Developing a Browser for Large Mathematical Expressions Given in Content MathML

Neil Chittenden

Bachelor of Science in Mathematics and Computing with Honours
The University of Bath
April 2009

Developing a Browser for Large Mathematical Expressions Given in Content MathML

Submitted by: Neil Chittenden

COPYRIGHT

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Batchelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifcally acknowledged, it is the work of the author.

Signed:

Abstract

Mathematical expressions can often be very large, and thus difficult to work with and understand, due to the large amount of information that they contain. It would therefore be useful to have a way in which to summarise, and then further analyse, the information in such an expression. This dissertation discusses the research and development of techniques to allow a user to interactively examine the structure of a large mathematical expression, from a content MathML[31] document, after first considering previous solutions to the problem.

The culmination of this project is an application demonstrating some key techniques used in displaying a large mathematical expression. These include summarising an expression using substitution of subexpressions, ellipses, and an alternate view of matrices. In addition to providing such elisions, a way of processing MathML documents that are possibly too large to be loaded into memory has been achieved. The technique developed for large MathML documents may be useful for applications using parts of large XML documents in general. Furthermore, interactivity with the expression is achieved with a technique for mapping user interaction to the original source document.

This project provides a firm base for future work into producing a complete browsing solution for a content MathML expression.

Contents

List of Figures

Acknowledgements

I would like to thank my project supervisors Dr. E. Cliffe and Prof. J. Davenport for all their support and guidance throughout the year. Also, thanks goes to my friends and family for all their useful comments.

Chapter 1

Introduction

1.1 Problem Description

Computer algebra systems, such as Maple[15] and Mathematica[45], commonly give rise to large mathematical expressions. Although it may be possible to render such expressions, they can be difficult for a user to understand due to the sheer amount of information that is presented. Figure 1.1 demonstrates this in Maple. Figure 1.2 shows how Maple can be configured to elide an expression, however, interaction with this elided expression is not user-friendly as it is restricted to typed commands or changing the global options. Some large mathematical expressions may also be too big to render. e.g. A 1000×1000 matrix could easily exceed 1GB in size when rendered.

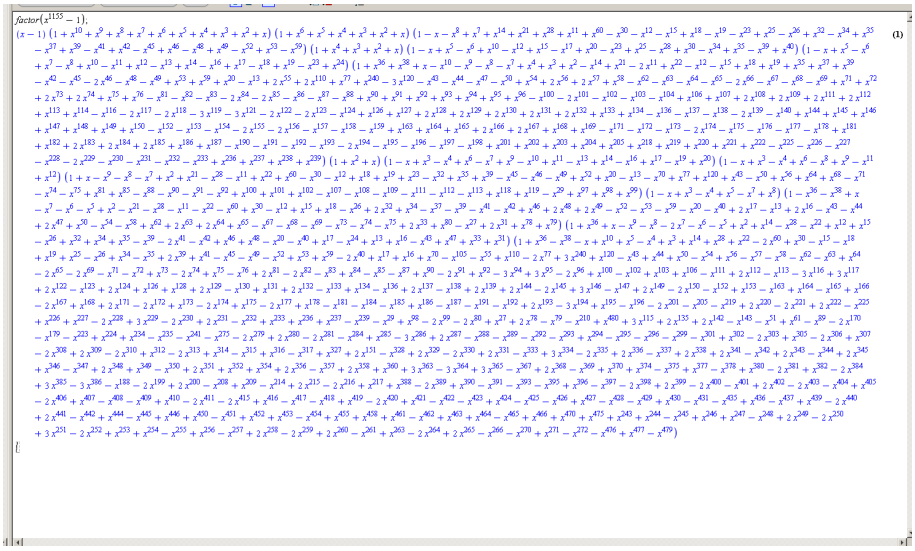

Figure 1.1: Screenshot of large expression in Maple[15]

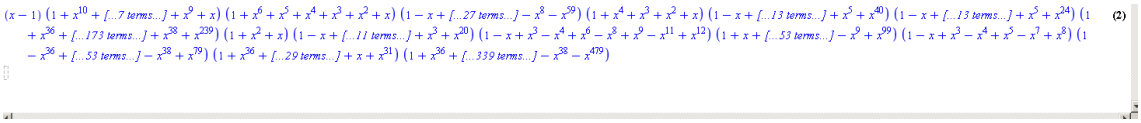

Figure 1.2: Screenshot of elided expression in Maple[15]

A better solution than just rendering an entire expression is to only display parts of it by performing some kind of elision on the mathematical content; that is, replacing parts of it with shorter, even if incomplete, sub-expressions. Performing such replacements then requires some way of expanding this shorter sub-expression to the original so that no information is lost. Our initial research on techniques in splitting up mathematical expressions only led to basic elision of multiplication symbols and brackets, which, although not trivial, is beyond the scope of the main problem. Also, how certain parts of mathematical

expressions, such as sum indices (Σ rather then $\Sigma_{i=1}^{n}$), are often elided is discussed in [13]. Since we are more interested in examining the structure of a mathematical expression, such elisions will not be considered. Some elision algorithms for different mathematical entities are therefore required to be created for this project.

Matrix elision in particular is an interesting area for this project and it will need to be decided what simplified forms matrices can take. This could involve checking if a matrix is in a recognisable form such as the zero matrix ($0_{n\times n}$), the identity matrix ($I_{n\times n}$) or a more complicated form such as a block matrix:

$$\begin{pmatrix} A_{m\times n} & B_{m\times p} \\ C_{q\times n} & D_{q\times p} \end{pmatrix}$$

Another format a matrix may take is having a common format where it is not necessary to display all the terms but instead makes use of ellipses. e.g.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \equiv \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 1 \end{pmatrix}_{4\times 4}$$

The process we may be interested in for this project for converting a "complete" matrix into a "Textbook-style" matrix is opposite to the method described in [19] of reading in a "Textbook-style" matrix for a computer. There are many formats that a matrix can take, so it will be necessary for the application to have a modular design so that checks for new formats can easily be added.

Large mathematical expressions produced by computer algebra systems, such as Maple or Mathematica, can usually be exported from the system in a markup language for mathematical content. The Content MathML (MathML-C) recommendation [31] from the World Wide Web Consortium (W3C) and the OpenMath standard [24] from the OpenMath Society are two widely supported markup languages that will be considered for this project. Since both MathML-C and OpenMath are XML-based languages, and there are XSLT stylesheets that can be used to convert between the two formats, it is sufficient for this project to only consider one of them.

Another problem to consider is the relation of the elision algorithms to the screen-width available to the renderer. A simple solution may be to have a fixed scrollable area to render on but a better solution may be to have more/less strict elision rules depending on the window size.

The purpose of this project is to design and implement an application that allows a user to browse, and thus make sense of, a large mathematical expression given in MathML-C and/or OpenMath. The user interface will initially provide a simplified view of the mathematical expression using some elision algorithms. Sub-expressions formed will then be expandable if the user wishes to see more information about a particular area. This

will enable a user to observe as much information about the expression as they require and make it easier to understand.

The application will need a way to render the mathematical content markup. Although there are a lot of applications for rendering and editing MathML, there are few free libraries for doing so. A possibility could be to create the application for a web-browser that has MathML rendering capabilities by basing it on JavaScript. This would pose several problems such as different rendering on different web-browsers although may be advantageous for setting up clickable MathML. A better solution would be to use the JEuclid [3] MathML rendering libraries for a Java Swing UI to produce a standalone Java application.

1.2 Objectives

This project aims to investigate and develop techniques in order to allow a user to interactively examine the structure of a large mathematical expression. The following statements summarise the project's main objectives.

- Research common simplications of mathematical entities.
- Create elision algorithm(s) for basic mathematical entities (e.g. sums and products).
- Investigate how advanced mathematical entities such as matrices can be represented.
- Investigate processing techniques of large XML documents.
- Develop a GUI to read in the MathML-C and/or OpenMath XML document and render a more simple summarised mathematical expression.
- Through the GUI, allow the user to interact with the summarised expression so that they can examine its structure.
- Test the elision algorithms created using several large mathematical expressions.

1.3 Outline

The following sections describe the project and how the main results were achieved. In particular, a thorough, in-depth literature survey is carried out in order to investigate existing systems, technology and techniques available (see chapter 2). After identifying key literature, the functional requirements of the application are summarised leading on to the design in chapter 4. Implementation details are then discussed in chapter 5. These sections then lead onto the main results of the project (chapter 6). Finally, the conclusions are discussed, with the entire project being examined (chapter 7).

Chapter 2

Literature Survey

The aim of this project is to develop a piece of software that can provide an interactive user-interface for browsing large mathematical expressions. In order to have a precise idea of what key features need implementing, and the ways in which they should be implemented, research into existing software and work in related areas is required. The two main areas of the problem are elision of mathematical expressions and the technology available.

As already mentioned, one of the main parts of the problem is elision of mathematical expressions. It will therefore be very useful to closely look at techniques and algorithms that already exist in this area. Analysing these techniques and/or developing new techniques will be key to this project.

Existing technologies that can be used to render a mathematical expression given in Content MathML (MathML-C)[31] or OpenMath[24] will also need to be looked into. The main purpose of the technological research will be to avoid 'reinventing the wheel' and thus selecting the most appropriate tools for the task in hand.

The aim of this research is to come to strong conclusions about the best way in which to implement this project.

2.1 Displaying Large Expressions

Definition. Elision is *the act or an instance of omitting something.*[35]

Displaying a large mathematical expression is identified as one of the main problems of creating a user interface for a Computer Algebra System (CAS) [12]. This is due to the large amount of information that can be produced by a CAS often being confusing to a human user when presented in its 'complete' form.

The research carried out by Soiffer[22] and Tyhurst[26] into the problems involved with producing a user interface for a CAS suggest several techniques on handling the display of large expressions. Tyhurst also remarks regarding the difficulties of the interface of a CAS not knowing any information about the primary interests of a mathematical expression. This means that the display ideas in designing such an interface can be directly related to this project (since the mathematical expressions read will also contain no such information).

The two areas that both Soiffer and Tyhurst identify are simplifying the visualisation for the user, and working with the display limitations (i.e. screen-width). These are:

- User-related
  - Use of ellipses to replace terms
  - Renaming a subexpression
  - Satellite view and fish eye view
- Display-related
  - Scrolling

- Line breaking
- Line cutting

They both also provide little, if any, information on dealing with matrices in particular, although the ideas mentioned can be extended for this case.

Soiffer also comments on early versions of MathScribe having an automated elision mechanism, which was subsequently dropped as a feature due to users not finding it very useful, instead having a user-controlled mechanism. While this may be true for smaller 'large' expressions, a much larger 'large' mathematical expression may then require a lot of effort to elide down to something comprehendible. e.g. trying to elide down a summation consisting of 1000 terms (ungrouped) to just the first and last terms. Although automatic elision will be useful, it is clear that having some user interaction to control custom elision is also necessary.

2.1.1 Use of Ellipses

Adding ellipses to elide an expression can be useful for summarising and reducing space. The following mathematical expressions demonstrate ellipses:

A summation

$$1+4+5+3+2+5+3 \text{ may become } 1+\cdots+3 \tag{2.1}$$

A function

$$f(x_1,x_2,x_3,x_4,x_5) \text{ may become } f(x_1,\cdots,x_5) \tag{2.2}$$

A matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix} \text{ may become } \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 4 \end{pmatrix}_{4\times 4} \tag{2.3}$$

The elided forms, such as the summation above, may reduce display space required but will lose some of the original mathematical meaning and/or may change its mathematical meaning. The summation with the ellipsis could be misconstrued as a sum from 1 to 3 so it is important that the user understands what's happening. A feature, to help clarify elisions, implemented in MathScribe[20] was to have a number above the ellipsis displaying to a user the number of subexpressions elided.

Simple Elision Algorithm

```
if nTerms >= eThresh then
```

```
  /* show first and last terms separated by an ellipsis */
  ...
else
  /* show all terms */
  ...
endif
```

Where `nTerms` is the number of terms within an expression relating to a particular mathematical operator and `eThresh` is the pre-defined elision threshold. In order for full usability of such a feature, `eThresh` should be user-customisable.

An additional option that may be useful to a user would be the possible 'shifting' or 'contracting' of an ellipsis so that more information about the elided expression can be deduced without having to expand it completely. e.g. $1+\cdots+3$ in equation (2.1) becomes $1+4+\cdots+3$ after shifting the ellipsis to the right, allowing the user to see an extra term.

2.1.2 Use of substitutions

Using a substitution in a mathematical expression is an obvious way of reducing the space used for it to be displayed. The following expressions demonstrate substitutions:

Nested functions:

$$f_1(f_2(f_3(f_4(f_5(f_6(x)))))) \tag{2.4}$$

may become

$$f_1(f_2(f_3(F))) \tag{2.5}$$

where

$$F := f_4(f_5(f_6(x))). \tag{2.6}$$

Matrices:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix} \tag{2.7}$$

may become

$$AB \tag{2.8}$$

where

$$A := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, B := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}. \tag{2.9}$$

In practice, the subexpression to be renamed may be very large, so identifying the substitution as in equation (2.6) would not be offered to the user without a request. Also, any automated substitutions will have to be clear so that the user does not confuse the substitutions with the original expression; a naming check or predefined naming convention.

In determining whether the depth of an expression's tree of nested functions is significant enough to perform an automated substitution, it will be important to avoid the overhead of calculating the full depth since this may be very large (or even infinite).

Algorithm to decide on whether to perform a nested expression substitution

```
boolean IsDepthBelowThreshold(TreeNode node, int dThresh)
{
  if(dThresh == 0 ||
     (dThresh == 1 && node.hasChildren())) {
    return false;
  } else if node.hasChildren() {
    for each childNode in node.Children {
      if( ! IsDepthBelowThreshold( childNode, dThresh - 1 )
        return false;
    }
  }

  return true;
}
```

Both performing a substitution and the use of ellipses (§2.1.1) could be used interchangeably to reduce the display space of an expression. Although this is the case, such usage may not be clear to the user. As with the use of ellipses, it would be useful to allow the user to specify a particular substitution to perform. Also, when performing any substitutions, it would be useful to replace each occurrence of the renamed subexpression. As noted in §4.3.3 of Soiffer (he calls this 'renaming'), comparing subexpressions can be implemented with a depth-first search of an expression's tree.

The problem with trying to do this in practice is when the nested depth of an expression is very large (or even infinite). Since expressions can only be verified as being equal if they match throughout their entire trees, it would be inefficient to perform such a check unless requested by the user. (A fully balanced binary tree would need 2^{depth} comparisons for equality to be true, so for n-ary functions this number could quickly become unmanageable).

2.1.3 Alternate views of an expression

Tyhurst mentions the use of:

Satellite view
Rendering the entire expression into a small area so that the user can get an idea of the 'shape' of the expression.

Fish-eye view
Whilst the user is exploring the expression, only render nearby subexpressions - based on a degree of interest algorithm.

These views may be useful depending on the way in which a user wants to browse a mathematical expression but would require advanced control of its rendering. Such control using a third-party mathematical expression renderer may be unavailable or very difficult and, thus, such views will not be considered.

