Exploiting Implicit Mathematical Semantics in Conversion between TEX and MathML

Stephen M. Watt
Ontario Research Centre for Computer Algebra
University of Western Ontario
London Canada, N6A 5B7
http://www.csd.uwo.ca/∼watt

Abstract

We present a new strategy for conversion between TEX mathematical expressions and MathML. Previous efforts have attempted either superficial high level transliteration or full execution of TEX. We observe that document markup typically uses macros that relate to semantics, and show how this structure can be exploited in the translation process. We present the overall architecture of two implementations that take advantage of this information: a converter from TEX to MathML and another from MathML to TEX. Our implementation allows control over which macros are expanded and which are mapped at a semantic level.

1 Introduction

MathML [1] has emerged as an important representation for mathematics in digital documents: Web pages containing MathML may be rendered by various web browsers, including the most popular ones. Major computer algebra systems can import and export MathML. MathML is the representation for mathematical formulae in all new US patents.

At the same time, TEX [2] remains important: To this day, it is the best typesetting engine for mathematical expressions. An enormous legacy of mathematical documents exist marked up in TEX, and some publishers that use SGML use embedded TEX islands for the mathematics.

Translation between TEX mathematics and MathML is an important problem, and will continue to be so for the foreseeable future: Translation from MathML to TEX is, at present, an important direction to obtain high-quality typeset mathematics from MathML. High-quality translation from TEX to MathML will be important for mathematical document management with XML tools. This will also provide a bridge between mathematical documents and mathematical computation systems.

Currently, mathematicians are well versed in composing papers in TEX, and it will be some time before tools are widespread for easy document preparation in XML with MathML. Conversion of TEX to MathML is necessary for the treatment of legacy documents and legacy authors in an XML environment.

This paper explores an approach to TEX/MathML conversion that can produce high level markup as output. We observe that important semantic concepts are often present in the source markup in the form of macro applications. Our approach is unique in retaining this implicit semantic information as the documents are translated. The translator output may be used immediately by applications, or can be adopted as "replacement source" for on-going work.

The principal contributions of this work are:

- A new approach to TEX/MathML translation that conserves and exploits implicit semantics.
- An architecture that allows new representations of semantic information to be associated with existing documents (e.g. association of OpenMath definitions to existing LATEX documents).
- Useful converters from TEX math to MathML and MathML to TEX, and an examination of issues that arise in their implementation.

The remainder of this paper is organized as follows: In Section 2 we describe previous approaches to the conversion between TEX and MathML. Section 3 shows how implicit semantics are conveyed by the use of macros. Section 4 discusses how these implicit semantics can be used for high level mappings in TEX/MathML conversion. Section 5 gives an example of a such a mapping and Section 6 outlines our mapping file format. Building on this, Section 7 describes aspects of our implementation of TEX to MathML conversion, and Section 8 describes aspects of our translation from MathML to TEX. Finally, we present our conclusions in Section 9.

2 Other Approaches to Conversion

Previous efforts have concentrated on the uni-directional conversion of TEX to MathML. For this, the simplest, naïve approach is to use an editing script, e.g. in `sed` or `Perl`, to transliterate known TEX commands to MathML. This may be sufficient for simple applications, but quickly runs into a number of problems:

- TEX macros may be defined by packages that are selected by the document
- TEX macros may be defined within the document
- TEX includes a full-fledged programming language that can be used to make decisions dynamically about what text should appear and how.

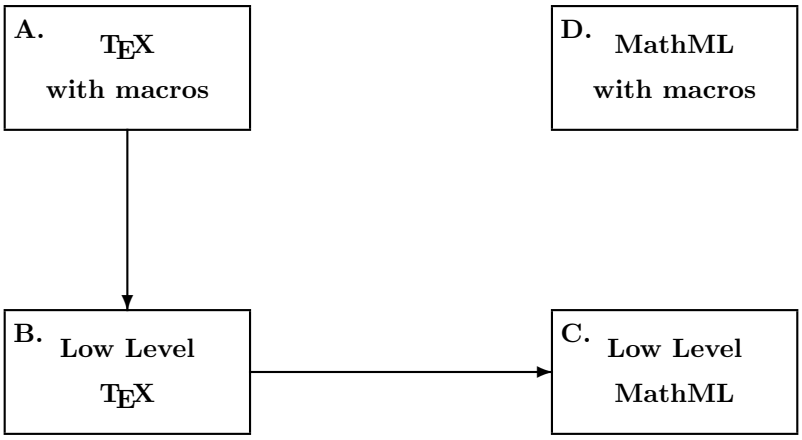

Figure 1: TEX macro expansion, followed by conversion

Typical TEX documents make frequent use of package-defined or user-defined macros, so any serious TEX mathematics to MathML converter must handle TEX macros.

