lexers or parsers for REDUCE, it is necessary to design and implement them. In order to do so it is important to establish what the requirements of such procedures are.

4.4.1 The Lexer

Both MathML and OpenMath are based on the structures defined by XML. The lexer must validate XML markup languages and extract the necessary tokens from the successive characters in the input source.

Hence it is important that our lexer tokenizes XML elements as well as determining the different attribute types and values an element may possess. These requirements must be met in order to retrieve the different attributes contained in MathML elements or to find out what symbol and content dictionary is expressed by an OpenMath `<OMS>` tag.

An XML lexer must also be flexible with spaces, ignoring any amount of spaces or return carriages contained in the input source.

4.4.2 The Parser

The lexical analysis and the following phase, the syntax analysis, will be grouped together into the same pass. Under that pass the lexer operates under the control of the parser. The parser will ask the lexical analyzer for the next token whenever it needs one. The lexer will return this information as well as storing the attribute types and values of the current token parsed. The parser will not generate a parse tree explicitly but rather go to intermediate code directly as syntax analysis takes place.

The parser will stop its task when a syntactical error or a misspelled or unrecognized token is encountered. It should not attempt to correct it. In some cases a constructive error message[2] will be printed to the user.

The parser we will implement will follow the widely used LL(1) parsing method also known as *predictive recursive descent* parsing. The parser will use top-down parsing following the grammars defined in both the MathML and OpenMath standards and will only need to look at the next token in the token stream [11].

4.5 Possible Future Extensions

The desire to extend the OpenMath/MathML interface to include new functions or adapt to changes was paramount in the design process. Here we would like to mention some possible extensions which could be added in the future.

Evaluation of expressions: It should be possible to extend the interface to allow evaluation of OpenMath and MathML expressions using REDUCE's computational power. This extension is possible because the intermediate representation was designed imitating REDUCE's internal representation of expressions. Without difficulty a procedure could be implemented which would evaluate an intermediate representation expression by mapping it to REDUCE's internal representation. The appropriate modules would then print out the evaluated intermediate representation as MathML or OpenMath.

Separate Interfaces to REDUCE: For the same reasons as expressed above, it is possible to modify the interface so it offers a MathML to REDUCE interface and/or an OpenMath to REDUCE interface. This would allow a REDUCE user to import and export MathML or OpenMath expressions separately for use in calculations or for transmission on the Internet to other applications.

Interfaces to other Representations: Because the system architecture is designed around the intermediate representation, it is possible to implement modules which transform the intermediate representation into other representations such as LaTeX, TeX, HTML, or WEBTeX,

[2] The efficiency of this facility will depend on the time left to correctly implement it

thus allowing translation from MathML or OpenMath to any of the mentioned representations.

Chapter 5

Testing

In order to confirm that the task of interfacing MathML and OpenMath has been achieved, we must test the program in a variety of situations. We will be ensuring that all guidelines described in Chapter 3 are properly adhered to. The testing will also prove the efficiency of the overall design.

The most important feature to look for is that the translation process does not alter or modify the expressions being translated. Additionally, it is essential that all results produced by the translator are compliant to the OpenMath and MathML standards. We will focus on ensuring semantic value is preserved as well as making sure that CDs appear correctly next to symbols, MathML attributes are accurate, and that OpenMath symbols not handled are properly dealt with. Finally we will test MathML outputs with the widespread MathML renderer: IBM TechExplorer.

We will aim to test varios specific aspects in more detail. We will describe the testing method and the results obtained.

The testing should verify whether the program accomplishes its task of translating OpenMath to MathML.

5.1 Translation

In order to verify whether the translation process is attained, we have created a test suite comprising over 170 MathML and OpenMath examples, more than 80 of each. Most of these examples come from the standard specifications and others have been designed to test specific aspects of the translator. Many of these examples were extensively used throughout the implementation phase.

By running these examples through the translator, it was possible to carefully check if the output produced for each example corresponded to the expected result. This careful analysis, example by example, has proven that in most cases, semantic value was preserved and a proper translation was carried out. However, it is important to concentrate on the difficulties arising from translating OpenMath and MathML.

5.1.1 Content Dictionaries

Aim: Given that symbols from different CDs may have the same name albeit different meanings, we must test the ability of the translator to relate symbols to their CDs and recognize the difference in meaning different CDs convey.

Testing method: In the test suite of 170 examples, there are a set of expressions testing the ability of the translator to relate symbols to their CDs. These examples contain expressions with symbols from different CDs but with the same name, to test the translator's faculty to properly handle them. There are examples where correct symbols appear next to wrong CDs and vice versa. The translator should recognize only the valid CDs.

Results: All examples were correctly treated. The translator was capable of recognizing valid symbols by consulting the tables, translating them accordingly. The results produced by these examples were very satisfactory.

Aim: To test whether OpenMath expressions generated have symbols appearing next to the correct CDs.

Testing Method: Running all the examples contained in `examples.mml` will generate a large number of OpenMath expressions. By looking at them carefully we will see if symbols are correctly related to their CDs.