2.1.4 Matrices

Since matrices are two-dimensional objects, they don't lend themselves to an obvious linearisation. Depending on a person's interpretation, they might want to read a matrix by rows or columns or even in some other way. For this reason they will need to be considered separately from other mathematical expressions. Another problem with deciding how to represent matrices in particular, are the many different forms of notation used. Some examples of how the matrix in equation (2.3) may be represented:

- As in equation (2.3) with ellipses.
- Renamed as in equation (2.8).
- Represented as a diagonal matrix function:

$$diag(1, 2, 3, 4)$$

- Represented as a block (diagonal) matrix [36]:

$$\begin{pmatrix} C & 0 \\ 0 & D \end{pmatrix}$$

A user may also find it useful to have a feature that will display a summary of the matrix. i.e. the matrix could be of the following special forms: a block diagonal matrix, diagonal matrix[37], identity matrix[38], upper or lower triangular matrix[43], symmetric matrix[42], skew-symmetric matrix[40], etc...

For the different representations of matrices, the order is not necessarily required for the substitution or use of ellipses methods described, but for all other analysis of a matrix, the order is needed. This means assuming that the matrix order will be an acceptable size to carry out computations on each element. There can also be some computation saved by

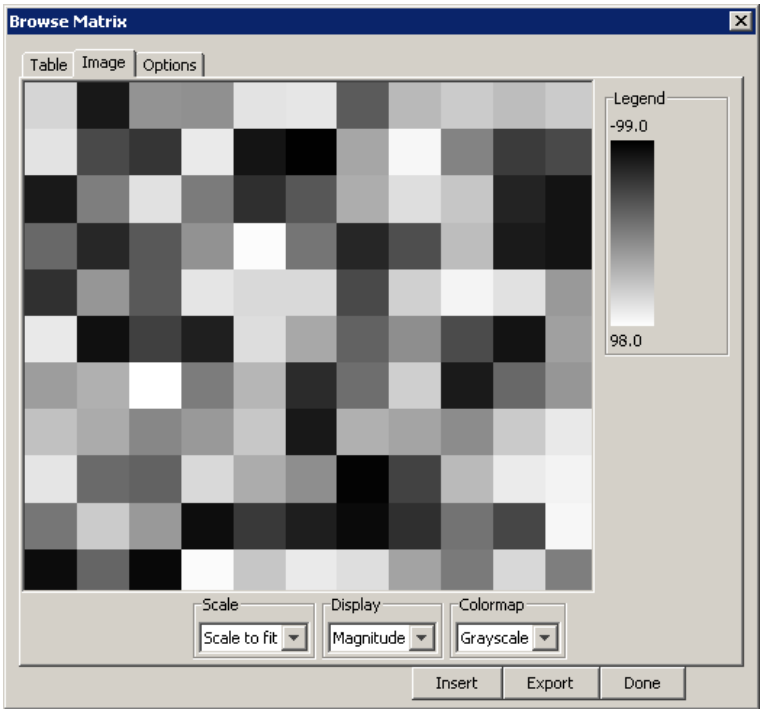

Figure 2.1: Bitmap view of randomly generated 11×11 matrix in Maple[15].

relationships between styles of matrices. e.g. the identity matrix is also a diagonal matrix, which in turn is also a block diagonal matrix.

Another way to represent a matrix, if it is very large, could be to provide a bitmap view of its entries. This feature is implemented in Maple[15] when the order of a matrix gets larger than 10 as seen in Figure 2.1.

Whereas Maple uses a scale for each number in the matrix, a more general display solution, that would give basic information about the matrix, would be to use colours for its key entries. i.e. different colours for zero elements, constants, and variables/subexpressions, since these are what the user will normally be interested in. This information will be enough for the user to come to basic conclusions about the matrix before investigating it further, as well as being easy to compute (since no prior knowledge of the range of values in the matrix is required). An option to produce a similar layout to Maple's could be added, but this would have an overhead of initially finding the range of values within the matrix.

Although not specifically aimed at matrices, Walmsley[34] has some interesting ideas about the representation of two-dimensional data structures. In §2.3 of Walmsley, he discusses navigation techniques of large 2D data structures including using a map window (moving a small rectangle representing the display area over a larger rectangle representing the entire image) and the previously mentioned fish-eye view. In combination with the bitmap view of a matrix, a map window could be useful for very large matrices although the issue of controlled rendering would arise. §3.2.5 and §3.2.6 of Walmsley also provide some interesting uses of eliding and hiding data but in general the thesis is largely unrelated to this project and heavily focused on Nial.

2.1.5 Display Limitations

Tyhurst[26] states:

> Despite the best efforts to reduce the quantity of information to be presented when displaying a mathematical expression, expressions must frequently be rendered in their entirety.

This means ways of handling cases when the mathematical expression becomes too large for the display must be considered. Without handling display limitations, the mathematical expression would appear truncated and not be useful to the user.

Scrolling

The simplest way in which to handle a mathematical expression that renders larger than the display area is to provide a method of vertical and horizontal scrolling. Its main advantages are:

- It is easy to implement, due the renderer not requiring extra processing.

- It is intuitive to use since there is no ambiguity introduced into the expression.

The main disadvantage of scrolling is an inefficient use of the display area for a wide but not tall expression (as is often the case in mathematics).

Line cutting

Line cutting is another way an expression could be handled and would involve splitting a rendered expression into screen-width chunks. Unlike line breaking, line cutting doesn't perform any intelligent splitting of an expression, any part of the expression which goes off-screen simply overflows onto the next row. Its advantage is a more efficient use of display area than scrolling for a wide expression. The disadvantages of this method are:

- Ambiguity in the mathematical expression could arise. For example, if an important part of a square root or fraction was cut and moved to the next line.
- Detecting user interaction with the expression is complicated if it's split across lines.
- Not better than scrolling if the width of a mathematical expression is marginally greater than the display width.

Line breaking

Line breaking is the process of splitting the expression across lines intelligently using operators so that the mathematical expression does not suffer from the ambiguity that might occur from line cutting. Structural line breaking is similar but each term of an expression appears on a different line indented to a different degree, producing a tree like structure. (See Figure 4.15 of Soiffer[22]). The disadvantages of line breaking are:

- Detecting user interaction with the expression is complicated if it's split across lines.
- If the subexpressions are sufficiently large, then it will suffer from width problems like scrolling.
- Requires more processing of expression during rendering.
- Structural line breaking can waste a lot of display space due to its tree-like structure.

Further remarks

The display limitations will arise from the renderer chosen for this project. Line breaking will only be able to be used if the renderer supports it since processing an expression to find 'intelligent' breaks needs to be carried out at render time.

2.1.6 Elision Searches

Finding material relating to elision of mathematical expressions proved very difficult. The reason for this could be that there is no concrete definition of elision in a mathematical context making material on the subject hard to come by. Another reason for this may be that elision is sometimes context-specific making it difficult to generalise. Or, more simply, the concept of eliding a mathematical expression may be thought of as trivial for a human (though this is not the case for computers).

In particular, the subject of matrix elision provided scarce background reading. Searching with terms involving combinations of the phrases "matrix elision", "computer algebra", "matrix simplification", "displaying large matrices" in several different locations [1] provided very few related articles.

2.2 Technology

The main technological features of a mathematical expression browser are reading in the expression, rendering the (elided) expression, and user interaction with the rendered (elided) expression. Since technologies already exist to carry out these aspects of a mathematical expression browser, they will be investigated to avoid implementation from scratch.

Since both MathML-C and OpenMath are XML[32]-based languages, and there are XSLT stylesheets that can be used to convert between the two formats [17], it is sufficient for this project to only consider one of them; MathML-C. The reason for this choice stems from initial research showing that there are active rendering tools available as opposed to OpenMath (only found inactive JOME[11]) and also our more familiarity with MathML-C.

2.2.1 MathML

This section describes the basic structure of MathML-C and how it can be manipulated to present elided expressions.

Example 2.2.1. *MathML-C representation of the expression* $1 + c_1 + c_2$ *(exported by Maple).*

```
<math xmlns='http://www.w3.org/1998/Math/MathML'>
  <apply>
    <plus/>
  <cn type='integer'>1</cn>
    <apply>
      <selector/>
```

[1] Places searched were the University of Bath library catalogue, Electronic Library Information Navigator, SpringerLink, Google Scholar, O'REILLY Open Books, and TechXtra.

```
      <ci>c</ci>
      <cn type='integer'>1</cn>
    </apply>
    <apply>
      <selector/>
      <ci>c</ci>
      <cn type='integer'>2</cn>
    </apply>
  </apply>
</math>
```

From Example 2.2.1, it can be seen that a basic MathML-C expression consists of an `apply` element that represents a function (`plus`) being applied to several arguments (the following n siblings), depending on the function. Each argument can also be another function application, represented by the further `apply` blocks.

A simple example of how an ellipsis may be used in an expression would be to replace the first inner `apply` block with `<ci>...</ci>`. Since all MathML-C elements support the `class`, `style`, `id` and `other` attributes, we may want to store additional information in these to help with the document processing; most suitable would be `other` since the other attributes are meant for usage with CSS. Similarly, if using a replacement symbol in an expression, we may want to make it clear that the symbol was not part of the original MathML-C document with some additional information.

All n-ary constructor 'container' MathML-C elements (see §4.2.2 of the MathML specification[31]) will need to be considered for automatic elision even though the most common will be the `apply` element. `list`, `set`, and `vector` can all be processed in a similar way to `apply`, whereas, `lambda`, `piecewise`, `matrix` and `matrixrow` will need special consideration.

- `lambda` takes $n-1$ bound variables with the last child element being the expression that the λ-function represents. This means $n-1$ variables before the last child element should be considered for elision. The elision would have to work by replacing the elided bound variables with `<bvar><ci>...</ci></bvar>` since `lambda` cannot take `<ci>...</ci>` like `apply`. Though, since this MathML-C element defines a λ-function, performing any elisions may result in changing the meaning of the function, therefore it should not be elided.
- `piecewise` takes n `piece` elements and an optional `otherwise` element (no ordering is required). The standard rendering for such an expression is with the `piece` elements in order before a final `otherwise` element if it is present. Elision of the `piecewise` will therefore need to be careful not to remove an `otherwise` element if present. Also, similarly to `lambda`, a special case is required to replace any elided `piece` elements; `<piece><ci>...</ci><ci>...</ci></piece>`.
- `matrix` and `matrixrow` would elide as expected; replacing the elided rows/columns with rows/columns of ellipses.

When considering elision based on the depth of a mathematical expression, it should be noted that not all of the MathML-C elements will affect the rendering of the expression. e.g. `bvar`, `declare`, `csymbol`, `semantics`. Also, `annotation` and `annotation-xml` should be ignored in all processing.

2.2.2 Rendering

The following section describes possible rendering solutions.

JEuclid[3]

JEuclid is an open source project for rendering both content and presentation forms of MathML and it can be integrated into other Java applications as a Swing or AWT component. It also has an active mailing list[4] for usage and support queries. Its main disadvantages are that it does not currently support automated line breaking and the information about the rendered MathML objects are encapsulated within the core library. Since it is open source, modifying the source code to allow access to the necessary objects is a possibility. JEuclid's own DOM (see §2.2.3) would lend itself to mapping its graphical information to the source MathML.

JOME[11]

JOME is an open source Java software component for editing and displaying OpenMath and MathML. Unfortunately, this rendering solution could not be investigated further since the project website is down and the project assumed discontinued.

Web browser integration

Another rendering solution could be to use the MathML rendering capabilities of a web browser. e.g. Firefox[16] can render presentation MathML when formed in an XHTML + MathML XML document. Firefox also features automatic line breaking of the rendered expression. Since Firefox allows selecting of parts of a rendered MathML expression (similar to selecting text), it lends itself to performing interaction with the rendered expression (perhaps with the help of JavaScript). Initial attempts of trying to use MathML with JavaScript hyperlinks in Firefox however, proved difficult.

Other rendering tools

GtkMathView[1] The GtkMathView C++ rendering tool for MathML will not be considered since it is platform specific to GNOME on Linux systems. Also, GUIs in C++ tend not to lend themselves to cross-platform compatibility.

WebEQ[10] WebEQ is a Java toolkit for building interactive web pages containing MathML equations. It is also scriptable giving it a lot of flexibility. Unfortunately it is not freely available and will not be considered for this project.

MathML Renderer[21] MathML Renderer is a C# MathML rendering library. It won't be considered since it is not freely available and C# is (fairly) platform specific to Microsoft Windows.

2.2.3 Parsing

Since the mathematical expression will be given as a MathML-C XML document, the way in which the document is read into the program needs to be considered. There are several solutions to parsing an XML document; using DOM[27], SAX[6], StAX[41], VTD-XML[47].

Document Object Model (DOM[27])

> The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page.

Although the specification does not require any particular data structure, DOM is typically implemented as a tree of the XML document. This provides a natural way of thinking and navigating the XML. Due to the navigation ability of DOM, to any XML node within the source document, it is particularly well-suited for random access such as with XPath. This makes relation to user interaction particularly simple since editing MathML to be rendered can be done by editing the DOM tree.

In order to enable random access to any XML nodes in the source document, DOM must form a representation of the entire source XML document. This presents issues with large memory usage for large XML documents that would not be seen with serial access to the document (SAX). i.e. an XML document that is larger than the machine's memory capacity could never be represented with DOM. With modern machines commonly possessing large amounts of RAM ($\geq$ 2GB), this is unlikely to occur, however, a much more memory efficient solution needs to be considered (since this is the aim of the project).

Since it is very popular, there are many existing DOM XML parsing solutions available for most programming languages, notably JAXP[2] and Xerces[23] both contain a DOM parsing interface.

Simple API for XML (SAX[6])

SAX is an event-based API used to process an XML document as a stream. Rather than constructing an internal tree structure of an XML document like the DOM, parsing events are reported to the application, which is then responsible for implementing handlers for the different events raised.

The advantage that a SAX-style parser has over a DOM-style parser is in memory usage. Whereas a DOM parser is required to construct a tree structure of the entire document in memory, a SAX parser only needs enough memory to store the data associated with the XML tree of the XML element it is currently processing. This will always be less than the entire document.

The main disadvantage of a SAX-style parser is when access to the full XML document is required, such as for processing XML with XSLT and XPath; see [39] for further information. Since random access to XML elements will be needed when the MathML document is associated with user interaction, using a SAX parser would not be suitable; it would have to parse the document for each user interaction. However, an extension to a SAX parser that remembered key XML element file positions (such as where automatic elisions occurred) may be more useful.

Similarly to the DOM, JAXP and Xerces also contain a SAX parsing interface.

Streaming API for XML (StAX[41])

Similarly to SAX, StAX is an event-based API used to process an XML document as a stream. Whereas a SAX parser 'pushes' data to the application, a StAX parser is used to 'pull' data to the application. i.e. the application decides when to process the next event from the parser rather than processing each event as the parser raises it.

StAX provides similar advantages to SAX over DOM, in terms of memory usage, but also suffers from the disadvantage of not enabling random access to the XML document.

JAXP and Xerces also contain a StAX parsing interface.

Virtual Token Descriptor for eXtensible Markup Language (VTD-XML[47])

VTD-XML is a collection of XML processing tools centred around a document-centric, non-extractive XML parser. That is, it keeps the source document unchanged, and uses a 64-bit VTD record[46] to represent a token in an XML document. It therefore only thinks of the XML as syntax and does not use the traditional object-oriented approach of DOM.