The next most obvious approach to TEX to MathML conversion is to process all TEX macros, and then convert the resulting low level TEX to MathML. This is illustrated in Figure 1. The MathML that results from this translation is low level and presentation-oriented, and does not normally reflect the natural grouping structure desired. Inferring high level MathML from the result (adding C→D in Figure 1) would be a challenging problem in artificial intelligence.

A second problem with this approach is that handling general TEX macros is difficult. Although the user level appearance of TEX source can be quite structured and high level, especially when LATEX is used, it is not possible to rely on this apparent structure in a program that handles TEX robustly. Loaded packages may have complex macro definitions that rely on the full generality of TEX, and which redefine TEX's usual behaviours. For this reason, programs that hope to handle TEX macros properly must effectively provide a full TEX implementation. Programs that provide only a subset with the commonly used features of TEX inevitably run into trouble.

One of the most successful prior strategies in translating TEX has been to use a full implementation of TEX to perform the macro expansion. Alternative versions of various macros are provided that generate identifiable objects in TEX's output. These are then used to form the desired result. This is the approach used, e.g., by TeX4ht [3]. As before, because the TEX macros are fully evaluated, the resulting MathML is typically quite low level.

We conclude that macro expanding then translating is best suited to display and does not provide converted documents suitable for further use.

3 Macros and Implicit Semantics

Both TEX and MathML admit macro mechanisms: natively so in TEX and via XSLT [4] with MathML. Authors may use pre-defined style sheets or define their own abbreviations, effectively extending the vocabulary of the environment. Macros are typically used as shorthands for lengthy expressions or to maintain notational independence. A simple example of notational independence would be, *e.g.*, to define `\Vector` to expand either to `\mathbf{v}` or `\vec{v}` or something else, depending on the style sheet used.

The starting point for the present work is the observation that the use of macros in a document's source often carries semantic information. We therefore see the strategy of expanding all macros, then translating, as fundamentally lacking. Instead, we believe it is much more desirable to retain the implicit semantic information in the result of any translation.

To illustrate, consider the following TEX input

```
\newcommand{\BesselJ}[1]{J_{#1}}
$$
\BesselJ3(z)=\left(\frac8{z^2}-1\right) \BesselJ1(z) - 4\BesselJ0(z)/z
$$
```

which produces the expression

$$J_3(z) = \left(\frac{8}{z^2} - 1\right) J_1(z) - 4J_0(z)/z$$

When `\BesselJ\alpha` expands to J_α and this translates to

```
<msub><mi>J</mi><mi>&alpha;</mi></msub>
```

then we have lost knowledge about J. The MathML application has no way to determine that J is a Bessel function and not, *e.g.*, a jet bundle, a current or angular momentum vector.

Implicit semantics have been seen to be quite common in practice [5]. In some cases, we can have a rather good idea about the semantic concept associated with a macro. In other cases, it will not be clear whether a macro corresponds to a semantic idea or whether it is simply an abbreviation, e.g. to save typing a collection of terms. One could argue, even in this case, that the desire to abbreviate a particular set of terms is somehow meaningful.

The decision about which macros are important to conserve, and which should be expanded, is something that requires some flexibility. We explore this point in the next sections.