Results: All examples analysed were correct. The generated symbols appeared next to the appropriate CDs. However, whenever a MathML element mapped to a symbol which belonged to more than one CD (like *inverse* for instance), there was no rule determining which CD to place. Nevertheless, all translation were correct.

5.1.2 MathML Attributes

Aim 1: In many cases attribute values modify the semantic value of MathML expressions and must be taken into account when translating. We would like to determine if the translator detects attribute values and takes them into account when translating.

Aim 2: Various OpenMath symbols map to MathML elements with specific attribute values. Does the translator detect these symbols in order to generate MathML elements with their correct attribute values?

Testing Method: We must gather all examples containing elements whose attribute values convey semantic meaning. We translate these elements from MathML to OpenMath and see if the semantic meaning has been preserved. We then translate back to MathML and the translator will have recognized these OpenMath elements and reproduced the original MathML element with the correct attribute value.

Results: All results were satisfactory. Attribute values were recognized, correctly interpreted and semantic meaning was preserved in all cases.

5.1.3 Extensibility

Aim: To test how well the translator copes with OpenMath symbols having no equivalent MathML element. We want to determine if they are properly translated and preserve their structure and meaning.

Testing Method: There are various types of situations where extensibility mechanisms must be employed. These are:

- Both the CD and symbol are not recognized
- The symbol is not recognized
- The CD is not recognized

In these three cases, the translator must employ MathML `<semantic>` tags to preserve semantic meaning. We will tes expressions based on these three cases.

Results: The translator automatically detected the unknown symbols and enveloped them inside `<semantic>` tags. The translator worked well in cases where the OpenMath symbol translated was constructed by *application*. An example of this is the `rank` operator seen in figure 3.5. In other cases (such as *binding*) the results were poor and in many cases incorrect.

5.2 Standard Compliance

For this translator to produce usable results it must imperatively conform to MathML and OpenMath standards. This implies that all expressions produced must be lexically and syntactically correct according to the specifications. We must examine that the translator can parse and generate valid expressions.

5.2.1 Parsing of Expressions

Aim: Valid MathML and OpenMath expressions must be parsed without difficulty by the translator. Lexical and syntax errors must be detected and attribute values must be extracted for use in translating.

Testing Method: The translator should correctly parse a large amount of valid MathML and OpenMath examples taken from the standards. These examples are contained in the 170 examples mentioned earlier and are considered to be correct. The World Wide Web Consortium also offers a test suite for testing applications for MathML compliance. This test suite is a series of valid MathML expressions, which if parsed prove that an application is MathML compliant. Unfortunately the URL link to this test suite was broken throughout the duration of this project and has not been able to be used.

We will also introduce incorrect expressions to see if the translator declares them as erroneous.

Results: The translator performs well. All supported operators are correctly parsed. Syntax is validated in both standards, thus distinguishing amongst correct and incorrect expressions. Unsupported elements as described in section refnosupport cause the translator to abort. The translator is MathML and OpenMath compliant.

5.2.2 Generation of Expressions

Aim: To determine if expressions produced by the translator are MathML and OpenMath compliant.

Testing Method 1: In order to determine whether the OpenMath expressions are compliant, we shall introduce them back into the translator. If the translator correctly parses them then they are compliant. Additionally, we will check the expressions individually to ensure they are correct.

Testing Method 2: To determine if MathML expressions produced are MathML compliant, we shall translate them back to OpenMath. If the translator reads through them correctly then we can conclude the expressions are compliant. Furthermore, we will try and render the generated MathML expressions using IBM's TechExplorer. If TechExplorer renders all MathML expressions generated then we have more reason's to confirm that the MathML output is MathML compliant.

Results: The testing procedures all produced satisfying results. The translator's output can be parsed by itself validating the expressions. Furthermore, IBM's TechExplorer successfully parsed and rendered a large number of generated MathML1.0 expressions.

The results produced throughout these tests were mostly accurate, and when not, the translator was corrected. We recomend the user to run all the examples in `examples.om` and `examples.mml` to get a better idea of the translation efficiency. It is possible that mistakes have pased unnoticed. Reassuringly, the design is robust enough and most bugs should be quick to eliminate.

5.3 Interface Limitations

Testing of the interface has also revealed its limitations. Various aspects of the translation process have not been properly solved. It is important that we enumerate the areas where the translator performs poorly.

In section 3.3.2 we mentioned that some OpenMath operators encoded their bound variables within lambda expressions. We also said that this was not compulsory. The translator however only deals with expressions where the bound variable is within a lambda expression. Other cases cause the translator to abort promptly.

The analysis in section 3.2.1 discussed the importance of defining each operator's scope. The OpenMath/MathML translator gets confused when scopes are ambiguous and aborts.

MathML element `partialdiff` is not translated properly in most cases. Only when the variables of differentiation have an order of derivation equal to one. In all other cases the translator produces incorrect results or aborts.