By not using the object-oriented approach as DOM does, and only using primitive data types, it is more efficient in terms of speed and memory usage [44], whilst still retaining random access to the XML document. Although it is more efficient than DOM, it still requires storing typically $1.3 - 1.5\times$ the size of the document in memory when offering the

similar functionality - this means very large XML documents still pose a memory problem. Extended VTD-XML claims to support full XPath queries on XML documents up to 256GB in size [48]. This would remove any memory issue, although it is unclear the efficiency cost in terms of time of processing compared to the standard version.

2.3 Summary

2.3.1 Interface/Features

Based on the discussion of displaying large mathematical expressions in §2.1, the following conclusions have been made about the user interface features.

Automated elisions will be performed on a mathematical expression replacing subexpressions with either ellipses or another variable based on user options. The thresholds for eliding an expression based on width (number of arguments) and depth of the expression will also be user customisable options. Any elision performed should also be easily recognisable, i.e. highlighting.

Interaction with the elided expression should then give the user options of uneliding subexpressions, shifting/expanding elisions (across the expression or up and down the expression tree), and performing custom elisions based on selection. For matrices, as well as performing some automated elision, a function similar to Figure 2.1 should be available to the user.

Finally, the possibility of exporting the expression that has been produced by the expression browser, as an image or MathML, would also be useful.

2.3.2 Technology

This section describes the conclusions made about the technology available.

An important target for the development of this software project is to ensure cross-platform compatibility. Designing the mathematical expression browser in such a way will make it accessible to the greatest possible user base. For this reason, and the wide availability of tools, the project will be implemented in Java.

Rendering

Since JEuclid will allow creation of a standalone application, it will be used as the rendering component of the mathematical expression browser along with the reasons described above. Also, with graphical information extracted from the source code, it may be possible to also offer other useful display options such as line breaking. If this route does in fact prove too complicated, the web browser integration rendering solution will be considered as a backup strategy.

Parsing

There are many implementations of the XML document processing techniques described, available in several different languages as described in §2.2.3. XOM[9] is an example of another XML parsing solution, which implements a dual streaming/tree-based API to minimise memory usage, and is based on a lot of the technologies described.

To avoid a large memory overhead from large XML documents, DOM cannot be used to read in the original MathML document. However, since JEuclid renders from a source MathML document (or its own DOM), it will be necessary to produce a preprocessed (elided) version of the source document to pass on to JEuclid. This means that the preprocessing (parsing) method chosen must remember where each elision occurs so that it can extract further information from the MathML tree as required by user interaction. Using XPath expressions for storing this information, and then browsing the source document as required, would work well with the majority of XML parsing techniques. Efficient, low memory usage parsing solutions, such as (extended) VTD-XML or XOM, are therefore expected to be the most useful. However, some tests of the document parsing/processing techniques we have described are necessary to analyse their performance on large documents, particularly their memory usage.

An issue with the parsing methods described is that they cannot handle the possibility of infinite depth XML trees within the source document. Since it is also unlikely that this case could occur, this project will instead consider large finite XML documents.

Chapter 3

Requirements

In this chapter, we discuss the requirements of the project application based upon the results of the literature survey. Since there are no similar existing systems, only a few key features will be targeted. The three main areas identified for the application are summarising a large mathematical expression, interaction with the summarised expression, and efficiently processing the source content MathML document.

3.1 Functional requirements

3.1.1 Reading a content MathML document

The application must be able to load in a content MathML document, specified by the user, for processing. This is fundamental to the application so that different expressions can be analysed.

3.1.2 Processing a large content MathML file

The application must be able to efficiently process a large document that may be larger than the system's memory capacity.

Satisfying this requirement, so that processing can be carried out fast enough for user interaction, is expected to be a particularly challenging area. At this stage, as identified in the literature survey, the underlying document processing technique that we should use cannot be decided without further analysis. For this reason, we need to run some performance tests (see §4.3) to conclude on an appropriate solution to process the document.

3.1.3 Elision

Perform some elision algorithms on the mathematical expressions.

Due to the absence of sufficient literature on methods of actually performing elisions, we will need to create our own elision algorithms. We expect that these elision algorithms will largely depend on the method of document processing. The two main elisions identified for long linear mathematical expressions (see §2.1) are on the width of an expression using an ellipsis, and performing substitutions.

Since implementing special elisions of matrices will add to the complexity of this project, a compromise has to be made by restricting the development in this area. For this reason we will restrict matrices to being represented initially by a placeholder which can be interacted with to produce a bitmap view, similar to figure 2.1. If there is time, further development into this area by perhaps looking at a view such as in equation 2.3 can be carried out.

3.1.4 Rendering

The application should be able to render the (elided) mathematical expressions.

This will require using an external MathML rendering library. Implementing our own rendering library is well beyond the scope of this project since there are available solutions already, as detailed in §2.2.2. By using an external rendering library, we are limited to the features it implements. This will mean that the application will depend on advancements of the rendering library for future versions of MathML. Also, any line-breaking functionality will require support from the renderer since this processing must be done at render time.

3.1.5 User Interaction

After an elided expression has been rendered, a user of the appication should be able to interact in some way to expand any subexpressions.

To be more user-friendly than the Maple elision, pictured in figure 1.2, we should allow the user to use the mouse to interact with the elided expression; a much more intuitive way of browsing than simply typing commands. This is another key area that is expected to be challenging because we require a mapping from the mouse click to the large source MathML-C document.

3.1.6 Exporting

A user of the application should be able to save the summarised expression.

3.1.7 Testing

The application should be thoroughly tested. This will require multiple content MathML documents to test the above requirements.

3.2 Non-functional requirements

- The application should be platform-independent. For this reason, Java will be used to implement the application with the target platform being Java Runtime Environment version 6. This will allow us to target the largest possible user-base.
- The GUI should be easy to use. An intuitive user interface is important to allow the application to be used efficiently.
- The code should be maintainable including suitable commenting.

3.3 Resources Required

The following resources will be required for this project:

Maple
Maple [15], a computer algebra system, will be useful in producing sample expressions to work with. This will be particularly useful in testing the elision algorithms on different expression cases. Maple is already installed at the University of Bath.

Java SDK
Since the application will be implemented in Java, the Java SDK will be required.

Subversion
Subversion[25], a version control system, will be used to safely store and manage the source code. The University computing services provide online-accessible Subversion repositories so no additional costs will be incurred.

Hardware
No particular hardware will be required since there are many Java tools for different operating systems.

Chapter 4

Design

The purpose of this chapter is to detail the structure of the application and explain key decisions in the design process. As already mentioned in §3, the three main areas identified for the application are summarising a large mathematical expression, interaction with the summarised expression, and efficiently processing the source content MathML document.

4.1 Basic operations

4.1.1 Program Flow

Figure 4.1 shows the basic flow of the program from when a document is loaded and what happens on subsequent user interaction.

4.1.2 Document processor

Appendix B contains the document processor Java interface.

4.2 User interaction

As decided in the literature survey, JEuclid will be used as the rendering component of the application. Since JEuclid stores the information used for rendering but it is not currently accessible, some modification to the JEuclid source is required to retrieve this information. After examining how JEuclid renders MathML-C, it actually initially transforms it to MathML-P before rendering. This means, internally, JEuclid does not currently store any information for mapping the mouse position back to a MathML-C document; only to a MathML-P document.

Since other renderers only support MathML-P anyway, this does not affect the choice of renderer for this application. To overcome this limitation of JEuclid (and any other MathML-P renderer), a way of mapping the rendered MathML-P back to its corresponding MathML-C needs to be considered. An alternate approach would be to modify JEuclid in the way in which MathML-C is handled. This will not be considered since it is expected to require a lot of work and would most probably require the MathML-P to MathML-C mapping to be considered at some point.

In addition to the MathML-P to MathML-C mapping, we may also want to store some additional information along with the rendered expression. This information would allow us to decide upon the user interactions available. e.g. this could be whether a particular part of a rendered expression is clickable or not.

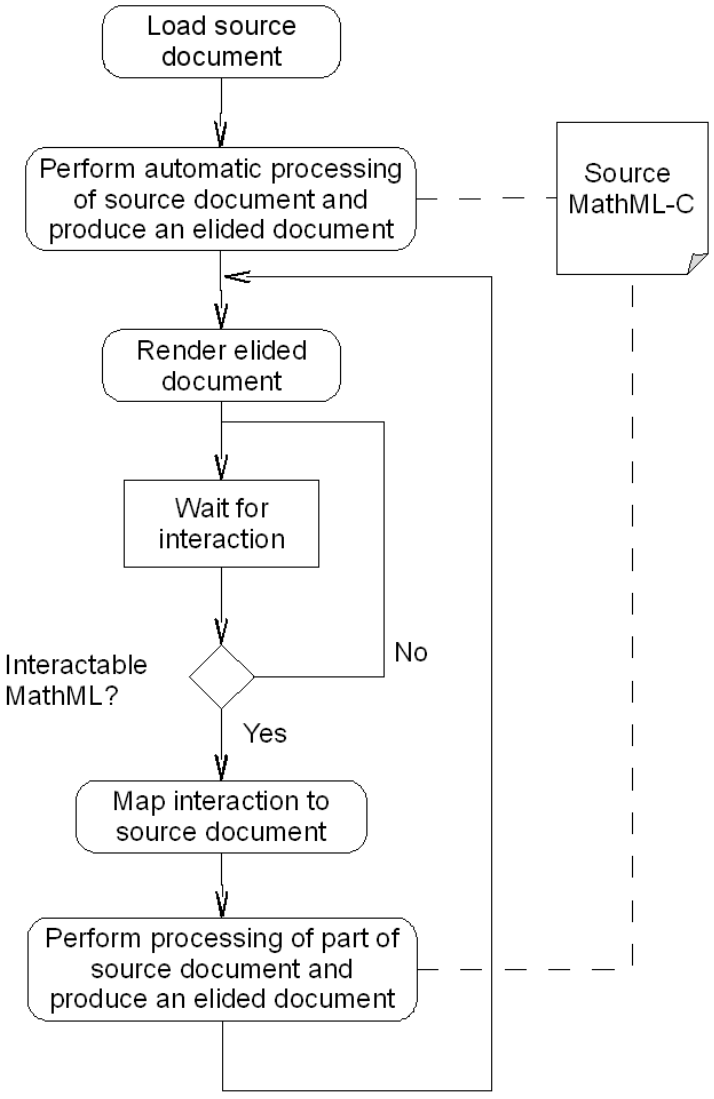

Figure 4.1: Program flow design

4.2.1 Mapping MathML-P to MathML-C

Any information to aid mapping from MathML-P to its corresponding MathML-C will need to be stored in the XML attributes of the particular MathML-P element. In general, this will provide quick access to the information and also be unobtrusive to the meaning of the MathML. There are three ways in which this mapping could be achieved; using the MathML `id` and `xref` attributes with parallel markup, using the combination of XLink and XPointer (see §4.2.1 below), or using a custom XML Schema[29] with our own notation. Another point to note is that we also need to map the elided MathML-C to the original source MathML-C document, thus, a solution that could handle both mappings is desired.

Using 'id' and 'xref' attributes

Example 4.2.1. *Example use of* `id` *and* `xref` *attributes.*

```
<apply>
  <plus/>
  <cn type='integer'>1</cn>
  <ci>a</ci>
</apply>
```

becomes

```
<semantics>
  <mrow xref="C1">
  <mn xref="C3">1</mn> <mo xref="C2">+</mo> <mi xref="C4">a</mi>
  </mrow>
  <annotation-xml encoding="MathML-Content">
    <apply id="C1">
      <plus id="C2" />
      <cn type='integer' id="C3" >1</cn>
      <ci id="C4">a</ci>
    </apply>
  </annotation-xml>
</semantics>
```

This solution, similar to that described in §5.3.3 of the MathML-C specification[31], involves wrapping the MathML-P and MathML-C inside a `semantics` element so that the content and presentation markup is parallel. The MathML-C elements are then marked with some unique `id`s which are referenced by the corresponding MathML-P elements using the `xref` attribute. To retrieve the `id`ed MathML-C element from the MathML-P, we would use the value in the `xref` attribute and search for the corresponding `id` in the content markup. The motivation behind this technique is so that when content markup is transformed for

display, it does not lose its meaning. It is also a well thought out technique as the `xref` and `id` attributes are in the MathML specification. Example 4.2.1 demonstrates this technique. Rodionov and Watt [18] discuss how this technique can be implemented using a tool chain of transforms using XSLT on the MathML-C.

Although this technique provides a solution for mapping MathML-P to MathML-C, it requires a lookup of the `id` for each element. This may not be too much of an overhead for a document that can be loaded into memory, but we also then need to map the elided MathML-C to the source document. In fact, DOM implementations often store the `id` in a hash table for fast lookup. Since the source MathML-C document could be very large, this approach would not be feasible as there would be too many `id` to `xref` mappings to store.

XLink[28] and XPointer[30]

XLink and XPointer are W3C specifications for inserting linking information between resources into XML documents.

Example 4.2.2. *Example use of XLink and XPointer.*

Source MathML-C

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <apply>
    <plus/>
    <cn type='integer'>1</cn>
    <ci>a</ci>
  </apply>
</math>
```

MathML-P

```
<math xmlns="http://www.w3.org/1998/Math/MathML"
    xmlns:xlink="http://www.w3.org/1999/xlink">
  <mrow xlink:href="source.xml#element(/1/1)">
    <mn xlink:href="source.xml#element(/1/1/2)">1</mn>
    <mo xlink:href="source.xml#element(/1/1/1)">+</mo>
    <mi xlink:href="source.xml#element(/1/1/3)">a</mi>
  </mrow>
</math>
```

By using XLink and XPointer, the MathML-P elements have the additional XLink attribute `href`. This XLink attribute uses the XPointer `element()` scheme to store addressing information of the corresponding MathML-C as demonstrated in example 4.2.2. In this

example, the main information we are interested in is the 'path' within `element`. If we follow the 'path' from the MathML-P element representing the identifier `a`, that is `/1/1/3`, on the source MathML-C document, this points us to the 3rd child of the 1st child of the 1st element of the document. Sure enough, this is the `plus` MathML-C element.

This solution does not have a lookup overhead like the previous parallel markup solution but introduces a new overhead of navigating down the XML tree to find a node. This is a much more feasible solution especially for a large document. For example, whereas using `id` and `xref` requires searching for an `id`, which could be on *any* element, using XLink and XPointer helps define a structure so that when the 'paths' do not match we can skip over the element and its children. Issues with using this method would arise as the depth of an expression got *extremely* deep as the XLink expression string increased to unmanageable sizes. Since mathematical expressions are more often wide than deep, we do not expect this situation to occur. Also, it is worth noting that to achieve this case, a user will have to be trying to view an expression of this depth. It is expected that rendering issues will arise long before this case.

4.2.2 Storing interaction information

By storing some information that allows the application to decide what user interaction is available, the overhead of looking up what can be processed, for each MathML-P element the mouse is moved over, is reduced to only those with this information. This would also include information such as whether the element had been elided and by how much, or whether it could be elided. For example, we may want to change the mouse cursor when we are over a matrix or substituted element.

This extra information could be stored within a lookup table to be retrieved as necessary. However, we may want to perform some rendering *based* on this extra information, such as highlighting a particular element if it is a substitution rather than part of the original document. In order to pass this extra rendering information onto the MathML being rendered, we will have to perform some form of transformation before rendering. Thus, to make the extra rendering information more accessible at transformation time, it would be better stored in an XML attribute of the elided document. When transforming to MathML-P, we could then simply copy this attribute.

The MathML `other` attribute could be used for this information, as it designed for any additional information required by an application. However, this attribute is deprecated in MathML version 2 (see §2.4.5 of the MathML specification[31]), so its use should be avoided. We can get around this limitation by defining our own custom XML Schema[29] that allows a custom attribute to store this information by specifying a namespace, as touched on in §7.2.3 of the MathML specification[31].

4.2.3 Summary

In order to map MathML-P to the corresponding MathML-C (and then map this elided MathML-C to the source MathML-C) and store some extra interaction information, we will use a custom XML Schema. This XML Schema will provide two attributes; one to store some basic interaction information, and the other to store an XLink expression as described in §4.2.1. The reason for using our custom XML Schema to store this information rather than using XLink is to avoid confusion, as we are not supporting the entire XLink syntax. (After the conclusions from §4.3 that we would be implementing a custom StAX-based document processor, the decision not to implement the full XLink syntax is to minimise unnecessary work.) These extra attributes will provide the method of mapping user interaction back to the source document.

Custom Schema

The two attributes that we shall introduce with our Schema are `link` and `elision`. Each of these will be global attributes, so they can appear on any XML element and, importantly, MathML elements in our case. For simplicity, they are both declared with the type `xs:string` as we do not require any special structure. Figure 4.2 defines such an XML Schema.