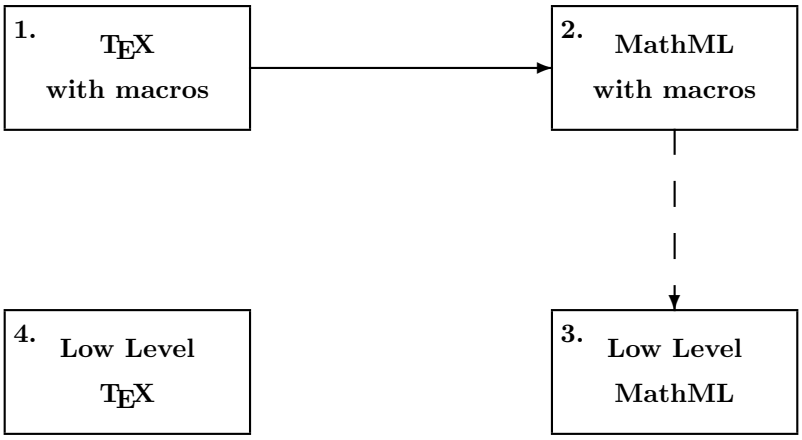

Figure 2: Mapping TEX to MathML+XML, followed by XSLT processing

4 Defining High Level Correspondences

The approach we have taken to conversion is that application of macros in TEX can correspond directly to XML forms in MathML. The TEX macros may be defined in common packages, or be defined by the user. The XML elements may be standard MathML, or be elements in some extension namespace. For the TEX to MathML direction, this approach is illustrated in Figure 2. For MathML to TEX conversion, we take the path $\mathbf{2} \rightarrow \mathbf{1} \rightarrow \mathbf{4}$.

This approach works by specifying bi-directioanl correspondences, such as

```
\Div{#1}   ⇔   <apply><divergence/>#1</apply>
```

We collect these correspondences in *mapping files*, used to control the behaviour of our converters. Before describing their contents, we note several general properties of mapping files:

- With our converters it is possible to have several mapping files, organized as the application sees fit. For example, it is possible to have one mapping file for each standard TEX package, or separate mapping files for different families of mathematical functions.
- The same mapping rules are used to specify conversion in both directions.

  This has the advantage of consistency and ease of maintenance—separate files for the two directions would easily get out of sync. This poses constraints on the forms that may appear in the rules, however, since either side may be used as a pattern and arbitrary rewriting leads to theoretical problems. There is a tension between the desire for more powerful uni-directional rules and the consistency of bi-directional rules.

- Our converters, both from TEX to MathML and from MathML to TEX, will expand any macros not found in a mapping file. When a pattern from a mapping file is found, the corresponding output is generated directly. Our implementations also use mapping files to handle most of the low level, built-in constructs.

- Mapping files may rely on associated helper stylesheets, either on the MathML+XML side, the TEX side, or both. These may be used to associate additional information with the generated document when it is later used. For example, generated MathML+XML could include elements that, when later expanded by XSLT, provide OpenMath annotations.

- In practice, complicated uses of low level TEX or XSLT are typically encapsulated in macro definitions. These abstractions may then be given mapping rules. This is a deliberate element of our design: the possibility of adding direct high level mappings allows us to get by with implementing only a small subset of TEX's programming language.

5 An Example

Mapping files are given in XML form and provide lists of templates representing bi-directional MathML/TEX rewrite rules. Each template is of the form

```
<map:template>
   <map:tex op="⟨TEX name⟩" params="⟨TEX expression⟩"/>
   <map:mml op="⟨XML tag⟩" mode="math|text|spec">
      ⟨MathML+XML expression⟩
   </map:mml>
</map:template>
```

Here, the prefix `map` is a namespace associated to the URI

`http://www.orcca.on.ca/mathml/tex2mml.xml`.

We give an example of how these can be specified and used: Suppose a user has defined two style sheets: a TEX package, `"combinatorics.cls"`, defining `\binom`, and an XSLT stylesheet, `"combinatorics.xsl"`, providing a rule for `<mmlx:binom>`.

The file `"combinatorics.cls"` would contain a definition such as

```
% TeX definition of \binom{..}{..}
\newcommand{\binom}[2]{\left({#1} \atop {#2}\right)}
```

and the file "combinatorics.xsl" would have a template such as

```
<!-- Template for an element <mmlx:binom> -->
<xsl:template match="apply/mmlx:binom[position()=1]">
   <mfenced>
      <mfrac thickness="0ex">
         <xsl:apply-templates select="*[2]"/>
         <xsl:apply-templates select="*[3]"/>
      </mfrac>
   </mfenced>
</xsl:template>
```