The translator will reject MathML expressions containing operators defined within `<semantic>` tags. This is clearly something which will have to be implemented in the future, since the `<semantic>` tags may contain OpenMath code.

The translator is capable of distinguishing incorrect expressions from valid ones. Unfortunately there was not enough time to implement a constructive set of error messages. These error messages should have been able to quickly inform the user why a translation might have been aborted, or what problems there are with the input impeding tranlation.

Finally, there is an aspect of REDUCE which limits the interface. Contrary to XML, which is case sensitive as is stated in the XML standard [10], REDUCE is case insensitive. Consequently when translating an expression with variables or function names using capital letters, REDUCE will produce only small letters. This may in some occasions create confusion for the user or even distort the semantics.

5.4 Conclusion

Although 170 examples might not be enough to test the translator in all situations, they did demonstrate that the translator coped well with the difficulties of OpenMath/MathML translation. Because these examples are a representative selection of most OpenMath and MathML operators and situations, the satisfying results confirm that the task of accurate translating has been achieved.

Appendix A

CDs and Symbols handled by the Interface

alg1	`one, zero`
arith1	`abs, conjugate, divide, minus, plus, power, product, root, sum, times, unary_minus`
arith2	`arg, inverse, times`
calculus1	`defint, diff, int, partialdiff`
fns1	`inverse, lambda`
integer1	`factorial, gcd , quotient, rem`
interval1	`integer_interval, interval, interval_cc, interval_co, interval_oc, interval_oo`
limit1	`both_sides, above, below, limit, null`
linalg1	`matrix, outerproduct, scalarproduct, vector, vectorproduct`

linalg2	`vector`
linalg3	`determinant, matrix_selector, selector, size, transpose, vector_selector`
list1	`list`
logic1	`and, false, implies, not, or, true, xor`
logic2	`equivalent`
minmax1	`max, min`
multiset1	`in, intersect, multiset, notin, notprsubset, notsubset, prsubset, set, setdiff, subset, union`
nums1	`based_integer, complex_cartesian, complex_polar, e, gamma, i, imaginary, infinity, nan, pi, rational, real`
omtypes	`float, integer`
quant1	`exists, forall`
relation1	`eq, geq, gt, leq, lt, neq`
relation2	`approx`
set1	`in, intersect, notin, notprsubset, notsubset, prsubset, set, setdiff, subset, union`
stats1	`mean, median, mode, moment, sdev, variance`

transc1	arccos, arccosh, arccot, arccoth, arccsc, arccsch, arcsec, arcsech, arcsin, arcsinh, arctan, arctanh, cos, cosh, cot, coth, csc, csch, exp, ln, log, sec, sech, sin, sinh, tan, tanh
transc2	arccot, arccoth, arccsc, arccsch, arcsec, arcsech, arcsinh, arctanh
typmml	complex_cartesian_type, complex_polar_type, constant_type, fn_type, integer_type, list_type, matrix_type, rational_type, real_type, set_type, type, vector_type
veccalc1	curl, divergence, grad, laplacian

Bibliography

[1] World Wibe Web Consortium
`http://www.w3.org`.

[2] OpenMath Society
`http://www.openmath.org`.

[3] *MathML Standard Specification*, W3C Working Draft 28 March 2000
`http://www.w3.org/Math`

[4] O. Caproti, D. P. Carlisle, A. M. Cohen, *The OpenMath Standard*, August 1999
`http://www.nag.co.uk/projects/openmath/omsoc/`

[5] HTML+ Specification,
`http://www.w3.org/MarkUp/HTMLPlus/htmlplus_1.html`

[6] HTML 3.0 Draft
`http://www.w3.org/MarkUp/html3/CoverPage.html`

[7] D. Raggett *HTML 3.2 Specification*, W3C 14 June 1997.
`http://www.w3.org/TR/REC-html32.html`

[8] SGML Specification,
`http://www.w3.org/TR/1999/REC-html401-19991224/conform.html#h-4.2`.

[9] LaTeX2HTML Manual
`http://www-dsed.llnl.gov/files/programs/UNIX/latex2html/manual/Snodel.html`

[10] Bray, Tim, Jean Paoli and C.M. Sperberg-Mcqueen, *Extensible Markup Language 1.0*, 10 February 1998
`http://www.w3.org/TR/1998/REC-xml-19980210.`

[11] A. V. Aho, J. D. Ullman. *Principles of Compiler Design.* Addison-Wesley Publications, 1979.

[12] J. P. Bennet. *Introduction to Compiling Techniques.* McGraw-Hill Publications, 1996.

Important Web Ressources

[13] TeX4ht Web Site
`http://www.cis.ohiostate.edu/ gurari/TeX4ht/mn.html`

[14] TtM Seb Site
`http://hutchinson.belmont.ma.us/tth/mml/`

[15] WebEQ Seb Site
`http://www.webeq.com/webeq/`

[16] ICEBrowser Seb Site
`http://www.icesoft.no/ICEBrowser/index.html`

[17] IBM TechExplorer web site
`http://www4.ibm.com/software/network/techexplorer/`