```
<?xml version="1.0" encoding="ascii"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <xs:attribute name="link" type="xs:string"/>
    <xs:attribute name="elision" type="xs:string"/>
</xs:schema>
```

Figure 4.2: Custom XML Schema

We will use the `link` attribute to store an XLink and XPointer expression, as detailed in §4.2.1, and the `elision` attribute to store extra interaction information. Example 4.2.3 demonstrates using a custom Schema in a MathML document.

Example 4.2.3. *Example use of custom Schema.*

```
<math xmlns="http://www.w3.org/1998/Math/MathML"
    xmlns:app="customSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="customSchema customSchema.xsd">
  <mrow app:link="source.xml#element(/1/1)" app:elision="none">
    <mn>1</mn>
    <mo>+</mo>
```

```
    <mi app:link="source.xml#element(/1/1/3)" app:elision="sub">a</mi>
  </mrow>
</math>
```

4.3 Traversing the large document

The way in which the source MathML-C document is parsed and processed needs to be considered. As discussed in §2.2.3, there are several techniques available. In this section we will perform some tests to examine the effectiveness of some document processing techniques, and then discuss how a chosen technique can be applied to the application.

The document processing techniques to be considered are extended VTD-XML, XSLT transforms using a StAX source, and StAX processing. Other techniques mentioned in the literature survey are not being tested for various reasons. XOM is not being considered since it is unclear from the documentation how to set up the parser without loading an entire document into memory. As already discussed, using DOM requires an entire document in memory. Furthermore, since using SAX to process the document is a similar streaming method to StAX (a 'push' rather than a 'pull' parser), it shall not be considered.

Tests are performed on a machine with an Intel Q6600 processor with 2GB RAM.

4.3.1 Memory Usage

Test

Since one the of the main objectives of the application is the ability to process large documents, it is important to test how the document processing techniques cope with a document that is too large for memory. The large MathML-C document used for this test will be a 10000×10000 matrix, almost 1GB on disk (approximately 950MB). See E.1.1 for the code used to generate this matrix.

Result

The code for these tests can be seen in E.2. Extended VTD-XML failed with an 'out of memory' error before getting to any processing of the document. This suggests that a representation of the entire document was being loaded into memory. Thus, this technique shall not be considered any further.

Similarly, using an XSLT processor with a StAX source also failed with an 'out of memory' error.

These results signify that an alternate processing technique for the MathML-C document is required. This processing technique will be based on a StAX parser, which parses the

document as a stream of XML events. One such existing technology is Joost[5], an implementation of STX[7].

4.3.2 Processing Speed

Test

After the results of the memory test, we now proceed with a speed test of STX. STX is an XSLT-like processing language that doesn't require creation of a document tree in memory. STX stylesheets would then be used to perform all document processing. By using some simple STX stylesheets (see E.2.3), we can observe how long it takes Joost to process the large matrix. This test should confirm that processing large MathML-C documents in this way is not an ideal solution as we have to read the entire document each time.

Result

With a simple STX stylesheet, Joost takes approximately 6.5mins to process the large matrix. With a blank STX stylesheet, Joost takes 6.2mins. While this length of time for initially processing the document may be expected, this length of time for each user interaction with such a large document is unacceptable for a responsive user interface. e.g. expanding an ellipsis should not take this long. Thus, a custom StAX-based document processor will be required. By using a StAX parser as the underlying method of traversing the document, we expect memory usage to be minimal.

4.3.3 Design of StAX document processor

StAX parser

As decided in §4.3, the document processor will be built on a StAX parser. The parser will be used to provide a stream of XML events (such as the start of an element, end of an element, and character data) for the processor to react to. The most important XML events will be the start and end elements. When start elements are read in by the parser, if they are elidable, they are processed with appropriate elision data, otherwise the XML event is simply copied to the elided document. In this way, the elided document is built as the processor uses the parser to traverse the source document. Figure A.1 in appendix A.1 shows how the processor uses the StAX parser.

Initial pass

An initial pass of the large MathML-C document will be required, performing an initial summarisation of the mathematical expression. When the elisions during this pass occur, the position of the document parser for each elision should be recorded. Since the processor

will be built on a StAX parser, stream positions of the parser at these points are an ideal way to record this information.

Restarting parser

By storing a mapping from document element, represented using XPointer, to the stream position, the document processor can then be restarted at that particular node. In this way, jumping to the specified stream position in the document, when required to do further processing (such as with user interaction), should be quicker than having to parse the entire document searching for a matching element.

A document processor designed in this way will be expected to initially perform a complete document parse which may be slow. However, further processing of the document based on recorded positions will avoid repeating this overhead.

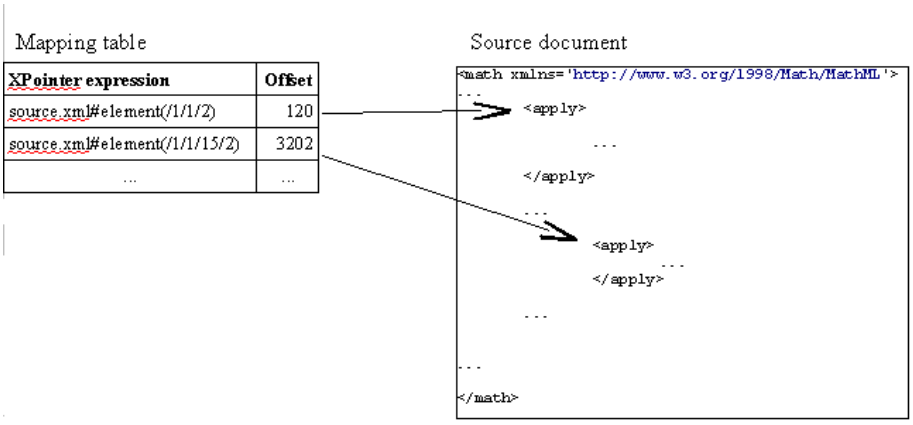

Figure 4.3: Mapping from XPointer expression to the source document

4.4 Overview

Initially, when a file is loaded, the document processor should perform elisions to summarise the expression. When an elision is performed, the processor then stores the file stream offset for the particular MathML-C element in a mapping table using its XPointer expression, see §4.2.1. The XPointer expression is then output in the `link` attribute of our custom Schema (see §4.2.3) on the elided document. Before we render, we transform this elided document using a modified XSL MathML-C to MathML-P stylesheet which retains the custom Schema attributes.

Following user interaction, we retrieve the MathML-P element that the mouse is over. If this element has an `elision` attribute in our custom Schema then we know we can further

process this element. Using the value from the `link` attribute, we then tell the file processor to elide the corresponding element in the source document along with some elision data. The processor then creates a new StAX parser initialised at the stream position retrieved from the mapping table based on the XPointer expression given. Using the elision data, the processor processes this element and outputs to the relevant part of the elided document. In this way we avoid parsing through the entire document each time.

4.5 Elision algorithms

It is important to note that since we are using a StAX parser as the underlying way of traversing the document, both the width and depth must be handled in the same algorithm (as seen in figure 4.4).

The main elision algorithm for `apply` elements is listed in figure 4.4. This algorithm elides by width of an expression using the `numLeft` and `numRight` as the number of terms left and right, respectively, of an ellipsis. We use the StAX parser, `reader`, to progress over the source MathML-C document and only process the child elements we want. This algorithm also decrements the `depth` when processing child elements to control their depth. Towards the start of the algorithm there is a test with `depth` so that a substitution can be performed if necessary.

4.6 User Interface Design

The user interface should be kept simple and intuitive, as detailed in 3.2. The main component of the user interface will be where the MathML is rendered and interacted with. The only other components that will be required are menus for loading the document and setting some elision thresholds. Also, a context-menu will be used on right-clicking part of the expression to give the choice of accessible elision options. The reason for keeping the design this simple is to retain a ‘clean’ user interface without cluttering it up with unnecessary components. Figure 4.5 shows our user interface design.

```
void processApplyElement(XMLStreamReader reader, XMLStreamWriter writer
  StAXState state, int numLeft, int numRight, int depth)
{
  /* store state position */

  // test whether the depth is too low to process
  if(depth <= 0) {
    /* perform substitution */
  }

  /* write out 'apply' to XMLStreamWriter */

  // read ahead and count children
  childCount = countChildren(reader, state);
  // extra +1 is for 'apply' element's operator 1st child
  isEliding = (childCount > (numLeft + numRight + 2));

  if (isEliding) {
    /* write elision attribute */
    int child = 0; // child counter

    while(reader.hasNext()) {
      reader.next();
      if( /* reader is on child */ ) {
        child++;
        if( (child <= (numLeft + 1))
          || (child > (childCount - numRight)) ) {
          processElement(reader, writer, state, depth - 1);
        } else {
          /* write ellipsis once only */
        }
      }
    }

  } else {
    /* process all children */
  }
}
```

Figure 4.4: Main elision algorithm on 'apply' element

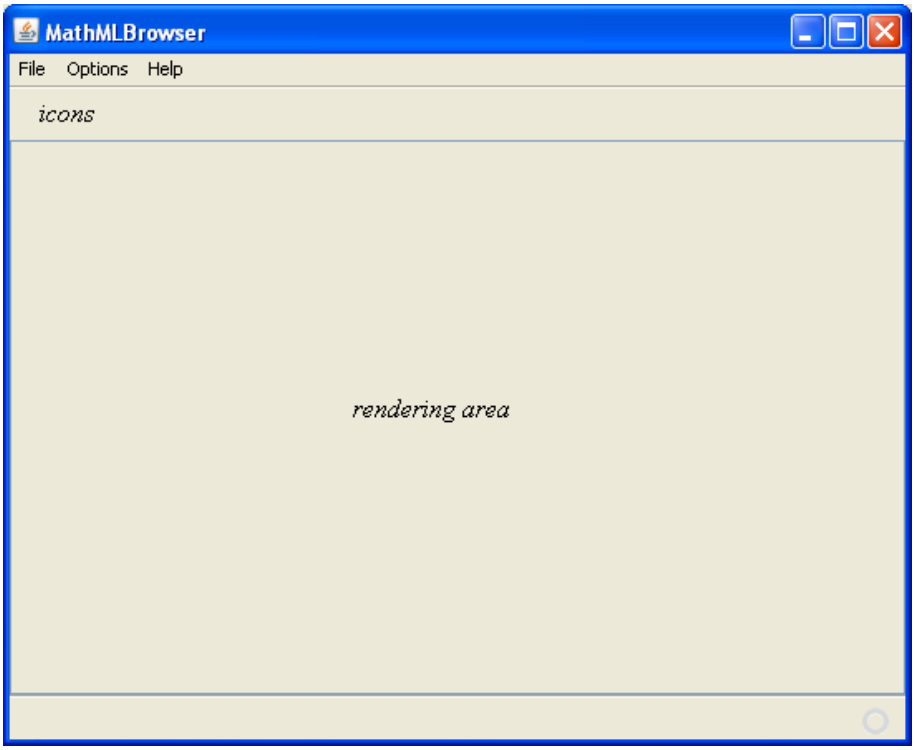

Figure 4.5: User interface design

Chapter 5

Implementation and Testing

This chapter describes key parts of the implementation and testing of the application.

5.1 Implementation

5.1.1 Language

As decided in the literature survey (see §2.3.2), the Java programming language was used to implement the application. Using a low-level programming language, such as C or C++, may have increased the performance of the application. However, there would be a trade-off of requiring either multiple implementations for multiple platforms or cross-platform developed libraries for the user interface, MathML rendering, and XML parsing. Also, as summarised in a Java vs C++ comparison[14], it is worth noting that the performance of Java on low-level and numeric benchmarks is not an issue.

5.1.2 Document processor

The majority of development effort went into creating the document processor. Although the basic design was discussed in chapter 4, implementing the processor proved more difficult than expected. The main issue stemmed from attempting to avoid implementing our own StAX document parser (as we wanted to avoid 're-inventing the wheel').

The StAX parser

Initially in the application implementation, the document processor was developed around the default Java StAX parser. This default `XMLStreamReader` implementation uses the StAX parser from the Apache Xerces Project[23]. The only way of retrieving the position of the parser on the file stream is using the location information returned by the function `XMLStreamReader.getLocation()`. This information is important for storing the location of key MathML-C elements as described in §4.3.3. However, the default `XMLStreamReader` implementation does not always return accurate file stream position information, especially after jumping to a file position. This meant that after a position was obtained, we tried to use backtracking in the file to retrieve the actual MathML-C element position.

The inconsistencies in the retrieved file stream positions and inability to find an appropriate backtracking technique (even after analysing the parser's source code) meant using the default parser was abandoned. Eventually, a suitable StAX parser that stores accurate stream position information was found in Woodstox[8]. Woodstox's StAX parser can be created with the `WstxInputFactory.P_PRESERVE_LOCATION` property to ensure stream position is stored accurately.

MathML elements

As highlighted in §2.2.1, the most common n-ary MathML-C element is the `apply` element. Due to time constraints, only this n-ary MathML-C element was considered for elision. This is an acceptable compromise in the application implementation since it demonstrates the way in which the other n-ary MathML-C elements could be elided.

For the same reason, only the bitmap representation of a matrix, from the discussion of representing matrices in §2.1.4, was implemented. Future work into other representations of matrices would be useful, particularly ways to display the actual content of a larger-than-memory matrix.

The elided document

For debugging purposes, the elided MathML-C document was initially written out as a file in the working directory of the application. However, when attempting to use the Woodstox library's `XMLStreamWriter` to a resulting DOM document so that the elided document could be stored in memory, we ran into issues. For this reason the elided document is still stored in the working directory of the application. Although this does not affect the functionality of the application, it is not a desired result. With further investigation into the Woodstox or another parser's libraries, we would expect to resolve this issue.

Recursive substitutions

To avoid the large overhead of comparing a MathML-C element and all its children to other MathML elements, we decided to not implement recursive substitutions. That is, replacing all occurrences of a MathML-C element when it is substituted. Not only would an element have to be compared against all the children of a substitution, but also we would have to test *all* the stored substitutions. This implementation decision detracts from the intuitiveness of the application, as when summarising mathematical expressions on paper people identify and substitute for common terms. A future development would certainly be to implement such a feature to make using the application more appealing to a user.

5.1.3 Other Elision Algorithms

Matrices are processed in a simpler way than figure 4.4 with their dimensions being calculated, then a simple check as to whether or not they should be elided.

Figure 5.1 outlines the algorithm used after clicking on the MathML. The `xpointer` string is retrieved from the attributes of the rendered MathML and then used in this algorithm. The key points to note are the copying of the elided document until the element to process, and then performing a 'jump' to the relevant part of the file stream on the source document. Because we move the underlying file stream, there is negligible time to move to the