A mapping file could give the correspondence between a use of \binom and a use of <mmlx:binom> as follows:

```
<map:template>
   <map:tex op="\binom" params="\patVAR!{a}\patVAR!{b}"/>
   <map:mml op="apply/mmlx:binomial">
      <apply>
         <mmlx:binomial/>
         <map:variable name="a"/>
         <map:variable name="b"/>
      </apply>
   </map:mml>
</map:template>
```

This would be used, e.g., by the MathML to TEX converter to translate

```
<apply>
   <mmlx:binomial/>
   <apply><plus/> <ci>a</ci> <ci>b</ci> </apply>
   <apply><plus/> <ci>c</ci> <ci>d</ci> </apply>
</apply>
```

to

```
\binom{a+b}{c+d}
```

instead of the lower level expression

```
\left(\atop{a+b}{c+d}\right)
```

6 Mapping Files

The structure of mapping files is specified by [6]. Here we summarize some of the main ideas.

A mapping file is a collection of templates, with each template having multiple forms of the same mathematical expression. At the moment, each template has

only two or three branches: a <map:tex> element for TEX, a <map:mml> element for MathML+XML, and possibly a <map:img> element for a graphical image.

The <map:tex> branch

The <map:tex> element must have an op attribute, and may have a params or prec attribute. When converting from TEX, the rule is matched based on the op attribute. If present, the params attribute specifies the TEX macro parameters using a simple pattern matching language suited to TEX. This is necessary because TEX macros may accept parameters in a wide variety of syntaxes. The value of a params attribute is treated as a normal TEX expression, with special handling of the following pseudo-macros which may appear:

Variables are introduced by the \patVAR pseudo-macro. This takes different forms depending on how many items it may match:

- \patVAR!{varname} matches exactly one item
- \patVAR*{varname} matches zero or more items
- \patVAR+{varname} matches one or more items
- \patVAR{varname} matches zero or one items

Repetition is specified by the \patREP pseudo-macro. It may take either of two forms:

- \patREP*{...} repeats a pattern zero or more times
- \patREP+{...} repeats a pattern one or more times

Uses of \patREP may be nested and may contain \patVARs.

As an example, the TEX \matrix command may be specified as

```
<map:tex
   op="\matrix"
   params="{\patREP+{\patVAR+{firstCol}\patREP*{&amp;\patVAR+{rest}}\cr}}"
/>
```

The use of the pseudo-macros \patVAR and \patREP must correspond to uses of <map:variable> and <map:rep> in the <map:mml> branch, if present.

When matching TeX expressions, it is possible to have more than one template whose <map:tex> branch applies. In this case, the value of the prec attribute is used to prioritize the matching rules.

The <map:mml> branch

The <map:mml> element must have an op attribute specifying the element the template is to match. The element has one child, which is a MathML+XML

fragment giving a prototypical use of the element. This fragment may contain `<map:variable>` and `<map:rep>` elements specifying pattern variables and repetitions. These are simpler than in the TEX branch because XML source has a more explicit structure.

As an example, the MathML branch for TEX `\matrix` example above would be

```
<map:mml op="mtable">
   <mtable>
     <map:rep>
       <mtr>
         <mtd> <map:variable name="firstCol"/> </mtd>
         <map:rep>
           <mtd> <map:variable name="rest"/> </mtd>
         </map:rep>
       </mtr>
     </map:rep>
   </mtable>
 </map:mml>
```

7 TEX to MathML

We outline here some of the main aspects of our TeX to MathML converter, and issues that have arisen in its implementation.

The converter is a program, written in Java, and with interfaces to allow command line or web-based invocation. Different entry points allow it to be invoked either on single TEX math expression or with a full TEX document. If a full TEX document is given, then it may be organized in any manner, with multiple files in various directories, and using packages, classes and style files from various locations.

The converter handles math mode only, and does not attempt to translate what occurs outside of mathematical expressions. Our design objective has been to have a converter that handled math mode well, and could be used in conjunction with other tools.

The first step performed by the converter is to expand macros that occur anywhere in the document to find all math mode islands. Handling macros in the entire document is necessary since macros may expand to text containing math. Macro expansions that do not contain any mathematics are reverted to the original un-expanded TEX. The result is a TEX document with all of the mathematics manifestly apparent.

The second step is to process each of the mathematical expressions. This process will be described shortly.