```
void setElision(String xpointer, Object elisionData, XMLStreamWriter writer)
{
  /* copy elided document up to element with specified xpointer */

  // get corresponding source doc position from xpointer expression
  StAXState state = elem2posTable.get(xpointer);

  // jump to part of stream we want
  FileInputStream inputStream = new FileInputStream(sourceFilePath);
  inputStream.getChannel.position(state.offset);
  instreamOffset = state.offset; // any file stream positions are offset by this amount

  XMLStreamReader inputReader = XMLInputFactory.createXMLStreamReader(inputStream);

  // move inputReader state to the next start element
  while( /* read is not start element */ )
    inputReader.next();

  /* preprocess elisionData */

  processElement(inputReader, writer, new StAXState(state), depth);

  /* copy rest of elided document */
}
```

Figure 5.1: Elision algorithm after interaction

particular part of the document. If we were to simply use a `XMLStreamReader` from the beginning of the file and skip to the relevant element, the cost of this would be too great, especially on a large document, leading to slow interactions.

5.1.4 Transforming MathML

So that the MathML-C can be rendered, it needs to be transformed to MathML-P. However, this cannot be done by simply using an existing MathML-C to MathML-P XSL stylesheet because of our custom Schema (as designed in §4.2.3). To make sure that our Schema is copied to the MathML-P, we modified JEuclid's `mathmlc2p.xsl` using a stylesheet (see E.3). While this method of modifying the stylesheet is not perfect, it ensures most of the MathML-P elements correctly copy the custom Schema attributes. The template rules of the resulting stylesheet with the `match` attribute as `m:apply[*[1][self::m:compose]]` and `m:apply[*[1][self::m:apply]]` need to be subsequently corrected.

5.1.5 Key Assumption

Throughout the implementation we have assumed that the elided document, produced by the document processor, is sufficiently small to handle. By assuming that this document can be loaded into memory, we do not take away from being able to process large documents. In fact, this assumption is a requirement, since XSL transforming the elided MathML-C to MathML-P requires loading the document into memory. Also, JEuclid requires this condition since it creates a DOM tree of the MathML to render.

Furthermore, it is worth noting that even the Maple factorisation of $(x^{1155} - 1)$, a product of 16 summations with the 16th containing over 300 terms, exported as MathML, is only 150KB. This amount can easily be loaded into memory on current desktop machines. Thus, as we do not expect a user to be trying to make sense of so many terms at once, our assumption is valid.

5.2 Testing

In order to evaluate the main functionality of the application, testing key components is required. The main parts of the application that need testing are the StAX document processor and user interaction.

To test the StAX document processor, firstly we need to check that the file position offsets, which are stored, are correct. Additionally, any new file position offsets stored after expanding part of the expression should also be analysed since these positions will have been stored after a 'jump' to part of the document. This is the motivation behind Test 1 (see C.1.1). For this test, we also chose a large enough file to ensure the underlying input stream would have to perform multiple reads. The reason for this condition is since it was an area

that seemed to cause inconsistent behaviour in the default Java StAX parser (described in §5.1.2). Since the operation of the application heavily relies on these file position offsets being stored correctly, the positive results of this test, see C.2.1, are very important.

Another interesting part of the system to test is the time taken for the StAX processor to process a large document. Test 2 (see C.1.2) processes the large matrix used in §4.3. From the results of Test 2 (see C.2.2), we can see that the StAX processor took 1 minute and 8 seconds. This is much faster than the ∼6 minutes taken using STX in §4.3.2.

Building on Test 2, it is important to test how long further interaction with a document that has a large part takes. Test 3 (see C.1.3) is designed for this purpose. By using an expression consisting of the large matrix from Test 2 multiplied by a long summation, we can analyse interaction time. The short time seen in the results (C.2.3), 31ms, demonstrates a sufficient interaction time.

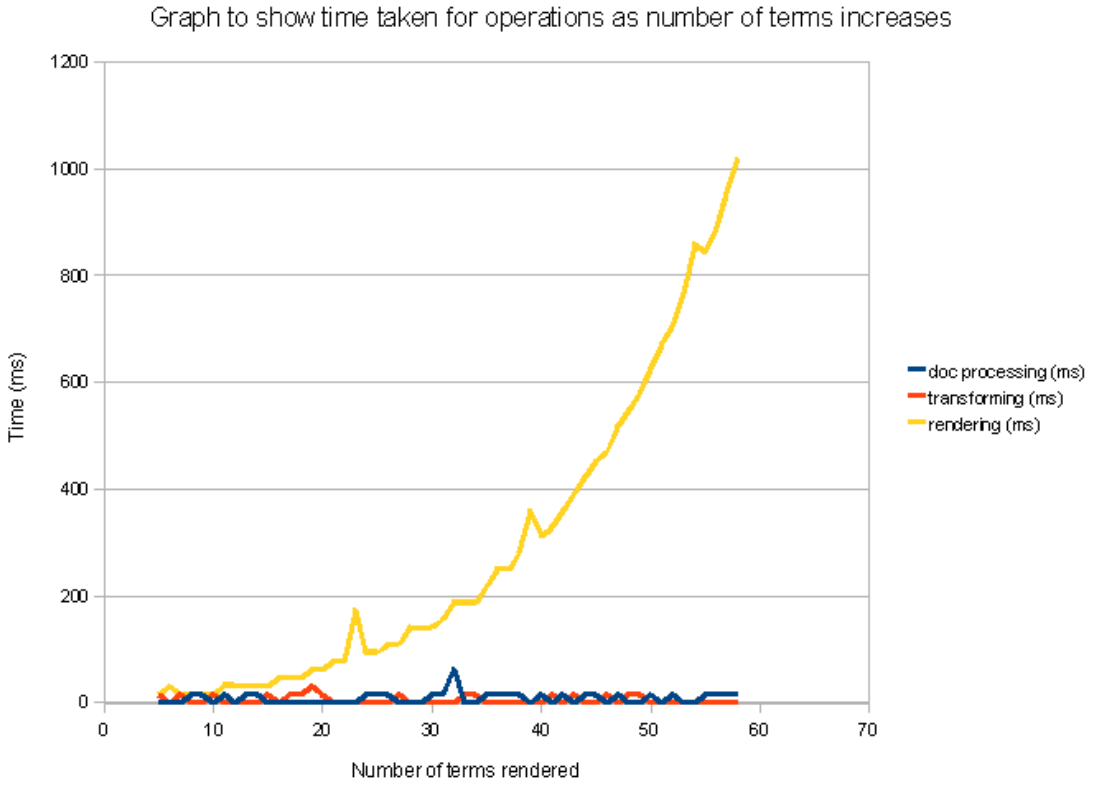

Figure 5.2: Interaction process times chart

In order to analyse user interaction times, we should analyse the times taken for the main process of operations on an interaction on a simple expression, increasing the number of terms in view. These are document processing, transforming the MathML-C to MathML-P, and rendering. This is the motivation for Test 4. The results of this test are summarised in

figure 5.2. It should be noted that we have excluded the first set of processing times from the chart since these are longer due to initialisation of parts of the application. This chart shows an exponential increase in rendering time as the number of terms in the summation is increased. Thus, the performance of the renderer is a significant factor in user interaction of this application. With hindsight, more investigation into the performance of MathML renderers should have been carried out. Also, when initially running this test, we saw that the MathML-C to MathML-P transform time was between 400ms and 600ms, dominating the interaction process. By compiling the XSL stylesheet, rather than interpreting it each time, this time was significantly reduced.

Part of the user interaction that we did not test was the rendering of MathML correctly. This relies entirely on JEuclid, and is thus not considered.

Chapter 6

Results

This section details the main results of the project as well as some key points from user evaluation of the application.

6.1 Summarising an Expression

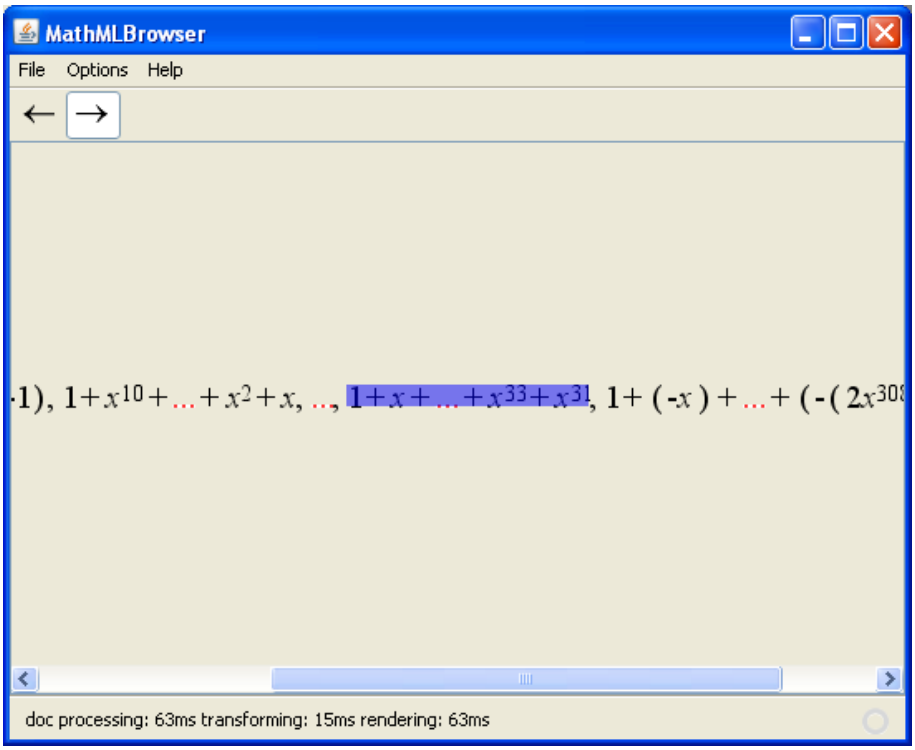

Figure 6.1: Wide elided expression

The application is able to summarise a large mathematical expression in the format of a MathML-C document. This involved developing elision algorithms based on width and depth of content MathML expressions on the `apply` element. Prespecified width and depth thresholds (specified on loading a document) are then used to elide a large expression. Figure 6.1 shows such a wide expression that is elided by width. This expression can then be interacted via left clicking for quick elision either to the left or right of an ellipsis (see the arrows in the user interface), or right clicking for further options. Figure 6.2 shows an automated substitution with a context-menu providing several options.

Although only the `apply` element is considered for this type of elision, these ideas can be extended to the other n-ary MathML-C elements. Without supporting such other n-ary MathML-C elements (summarised in §2.2.1), the application is not a complete browsing solution for MathML-C.

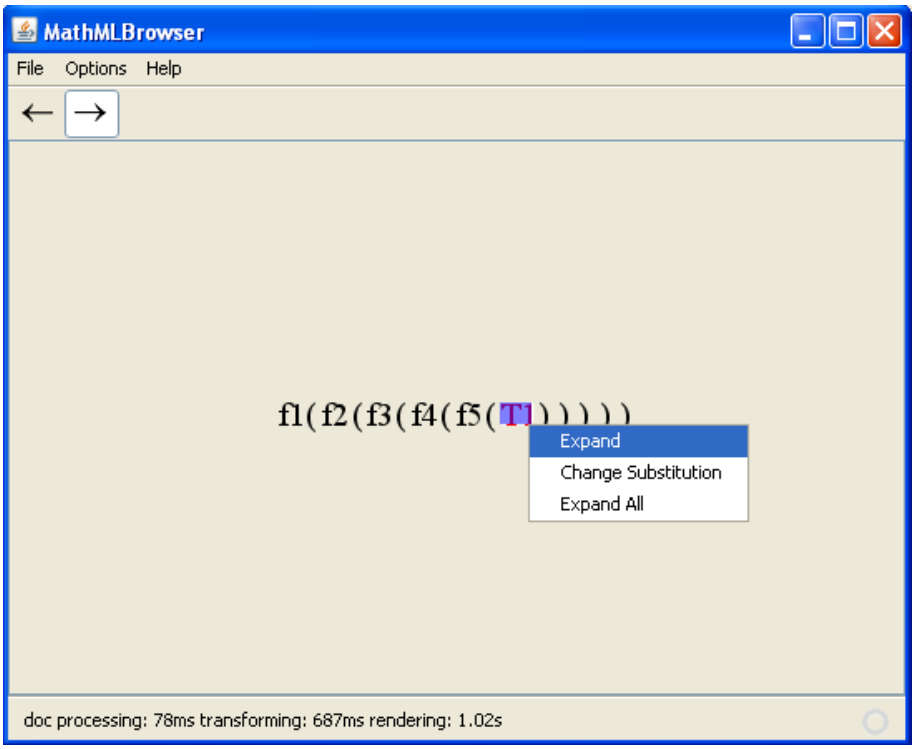

Figure 6.2: Substituted expression

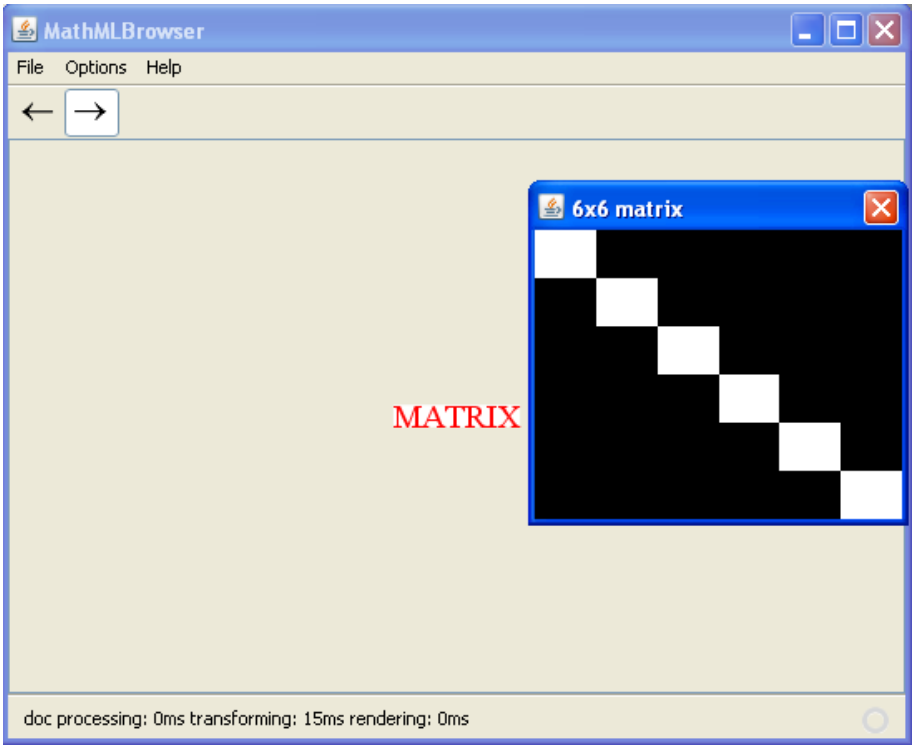

Figure 6.3: Small matrix

Figure 6.4: Large matrix

Also, an alternate visualisation of a matrix was implemented based upon a method from Maple; see Figure 2.1. This view allows a user to observe the structure of a large matrix that is too large to comprehensibly look at or render. Figure 6.3 shows how this view appears for a small matrix and figure 6.4 demonstrates the view for a large matrix. Although the 100×100 matrix in figure 6.4 allows us to view such a large matrix, it doesn't scale well in terms of size. i.e. the squares representing data in the matrix are much smaller than in figure 6.3. In future versions of the application, we would expect to 'polish off' this view and use some sort of zooming function to keep the size of squares representing entries in a matrix of a consistent size. The maximum limit imposed for a matrix that could be loaded in this way is 1000×1000. This restriction was decided based on current screen resolution statistics [33] and memory issues with higher orders of magnitude. (The resolution is important since the summary of the matrix requires at least one pixel per entry.)

Since after analysing some MathML-C a user may want to save the result, the application has a menu option (File → Export) that allows the rendered MathML-P to be saved. For example, this may be useful for importing into a report. Also, as a side effect of the processing technique on MathML-C documents, the application is able to load MathML-P for rendering, but does not support any interaction.

6.2 Interactive processing of the source content MathML document

By using a custom StAX-based processor and storing file stream positions of elisions made, the application allows relatively quick access to the source document in order to perform any further expansion/elision of particular content MathML elements. This is achieved with the initial overhead of an initial pass of the entire source document. Figure 6.5 displays a processing time summary of 2mins for initially loading a large document (this file is the MathML-C generated by E.1.3, and is approximately 1GB). This provides a way of processing the document efficiently and avoiding loading an entire document into memory. There are two main costs of this method of processing the document; time spent performing further processing and XML namespaces.

Since the document is not loaded entirely into memory, further processing after user interaction will take the time required to process the relevent part of the file. This information is on disk and therefore will not be as quick as an in-memory solution such as DOM.

The other cost is the use of an XML namespace prefix in the MathML-C document rather than having no prefix. e.g. `m:apply` rather than `apply`. This is due to jumping to stream positions within the file and then using a StAX parser when further processing the document. This issue stems from utilising an existing StAX parser; Woodstox. Since we wanted to avoid implementing our own XML parser, this is an acceptable cost.

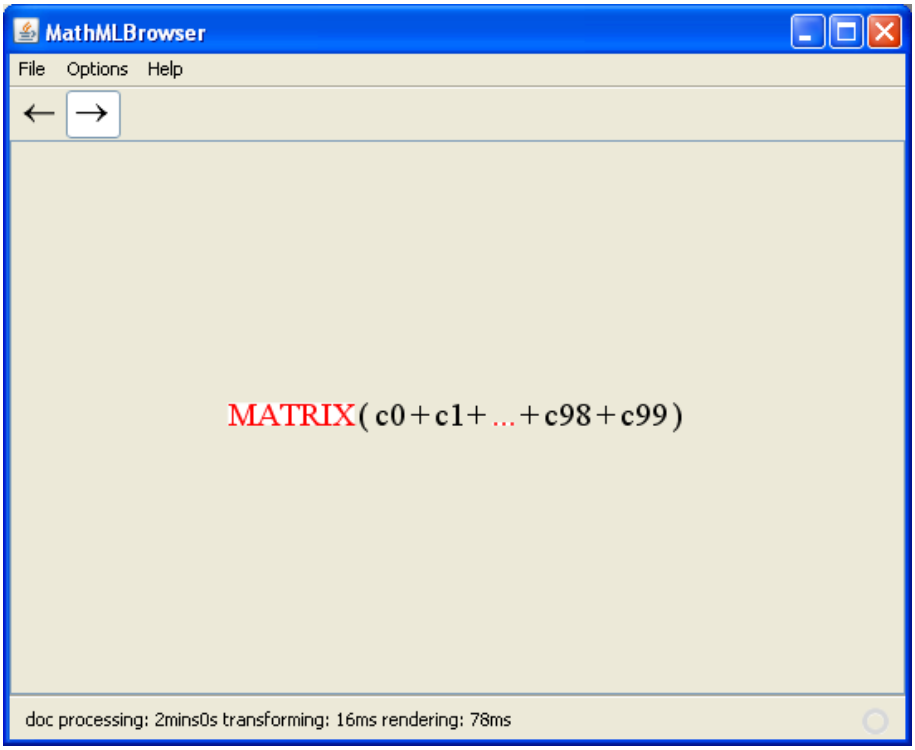

Figure 6.5: Large matrix with other expression

6.3 Interacting with the rendered expression

By using the XPointer element scheme, rendered presentation MathML can be mapped back to the source document in order to process user interactions. As seen in figure 6.1, the mouse position can be mapped to the rendered MathML allowing us to highlight the interactable expression.

Chapter 7

Conclusions

7.1 Review

The purpose of this project was to create an application that allows a user to browse a large mathematical expression represented as a MathML-C document. This is achieved through the three main results of the project (summarised in chapter 6) by giving the user the ability to browse via width and depth of the expression and perform substitutions. The main focus of the project was processing a large document.

This result was achieved by developing a novel StAX-based processor to enable efficient processing of a large MathML-C document by storing important document positions. See §6.2 for further details. One factor that made this part of the project take longer than expected was since, originally, we were expecting to navigate the source document using XPath expressions and XSLT. Thus, there was a lot of effort spent learning advanced XSLT and producing stylesheets that could have been avoided. In retrospect, testing the different ways of processing a large MathML-C document at an earlier stage would be a key decision for the project.

After deciding upon a document processing solution, another difficulty that arose was in retrieving precise stream positions from the underlying StAX parser. As detailed in §5.1.2, a suitable parsing solution was found very late on in development. Had this solution been used from the start, this project could have achieved more. Furthermore, the XML namespace cost detailed in §6.2 could be resolved with a custom StAX parser that allowed namespaces to be set before parsing a section of the file.

Another key design decision was to use JEuclid as the renderer for the application. This choice stems from deciding to use Java as the programming language of the application to allow cross-platform compatibility. Since JEuclid needs to read in the entire MathML document each time re-rendering is required, and recalculates its rendering information, it can be slow for large expressions. Figure 5.2 demonstrates an exponential increase in rendering time as more terms of an expression are rendered. If we were able to render parts of an expression, maybe render time would not be so much of an issue. Further analysis of other rendering solutions, such as web browser integration and GtkMathView discussed in §2.2.2, would ideally be carried out if this project were to be repeated. We also note that the code we developed for mapping mouse position to rendering information, by editing the JEuclid source, is now included in JEuclid release 3.1.5.

Since the initial summarisation of a mathematical expression always requires a full parse of the source MathML-C document, large documents will take a non-trivial amount of time to load. Further investigation into the optimisation of the initial processing would have ideally been carried out with more time. Similarly further optimisations of the main elision algorithm should be looked into by analysing what extra data could be stored to speed it up. This may be information such as the position of the end element corresponding to the start element, so that skipping over elements, when further processing after user interaction, is quicker.

As detailed in the implementation (see §5.1.2), only the `apply` MathML-C element was

considered for width-based elisions. i.e. using an ellipsis. For this reason, the application is not a complete MathML-C browsing solution. However, even though the application only elides `apply` elements, it is a firm base for future work.

7.2 Future Work

The addition of support for the other n-ary elements detailed in §2.2.1 would be required to make this application a complete browsing solution for MathML-C. The elision process for these elements is expected to be very similar to `apply` elements.

A particularly interesting area of work that we did not proceed with development is different representations of matrices. (This is discussed in §2.1.4.) While the solution that was implemented allows a user to view the essential structure of a matrix, using a bitmap view, they then may want to analyse some of the actual terms of the matrix. A window that rendered the actual matrix content from a particular region of the matrix would be useful for this purpose. From this region, then allowing expansion in the different axes would provide a useful method for browsing a matrix's actual content. However, design of such a method would have to be carefully carried out so that large matrices, possibly larger than memory, could also be browsed efficiently in this way.

Another useful piece of functionality for matrices would be to allow a user to perform a query on their structure. By producing some summary information, the user would be able to better analyse the matrix. Such information could be whether the matrix is diagonal, upper triangular, the identity, etc... as mentioned in §2.1.4.

For linear n-ary expressions, inserting more than one ellipsis may be a possible extension. This would allow analysis of multiple parts of a wide expression. For example, for an expression we may have the following summary; $1 + \cdots + 10$. Using a second ellipsis, the structure of this expression may be analysed further, say, $1 + \cdots + 5 + \cdots + 10$.

7.3 Final remark

The main results of this project culminate in the ability for a user to interact with a large mathematical expression given as a content MathML document. Although the application doesn't implement all the features a user could want in order to browse a large content MathML document, it provides a firm base of key features that future work could build on. Particularly, the method of processing large documents.

Overall, working on this project has been enjoyable. Other ways of interactivity with large matrices would have been particularly interesting if sufficient time were available. However, the navigation of such larger-than-memory two-dimensional data structures would require careful consideration to allow efficient processing.

Bibliography

[1] GtkMathView. http://helm.cs.unibo.it/software/mml-widget/.

[2] Java API for XML Processing (JAXP). https://jaxp.dev.java.net/.

[3] JEuclid. http://jeuclid.sourceforge.net/.

[4] JEuclid mailing list. jeuclid-users@lists.sourceforge.net.

[5] Joost - The Streaming XML Transformer. http://joost.sourceforge.net/.

[6] Simple API for XML (SAX). http://www.saxproject.org/.

[7] Streaming Transformations for XML. http://stx.sourceforge.net/.

[8] Woodstox - High-performance XML processor. http://woodstox.codehaus.org/.

[9] XOM. http://www.xom.nu/.

[10] Design Science. WebEQ. http://helm.cs.unibo.it/software/mml-widget/.

[11] Laurent Dirat. JOME, a software component for interactive and distributed mathematics. *SIGSAM Bull.*, 34(2):38–42, 2000.

[12] Norbert Kajler and Neil Soiffer. A Survey of User Interfaces for Computer Algebra Systems. *Journal of Symbolic Computation*, 25(2):127–159, 1998.

[13] Michael Kohlhase, Christoph Lange, and Florian Rabe. Presenting mathematical content with flexible elisions. October 2007.

[14] J. P. Lewis and Ulrich Neumann. Performance of Java vs C++. 2004. http://www.idiom.com/~zilla/Computer/javaCbenchmark.html.

[15] Maplesoft. Maple 12. http://www.maplesoft.com/Products/Maple/.

[16] Mozilla. Firefox. http://www.mozilla.com/firefox/.

[17] Ontario Research Center for Computer Algebra. mathml@ORCCA. http://www.orcca.on.ca/MathML/.

[18] Igor Rodionov and Stephen M. Watt. Content-Faithful Stylesheets for MathML. 2000. Available as Technical Report TR-00-24.

[19] Alan Sexton and Volker Sorge. Processing textbook-style matrices. Mathematical Knowledge Management, 4th International Conference, 2005.

[20] C. J. Smith and N. Soiffer. Mathscribe: a user interface for computer algebra systems. In *SYMSAC '86: Proceedings of the fifth ACM symposium on Symbolic and algebraic computation*, pages 7–12, New York, NY, USA, 1986. ACM.

[21] soft4science. MathML Renderer. `http://www.soft4science.com/products/MathML_Renderer/s4s_MathML_Renderer.html`.

[22] Neil Morrell Soiffer. *The Design of a User Interface for Computer Algebra Systems*. PhD thesis, EECS Department, University of California, Berkeley, Apr 1991.

[23] The Apache Software Foundation. The Apache Xerces Project. `http://xerces.apache.org/`.

[24] The OpenMath Society. The OpenMath Standard 2.0. Jul 2004. `http://www.openmath.org/standard/om20/`.

[25] Tigris.org. Subversion. `http://subversion.tigris.org/`.

[26] T. R. Tyhurst. Mathematical Output Presentation in User Interfaces for Computer Algebra Systems. Master's thesis, University of Waterloo, Ontario, 1993. Available as Technical Report CS-93-05.

[27] W3C. Document Object Model (DOM). `http://www.w3.org/DOM/`.

[28] W3C. XML Linking Language (XLink) Version 1.0. `http://www.w3.org/TR/xlink/`.

[29] W3C. XML Schema. `http://www.w3.org/XML/Schema`.

[30] W3C. XPointer element() Scheme. `http://www.w3.org/TR/xptr-element/`.

[31] W3C. Mathematical Markup Language (MathML) Version 2.0 (Second Edition). Oct 2003. `http://www.w3.org/TR/MathML2/`.

[32] W3C. Extensible Markup Language (XML) 1.0 (Fourth Edition). Aug 2006. `http://www.w3.org/TR/2006/REC-xml-20060816/`.

[33] W3Schools. Browser Display Statistics, 2009. `http://www.w3schools.com/browsers/browsers_display.asp`.

[34] C. J. M. Walmsley. An Extensible System for the Display of Nested Array Data Structures. Master's thesis, Queen's University, Ontario, 1998.

[35] Merriam Webster. Merriam Webster Online. `http://www.merriam-webster.com/`.

[36] Wikipedia. Block matrix — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Block_matrix&oldid=251757182, 2008. [Online; accessed 29-November-2008].

[37] Wikipedia. Diagonal matrix — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Diagonal_matrix&oldid=252307693, 2008. [Online; accessed 29-November-2008].

[38] Wikipedia. Identity matrix — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Identity_matrix&oldid=240592360, 2008. [Online; accessed 29-November-2008].

[39] Wikipedia. SAX — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=SAX&oldid=252889985, 2008. [Online; accessed 5-December-2008].

[40] Wikipedia. Skew-symmetric matrix — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Skew-symmetric_matrix&oldid=250860498, 2008. [Online; accessed 29-November-2008].

[41] Wikipedia. StAX — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=StAX&oldid=232549218, 2008. [Online; accessed 7-December-2008].

[42] Wikipedia. Symmetric matrix — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Symmetric_matrix&oldid=254527682, 2008. [Online; accessed 29-November-2008].

[43] Wikipedia. Triangular matrix — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Triangular_matrix&oldid=233051875, 2008. [Online; accessed 29-November-2008].

[44] Wikipedia. VTD-XML — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=VTD-XML&oldid=255103230, 2008. [Online; accessed 5-December-2008].

[45] Wolfram. Mathematica. http://www.wolfram.com/products/mathematica/index.html.

[46] XimpleWare. A Quick Overview on Virtual Token Descriptor. http://vtd-xml.sourceforge.net/VTD.html.

[47] XimpleWare. VTD-XML: The Future of XML Processing. http://vtd-xml.sourceforge.net/.

[48] XimpleWare. Extended VTD-XML released, Oct 2008. http://sourceforge.net/forum/forum.php?forum_id=878812.

Appendix A

Design Diagrams

A.1 Processor with parser

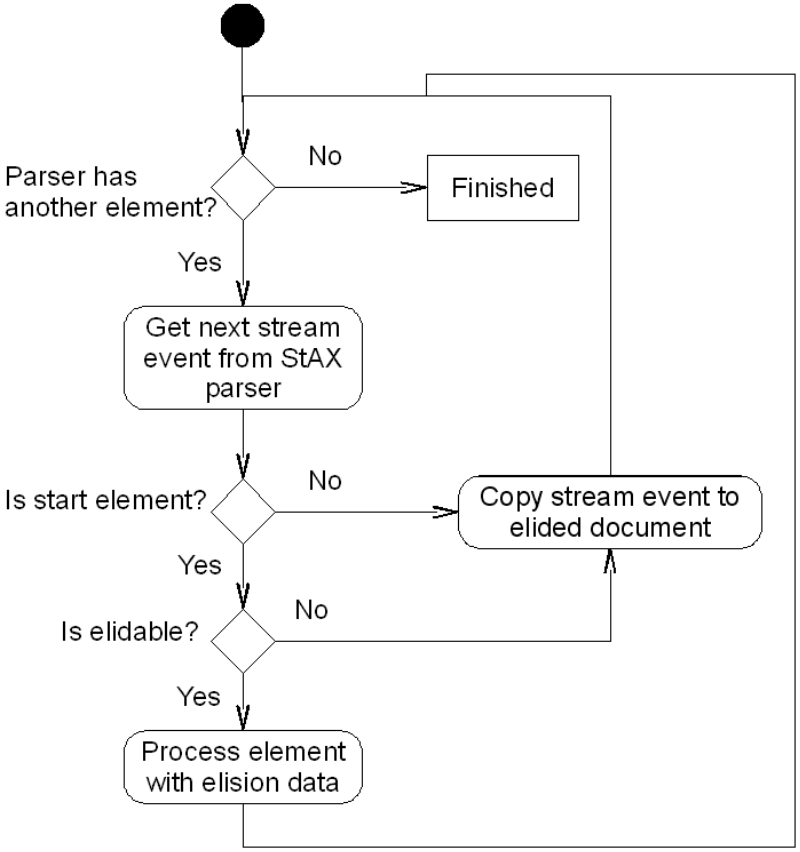

Figure A.1: Processor flow design

Appendix B

Document Processor Interface