The third step is to produce the desired resulting document. Depending on how the converter was invoked, this can be

- a single MathML expression,
- an XML structure containing all the converted TEX math islands, or
- a TEX document with the in-line math replaced by `\verbXML"..."` and the display math replaced by `\begin{verbatimXML}...\end{verbatimXML}`.

  The resulting TEX document has the same structure as the original source: either a single file, or a collection of files in some tree structure. The resulting document may be processed by other tools, such as `LaTeX2HTML`.

In all cases the converter output is MathML, possibly including additional XML elements in extension namespaces.

The core of this process is the conversion of a single TEX math expression to MathML. This is accomplished by evaluating the TEX macros that occur in the math expression. Macros with rules in any of the mapping files are translated directly to XML. Although some primitive TEX commands are handled directly by hard-coded routines, the translation of most of the basic elements is also handled by a mapping file, in the same format, for primitives.

Ultimately, the result of the expansion is an XML fragment. This will, depending on the mapping files used, contain some combination of Presentation MathML, Content MathML and whatever extension elements were given in the applied mapping file rules (e.g. `<mmlx:binom>`).

Since we have not provided a full TEX implementation, the expansion of some macros may result in TEX constructs we do handle. In this case, the user should provide an additional mapping file specifying what this macro corresponds to at a high level. Our design rationale is that if a macro's implementation involves too many low level constructs, then the result will likely have lost its implicit semantic intent.

The final step in the conversion of a TEX math expression to MathML+XML is to perform certain expression re-organizations. This is necessary for two reasons:

First, we wish to remove structure that is an artefact of the TEX environment: Because TEX does not have automatic line-breaking, authors must break large expressions by hand. Some of the breaks for multi-line layout carry semantics, e.g.

$$\begin{aligned} P_i(t) &= \int_0^{\nu_i(t)} p_i(\tau)\mathrm{d}\tau \\ Q_i(t) &= \int_0^{\nu_i(t)} q_i(\tau)\mathrm{d}\tau \end{aligned}$$

Other breaks are solely to wrap long lines, e.g.

$$\begin{aligned} P_*(t) + Q_*(t) &= 1 - (1 - a_1)e^{-\lambda_2 t/2} - (1 - a_2)e^{-\lambda_1 t/2} \\ &\quad +(1 - a_1 - a_2)e^{t(\lambda_1+\lambda_2)/2} \end{aligned}$$

We wish to preserve the former and eliminate the later.

Second, we wish to add structure that is missing from the TEX source: Although mathematical expressions typeset with TEX look as though they are grouped according to some mathematical precedence, they really aren't. It is the spacing rules common mathematical operators that give the expressions this appearance. For example, the TEX input `$$(x+1)^2=x^2+2x+1$$` displays as

$$(x + 1)^2 = x^2 + 2x + 1$$

Note the extra space around the "+" and "=" characters. This leads the eye to parse the expression. Also note that the expression $(x + 1)$ appears to have a superscript 2. The actual grouping of the TEX tree, however, is as follows

That is, we have a single node containing 12 elements, two of which are the composite nodes for "$)^2$" and "x^2". We *re-associate* the tree so that it is the subexpression $(x+1)$ that is squared, not just the parenthesis, and *refine* the tree to insert grouping `<mrows>` corresponding to operator precedence. It would be very difficult to do this in complete generality, so we simply attempt to handle the common cases.

8 MathML to TEX

We discuss here a few properties of our MathML to TeX converter, and issues that have arisen in its implementation.

The converter is able to handle the full range math formulas, including matrices, multi-line equations and equation arrays. It is structured to be able to share as much code as possible with the TEX to MathML converter, including the classes to represent TEX and XML objects. The converter may be run both in command line mode and with a GUI that allows nodes to be expanded interactively.

The converter is driven from the same set of mapping files as the TEX to MathML converter. Users may elect to provide there own mapping files in addition to, or instead of, the standard ones we provide.

The main problems the converter faces are related to a similar problem that computer algebra systems have for output: how to deal with large expressions in various contexts.