```
package mathmlbrowser;

import org.w3c.dom.Node;

/**
 * Interface used for interactive elision of a MathML-C file.
 */
public interface IFileProcessor
{
    /**
     * The types of elision.
     */
    public enum ElisionType
    {
        DEFAULT, DEPTH, WIDTH, WIDTHR, WIDTHL, ALL, SUB
    }

    /**
     * Processes the file to produce an elided document accessible using
     * {@link #getElidedDoc() }
     * @throws mathmlbrowser.FileProcessingException
     */
    public void process() throws FileProcessingException;

    /**
     * Gets a DOM representation of the elided document.
     * @return the elided document
     */
    public Node getElidedDoc();

    /**
     * Set the depth in an expression at which elision occurs.
     * @param depth
     */
    public void setElisionDepthThresh(int depth);
    /**
     * @return depth in an expression at which elision occurs
     */
    public int getElisionDepthThresh();

    /**
     * Set the width of an expression at which elision occurs.
     * @param width
     */
    public void setElisionWidthThresh(int width);
    /**
     * @return width of an expression at which elision occurs
     */
```

```
    public int getElisionWidthThresh();

    /**
     * Set the width and height of a matrix at which elision occurs
     * @param size
     */
    public void setMatrixThresh(int size);

    /**
     * @return width and height of a matrix at which elision occurs
     */
    public int getMatrixThresh();

    /**
     * Expand an elided element
     * @param xpointer  the element to set the elision on
     * @param elisionType  the type of elision
     * @param amount  the amount of elision (based on elisionType)
     */
    public void setElementElision(String xpointer, ElisionType elisionType, int amount)
            throws FileProcessingException;

    /**
     * Set an elision by width.
     * @param xpointer  the element to set the elision on
     * @param amountLeft  the number of terms left of the ellipsis
     * @param amountRight  the number of terms right of the ellipsis
     * @throws mathmlbrowser.FileProcessingException
     */
    public void setWidthElision(String xpointer, int amountLeft, int amountRight)
            throws FileProcessingException;

    /**
     *
     * @param xpointer  the element to set the elision on
     * @param subText  the substitution text to use
     * @throws mathmlbrowser.FileProcessingException
     */
    public void setSubstitution(String xpointer, String subText)
        throws FileProcessingException;

}
```

Appendix C

Testing

C.1 Test Plan

C.1.1 Test 1

We need to test correct positions within a document are stored correctly on the initial pass, and on further processing. The MathML-C file to be used is the result of exporting the expression after running `factor(x^1155 - 1)` in Maple.

Test1.java

```
package mathmlbrowser.tests.test1;

import mathmlbrowser.IFileProcessor.ElisionType;
import mathmlbrowser.StAXFileProcessor;

/** This test is designed to test the stored offsets are correct */
public class Test1
{
    public static void main(String[] args)
    {
        try {
            StAXFileProcessor sfp =
                    new StAXFileProcessor("src/mathmlbrowser/tests/test1/test1.xml");
            sfp.process();
            sfp.printOffsetTable();

            String xpointer = "test1.xml#element(/1/1/17)";
            System.out.println("Expanding " + xpointer);
            sfp.setElementElision(xpointer, ElisionType.WIDTHL, 2);
            sfp.printOffsetTable();
            sfp.close();
        } catch(Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

C.1.2 Test 2

This test should analyse the speed of the StAX processor for a large file. The large matrix generated by E.1.1 will be used for this test.

Test2.java

```
package mathmlbrowser.tests.test2;

import mathmlbrowser.StAXFileProcessor;
import mathmlbrowser.Utils;

public class Test2
{
    public static void main(String[] args)
    {
        try {
            StAXFileProcessor sfp =
                    new StAXFileProcessor("../GenerateLargeMatrix/LargeMatrix.mml");
            long time1 = System.currentTimeMillis();
            sfp.process();
            long time2 = System.currentTimeMillis();
            System.out.println("Time taken to process: "
                    + Utils.getTimeStr(time2 - time1));
            sfp.close();
        } catch(Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

C.1.3 Test 3

This test should analyse the speed of extra processing by the StAX processor after a large part of the document. For this test we use a MathML-C document of the following format: $M * (a + b + c + \cdots)$ where M is the large matrix from Test2, and $a + b + c + \cdots$ is a summation of 100 terms. This file is generated by E.1.3.

Test3.java

```
package mathmlbrowser.tests.test3;

import mathmlbrowser.IFileProcessor.ElisionType;
import mathmlbrowser.StAXFileProcessor;
import mathmlbrowser.Utils;

public class Test3
{
    public static void main(String[] args)
    {
        try {
```

```
            StAXFileProcessor sfp =
                    new StAXFileProcessor("../GenerateLargeMatrix/LargeExpr.mml");
            long time1 = System.currentTimeMillis();
            sfp.process();
            long time2 = System.currentTimeMillis();
            String xpointer = "LargeExpr.mml#element(/1/1/3)";
            sfp.setElementElision(xpointer, ElisionType.WIDTHR, 2);
            long time3 = System.currentTimeMillis();
            System.out.println("Time taken to process: "
                    + Utils.getTimeStr(time2 - time1));
            System.out.println("Time taken to additionally process: "
                    + Utils.getTimeStr(time3 - time2));
            sfp.printOffsetTable();
            sfp.close();
        } catch(Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

C.1.4 Test 4

Test how wide an expression can be before noticing issues with rendering. To test this we ran the main application using a wide summation of 100 terms, generated by E.1.2. The times for operations were noted after each mouse interaction that revealed another term.

C.2 Test results

C.2.1 Test 1

Output:

```
=== Element offset table ===
    offset xpointer
        53 test1.xml#element(/1/1)
        92 test1.xml#element(/1/1/2)
       213 test1.xml#element(/1/1/3)
       295 test1.xml#element(/1/1/3/3)
      1363 test1.xml#element(/1/1/3/11)
     72697 test1.xml#element(/1/1/16)
     78034 test1.xml#element(/1/1/16/33)
     78175 test1.xml#element(/1/1/16/34)
```

```
     78328 test1.xml#element(/1/1/17)
     78415 test1.xml#element(/1/1/17/3)
    149758 test1.xml#element(/1/1/17/343)
    149793 test1.xml#element(/1/1/17/343/2)
    149893 test1.xml#element(/1/1/17/343/2/3)
    150085 test1.xml#element(/1/1/17/344)
    150179 test1.xml#element(/1/1/17/344/3)
Expanding test1.xml#element(/1/1/17)
=== Element offset table ===
    offset xpointer
        53 test1.xml#element(/1/1)
        92 test1.xml#element(/1/1/2)
       213 test1.xml#element(/1/1/3)
       295 test1.xml#element(/1/1/3/3)
      1363 test1.xml#element(/1/1/3/11)
     72697 test1.xml#element(/1/1/16)
     78034 test1.xml#element(/1/1/16/33)
     78175 test1.xml#element(/1/1/16/34)
     78328 test1.xml#element(/1/1/17)
     78415 test1.xml#element(/1/1/17/3)
     78496 test1.xml#element(/1/1/17/4)
     78590 test1.xml#element(/1/1/17/4/3)
     78756 test1.xml#element(/1/1/17/5)
     78791 test1.xml#element(/1/1/17/5/2)
    149758 test1.xml#element(/1/1/17/343)
    149793 test1.xml#element(/1/1/17/343/2)
    149893 test1.xml#element(/1/1/17/343/2/3)
    150085 test1.xml#element(/1/1/17/344)
    150179 test1.xml#element(/1/1/17/344/3)
```

Using a text editor, the positions were analysed. These stored file positions all correspond to their XPointer expression correctly.

C.2.2 Test 2

Output:

```
Time taken to process: 1mins8s
```

C.2.3 Test 3

Output:

```
Time taken to process: 1mins24s
Time taken to additionally process: 31ms
=== Element offset table ===
    offset xpointer
        89 LargeExpr.mml#element(/1/1)
       105 LargeExpr.mml#element(/1/1/2)
1000250124 LargeExpr.mml#element(/1/1/3)
```

This performs the additional processing quickly; in 31ms.

C.2.4 Test 4

Num. items rendered	doc processing (ms)	transforming (ms)	rendering (ms)
Initial (4)	78	1008	2450
5	0	16	16
6	0	0	31
7	0	16	16
8	16	0	16
9	15	0	16
10	0	16	15
11	15	0	32
12	0	0	32
13	16	0	31
14	16	0	31
15	0	16	31
16	0	0	47
17	0	16	47
18	0	16	47
19	0	31	62
20	0	16	62
21	0	0	78
22	0	0	78
23	0	0	172
24	15	0	94
25	15	0	94
26	15	0	109
27	0	15	110
28	0	0	141
29	0	0	140
30	15	0	141
31	16	0	156
32	63	0	187
33	0	16	187
34	0	16	187
35	15	0	219
36	15	0	250
37	16	0	250

Num. items rendered	doc processing (ms)	transforming (ms)	rendering (ms)
38	16	0	281
39	0	0	359
40	16	0	312
41	0	16	328
42	16	0	359
43	0	15	391
44	16	0	422
45	15	0	453
46	0	16	469
47	16	0	516
48	0	16	547
49	0	16	578
50	16	0	625
51	0	0	672
52	15	0	704
53	0	0	766
54	0	0	859
55	16	0	843
56	16	0	890
57	16	0	953
58	16	0	1020

Appendix D

User Documentation

The user interface is fairly intuitive, this chapter outlines the basic process of browsing a MathML-C expression.

D.1 Loading a file

Use File → Load to load a content MathML file.

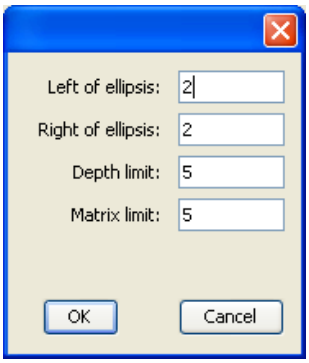

Figure D.1: Initial thresholds

After selecting the file, the user is prompted for some elision thresholds. (Figure D.1)

D.2 Interaction

After initially processing the MathML-C document, a summary is displayed. (Figure D.2)

Width elisions (with an ellipsis) can then be quickly navigated across by left clicking (after selecting the appropriate arrow icon on the toolbar). Other options are also available on a context menu when right clicking an expression. These include expanding and inserting substitutions as well as expanding the entire element. For substitutions, the default left clicking function is to expand the elided expression.

When a matrix is too large to be displayed, it can be viewed as a bitmap image, as seen in figure D.3. Zero entries in the matrix are represented as white, other constants as black, and mathematical expressions as grey entries.

D.3 Other options

The application also provides an option to export the presentation MathML that has been rendered via the File → Export menu.