The first of these problems is line breaking. We select appropriate places to break the expression into multi-line output according to the expression's tree structure. For example, in the expression

$$a \cdot b \cdot c + d \cdot e \cdot f + g \cdot h \cdot i,$$

it is preferable to split at a plus sign rather than at a mulitplication dot. There is also the question of how to deal with operators at the break. According to different notational conventions, the operator may appear at the end of the first line, at the beginning of the second line or both, i.e.

$$\begin{array}{lll} a-b- & a-b & a-b- \\ c-d & -c-d & -c-d \end{array}$$

This might seem like a simple point, but the meaning of the expression depends on choice of convention when certain operators are doubled.

The second problem is dealing with large subexpressions in two-dimensional layout schema. This occurs, for example, when a large expression occurs as the lower limit of a sum, as the degree of a radical symbol, or as the numerator of a built-up fraction. Our solution is to define a box for the subexpression and wrap the subexpression to fit, e.g.:

$$\sum_{\begin{array}{l}\ell = a+b+c+d+ \\ e+f+g+h+i+ \\ m+n+o+p+q\end{array}}^{n} \ell^n n^\ell$$

$$\frac{\left(\begin{array}{l} x^{-1024}+1+x+x^2+x^3+x^4+x^5+x^6+x^7+x^8+x^9+ \\ x^{10}+x^{11}+x^{12}+x^{13}+x^{14}+x^{15}+x^{16}+x^{17}+x^{18}+x^{19}+ \\ x^{20}+x^{21}+x^{22}+x^{23}+x^{24}+x^{25}+x^{26}+x^{27}+x^{28}+x^{29}+ \\ x^{30}+x^{31}+x^{32}+x^{33}+x^{34}+x^{35}+x^{36}+x^{37}+x^{38}+x^{39} \end{array}\right)}{m+n+o+p}$$

9 Conclusion

We have shown how library and user-defined macros can carry implicit semantics in TEX and MathML documents, and we have shown how these implicit semantics can be exploited in the translation between TEX and MathML.

We have presented our implementation of translators from TEX to MathML, and from MathML to TEX, and have discussed some of the issues that arise in their implementation. We have shown how our implementation makes use of "mapping files" to identify high level correspondences between application of TEX macros and XML elements. Our translators use these mapping files to carry the high level semantic markup through TEX/MathML translations

while expanding macros that do not carry useful meaning. These translators are available at [7].

Is this the final word? Once we have translated all of our legacy TEX documents to semantically rich MathML, have we solved the translation problem? In our opinion, we are only at the early stages of an on-going evolution. Over time, there will be many new representations for the storage and exchange of mathematical objects. For these future representations, it is important that we conserve whatever mathematical semantics are present as we translate our current documents into new forms.

Acknowledgments

The implementation of the converters described in this article has been a group effort. The author thanks Igor Rodionov as primary implementor of the TEX to MathML converter and Elena Smirnova as primary implementor of the the MathML to TEX converter.

References

[1] *Mathematical Markup Language (MathML) Version 2.0*, D. Carlisle, P. Ion, R. Miner and N. Poppelier (editors), R. Ausbrooks, S. Buswell, S. Dalmas, S. Devitt, A. Diaz, R. Hunter, B. Smith, N. Soiffer, R. Sutor and S. Watt, *W3C Recommendation 21 Feb 2001*, `http://www.w3.org/TR/MathML2`.

[2] *The TEXbook*, Donald Knuth, Volume A of *Computers and Typesetting*, Addison-Wesley, Reading Massachusetts, second edition, 1984.

[3] *From LaTeX to MathML and Back with TeX4ht and PassiveTeX*, Eitan Gurari and Sebastian Rahtz, Presentation materials from talk at *MathML International Conf. 2000*, 20-21 Oct 2000, Urbana-Champaign USA `http://www.mathmlconference.org/2000/materials/Gurari.zip`

[4] *XSL Transformations (XSLT) Version 1.0*, James Clark (editor), *W3C Recommendation 16 November 1999*, `http://www.w3.org/TR/xslt`.

[5] Boeing corpus of mathematical TEX, Ivor Phillips (personal communication).

[6] *The TEX/MathML Map File Specification*, Ontario Research Centre for Computer Algebra, `http://www.orcca.on.ca/MathML/texmml/MapSpecWeb.html`

[7] *Software developed at ORCCA*, `http://www.orcca.on.ca/MathML/software.html`