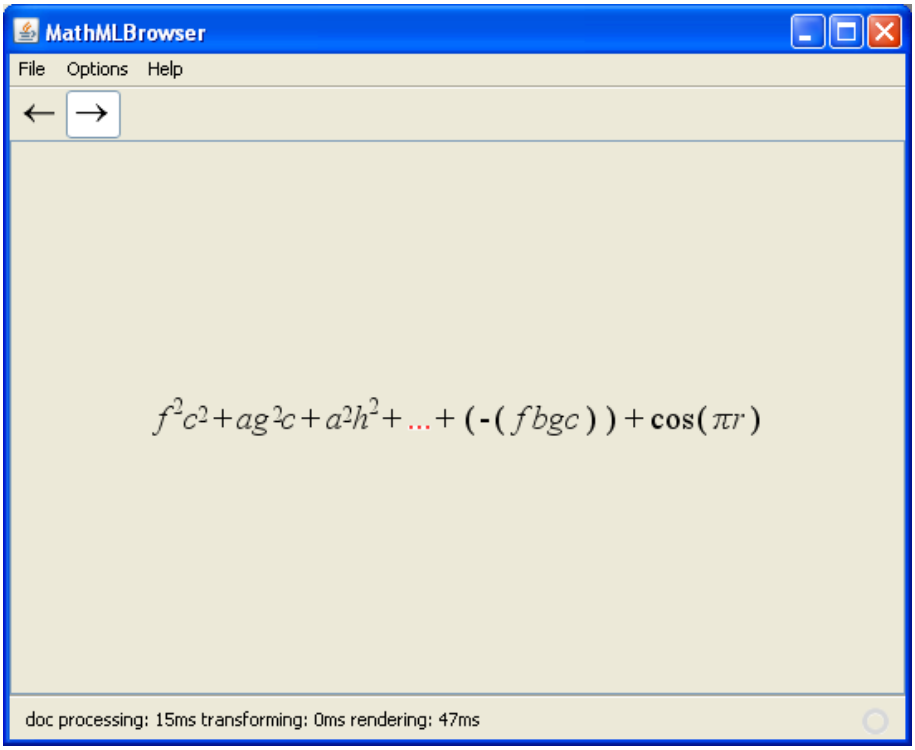

Figure D.2: Initial summarisation

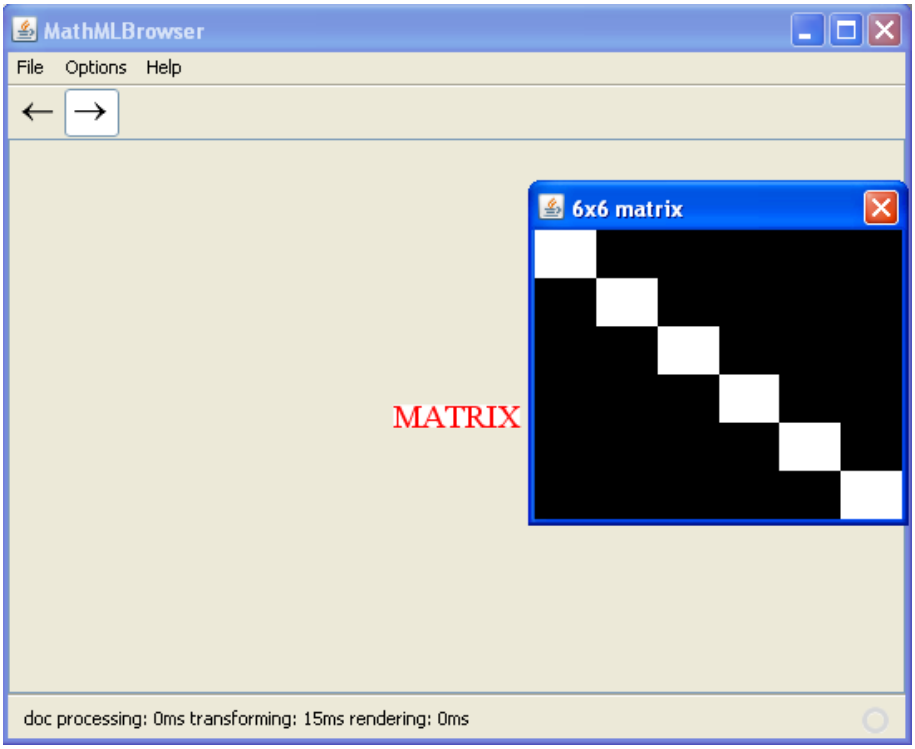

Figure D.3: Matrix bitmap view

Appendix E

Code

E.1 Large MathML generating code

E.1.1 File: CreateMatrix.java

```
package largematrix;

import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintStream;

public class CreateMatrix
{

    final static int M = 10000,  N = 10000;
    final static String fileName = "LargeMatrix.mml";

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        try {
            FileWriter writer = new FileWriter(fileName);
            writer.write("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");
            writer.write("<math xmlns='http://www.w3.org/1998/Math/MathML'>\n");
            writeMatrix(writer);
            writer.write("</math>\n");
            writer.close();
        } catch (IOException ex) {
            System.out.println("File IO error with " + fileName);
        }
    }

    public static void writeMatrix(FileWriter writer)
            throws IOException
    {
        writer.write("<matrix>\n");

        for (int i = 0; i < M; i++) {
            System.out.println("Writing row " + i);
            writer.write("<matrixrow>\n");

            for (int j = 0; j < N; j++) {
                writer.write("<cn>" + (i == j ? "1" : "0") + "</cn>");
            }

            writer.write("</matrixrow>\n");
        }

        writer.write("</matrix>\n");
    }
}
```

E.1.2 File: CreateWideExpr.java

```
package largematrix;

import java.io.FileWriter;
import java.io.IOException;

public class CreateWideExpr
{
    final static String fileName = "test5.mml";
    final static int numTerms = 100;
    public static void main(String[] args)
    {
        try {
            FileWriter writer = new FileWriter(fileName);
            writer.write("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");
            writer.write("<math xmlns='http://www.w3.org/1998/Math/MathML'>\n");
            writeExpr(writer, numTerms);
            writer.write("</math>\n");
            writer.close();
        } catch (IOException ex) {
            System.out.println("File IO error with " + fileName);
        }
```

```
    }

    public static void writeExpr(FileWriter writer, int count)
            throws IOException
    {
        writer.write("<apply><plus/>\n");

        for(int i = 0; i < count; i++) {
            writer.write("<ci>");
            writer.write("c" + i);
            writer.write("</ci>\n");
        }

        writer.write("</apply>\n");
    }
}
```

E.1.3 File: CreateExprWithLargeMatrix.java

```
package largematrix;

import java.io.FileWriter;
import java.io.IOException;

public class CreateExprWithLargeMatrix
{
    final static String fileName = "LargeExpr.mml";
    public static void main(String[] args)
    {
        try {
            FileWriter writer = new FileWriter(fileName);
            writer.write("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");
            writer.write("<math xmlns='http://www.w3.org/1998/Math/MathML'>\n");
            writer.write("<apply><times/>\n");
            CreateMatrix.writeMatrix(writer);
            CreateWideExpr.writeExpr(writer, CreateWideExpr.numTerms);
            writer.write("</apply>\n");
            writer.write("</math>\n");
            writer.close();
        } catch (IOException ex) {
            System.out.println("File IO error with " + fileName);
        }
    }
}
```

E.2 Traversing the document design tests

E.2.1 File: VTDXML.java

```
package largematrix;

import com.ximpleware.extended.*;

public class VTDXML
{
    public static void main(String[] args)
    {
        try {
            XMLMemMappedBuffer xb = new XMLMemMappedBuffer();
            VTDGenHuge vg = new VTDGenHuge();
            xb.readFile(CreateMatrix.fileName);
            vg.setDoc(xb);
            vg.parse(true);
            VTDNavHuge vn = vg.getNav();
            //if (vg.parseFile(CreateMatrix.fileName, false,
            //VTDGenHuge.MEM_MAPPED)) {
            //VTDNavHuge vn = vg.getNav();
            AutoPilotHuge ap = new AutoPilotHuge(vn);

            ap.selectXPath("matrixrow[5000]/*[5000]");
            int result = -1;
            int count = 0;
            while ((result = ap.evalXPath()) != -1) {
                System.out.print("" + result + " ");
                System.out.print("Element name ==> " + vn.toString(result));
                int t = vn.getText(); // get the index of the text (CDATA)
```

```
                if (t != -1) {
                    System.out.println(" Text ===> " + vn.toNormalizedString(t));
                }
                System.out.println("\n ==========");
                count++;
            }

            System.out.println("Total # of element " + count);
        } catch (Exception ex) {
            ex.printStackTrace();
        }

    //}

    }
}
```

E.2.2 File: simpleElision.xsl

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

<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:app="customSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="customSchema customSchema.xsd"
    xmlns:mml="http://www.w3.org/1998/Math/MathML"
    xmlns="http://www.w3.org/1998/Math/MathML"
    version="1.0" exclude-result-prefixes="mml xsi app"
        >
    <xsl:output method="xml" indent="yes" omit-xml-declaration="no"/>

        <!--
                maximum allowed args of math expression
                - min val of threshwidth should be 3, 1 term + elision term + 1 term
        -->
        <xsl:param name="threshwidth" select="3"/>

        <!--
                no. args left/right of width elision
                - the default values are for
                        # left of elision = # right of elision (threshwidth odd)
                        # left of elision = # right of elision + 1 (threshwidth even)
                - should override both or none
        -->
    <xsl:param name="threshwidthL" select="floor($threshwidth div 2)" />
    <xsl:param name="threshwidthR" select="$threshwidth - $threshwidthL - 1" />

    <xsl:template match="mml:apply">
        <xsl:variable name="numchild" select="count(*)"/>
        <xsl:copy>
            <xsl:copy-of select="@*"/>
            <xsl:choose>
                <xsl:when test="($numchild &gt; ($threshwidth + 1))
                                and ($numchild &gt; ($threshwidthL + $threshwidthR + 2))">
                    <xsl:attribute name="app:elision">width</xsl:attribute>

                    <xsl:call-template name="apply_elide_width">
                        <xsl:with-param name="node" select="."/>
                        <xsl:with-param name="numleft" select="$threshwidthL"/>
                        <xsl:with-param name="numright" select="$threshwidthR"/>
                    </xsl:call-template>
                </xsl:when>

                <xsl:otherwise>
                    <xsl:attribute name="app:elision">none</xsl:attribute>
                    <xsl:apply-templates/>
                </xsl:otherwise>
            </xsl:choose>
        </xsl:copy>
    </xsl:template>

        <xsl:template match="mml:matrix">
                <ci type="matrix">
                        <xsl:attribute name="app:elision">matrix</xsl:attribute>
                        <xsl:text>M</xsl:text>
                </ci>
        </xsl:template>

    <xsl:template name="apply_elide_width">
        <xsl:param name="node" />
```

```
        <xsl:param name="numleft"/>
        <xsl:param name="numright"/>
        <xsl:apply-templates select="$node/*[1]"/>
        <xsl:apply-templates
            select="$node/*[(position()&gt;1) and (position()&lt;=($numleft + 1))]"/>
        <ci>...</ci>
        <xsl:apply-templates
            select="$node/*[(position()&gt;=(last()-($numright - 1)))]"/>
    </xsl:template>

    <!--<xsl:template name="make_elision_attr">
        <xsl:param name="elision"/>
        <xsl:attribute name="app:elision">
            <xsl:choose>
                <xsl:when test="$elision = ''">
                    <xsl:text>none</xsl:text>
                </xsl:when>
                <xsl:otherwise>
                    <xsl:value-of select="$elision"/>
                </xsl:otherwise>
            </xsl:choose>
        </xsl:attribute>
    </xsl:template>-->

    <xsl:template match="node()|@*">
        <xsl:copy>
            <xsl:apply-templates select="@*"/>
            <xsl:apply-templates/>
        </xsl:copy>
    </xsl:template>
</xsl:stylesheet>
```

E.2.3 File: test.stx

```
<stx:transform xmlns:stx="http://stx.sourceforge.net/2002/ns" version="1.0"
    pass-through="none" output-method="xml">

  <!--<stx:template match="matrix">
    <cn type="matrix">M</cn>
  </stx:template>-->

</stx:transform>
```

E.3 File: mathmlc2p_modifier.xsl

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

<!--
    This template is used to modify the JEuclid mathmlc2p.xsl (content MathML to
    presentation MathML converter) so that our custom app:link and app:elision
    attributes are copied from the content MathML on transformation onto the
    presentation MathML.

    This stylesheet adds a call to the template "transfer_app_attr" to the 1st
    MathML-P child of each XSL template in mathmlc2p.xsl. Issues arise in the
    templates in mathmlc2p.xsl with the matches:
    m:apply[*[1][self::m:compose]]
        - There is no mrow surrounding the entire expression; this causes a
          call to transfer_app_attribute in the for-each loop.
        - Either mathmlc2p.xsl should add a mrow around the template or the
          resulting new stylesheet should be rectified.
    m:apply[*[1][self::m:apply]]
        - The call to transfer_app_attribute is placed around the entire
          expression (when there is >=2 children). For a more user friendly
          and intuitive solution, move this call into the 2nd mfenced
          expression. (i.e. so arguments and function can be handled seperately)

    The new stylesheet obtained also has the XSL parameters 'backgroundCol' and
    'mathCol' so that the colors of highlights can be customised.
-->

<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:app="customSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="customSchema customSchema.xsd"
    xmlns:mml="http://www.w3.org/1998/Math/MathML"
    xmlns="http://www.w3.org/1998/Math/MathML"
```

```
    version="1.0" exclude-result-prefixes="mml xsi app"
        >
    <xsl:output method="xml" indent="yes" omit-xml-declaration="no" encoding="ascii"/>

    <!--
        Add transfer_app_attr template to copy custom app:link and app:elision
        attributes and highlight specified elisions.
    -->
        <xsl:template match="xsl:stylesheet">
                <xsl:copy>
                        <xsl:copy-of select="@*"/>
                        <xsl:text disable-output-escaping="yes">
                &lt;xsl:param name="backgroundCol"&gt;white&lt;/xsl:param&gt;
                &lt;xsl:param name="mathCol"&gt;red&lt;/xsl:param&gt;
                                &lt;xsl:template name="transfer_app_attr"
                    xmlns:app="customSchema"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="customSchema customSchema.xsd"&gt;

                                        &lt;xsl:param name="node" select="."/&gt;
                    &lt;xsl:copy-of select="$node/@app:elision|$node/@app:link"/&gt;
                    &lt;xsl:if test="starts-with($node/@app:elision,'ellipsis') or
____________________________________starts-with($node/@app:elision,'sub') or
____________________________________starts-with($node/@app:elision,'matrix')"&gt;
                        &lt;xsl:attribute name="mathbackground"&gt;
                            &lt;xsl:value-of select="$backgroundCol"/&gt;
                        &lt;/xsl:attribute&gt;
                        &lt;xsl:attribute name="mathcolor"&gt;
                            &lt;xsl:value-of select="$mathCol"/&gt;
                        &lt;/xsl:attribute&gt;
                    &lt;/xsl:if&gt;
                                &lt;/xsl:template&gt;
                        </xsl:text>
                        <xsl:apply-templates select="node()"/>
                </xsl:copy>
        </xsl:template>

    <!--
        Add transfer_app_attribute to templates matching 'm:' or with a 'name'
        attribute.
    -->
        <!--<xsl:template match="xsl:template[starts-with(@match,'m:apply')]">-->
        <xsl:template match="xsl:template[starts-with(@match,'m:') or @name]">
                <xsl:call-template name="mark-first-mml"/>
        </xsl:template>

    <!--
        Mark first child of MathML in the stylesheet with the
        transfer_app_attribute template.
    -->
        <xsl:template name="mark-first-mml">
                <xsl:copy>
                        <xsl:copy-of select="@*|text()"/>
                        <!--<xsl:if test="count(*) = 0">
                                <xsl:copy-of select="text()"/>
                        </xsl:if>-->
                        <xsl:for-each select="*">
                                <xsl:choose>
                                        <xsl:when test="namespace-uri() =
                                                'http://www.w3.org/1998/Math/MathML'">
                                                <xsl:copy>
                                                        <xsl:copy-of select="@*|text()"/>
<xsl:text disable-output-escaping="yes">&lt;xsl:call-template
     name="transfer_app_attr"/&gt;</xsl:text>
                                                        <xsl:copy-of select="*"/>
                                                </xsl:copy>
                                        </xsl:when>
                                        <xsl:otherwise>
                                                <xsl:call-template name="mark-first-mml"/>
                                        </xsl:otherwise>
                                </xsl:choose>
                        </xsl:for-each>
                </xsl:copy>
        </xsl:template>

    <xsl:template match="node()|@*">
        <xsl:copy>
            <xsl:apply-templates select="@*"/>
            <xsl:apply-templates/>
        </xsl:copy>
    </xsl:template>
</xsl:stylesheet>
```

E.4 Other code

The remaining source code can be found on the accompanying CD.