How (and why) I use GLOSS to write XHTML+MathML

Richard Kaye, School of Mathematics, University of Birmingham

2006-05-24

'A discerning friend of mine,' said Don Quixote, 'was of opinion that no one ought to waste labour in glossing verses; and the reason he gave was that the gloss can never come up to the text, and that often or most frequently it wanders away from the meaning and purpose aimed at in the glossed lines; and besides, that the laws of the gloss were too strict, as they did not allow interrogations, nor "said he," nor "I say," nor turning verbs into nouns, or altering the construction, not to speak of other restrictions and limitations that fetter gloss-writers, as you no doubt know.'

from Don Quixote,
by Miguel de Cervantes,
Translated by John Ormsby

1 XML

XML is a general format for exchange of information between computer systems. It was originally devised as a 'light' version of SGML intended to present complex structured data containing the meaning or other information suggesting possible rendition of each individual part. Thus the presentation-MathML (p-MathML) code for $a + 3 = \beta$,

```
<mrow>
  <mrow>
    <mi>a</mi><mo>+</mo><mn>3</mn>
  </mrow>
  <mo>=</mo>
  <mi>&beta;</mi>
</mrow>
```

indicates that `a` and `&beta;` are identifiers or variables (and probably will be type-set in a font suitable for variables), `3` is a number (to be type-set in another font) and `+`, `=` are operators (with some extra space around them). The `mrow` delimits the sub-expressions so that the whole thing can be unambiguously read.

As XML is intended as a universal medium, there are a great number of computer systems equipped for reading and using XML data, including systems in

web browsers such as for rendering mathematics to the visually impaired, that we (as authors) have little control over. That means, for mathematics, that we must be much more precise in marking up the individual expressions and subexpressions than we are accustomed to. Typing around eighty characters for the ambiguous (but conventional) `a+3=beta` seems a lot. And it is in fact worse: I still haven't pointed out that in fact I intended the objects 'a' and 'beta' to be elements of the field with two elements, addition is addition modulo 2, equality is congruence modulo 2, and '3' means the equivalence class modulo 2 of the number 3 (which, of course, is the same as the equivalence class of '1'). We will have to say all of this (and we can, using other XML mark-up from OpenMath or content-MathML) if we are going to type our mathematics in a way that can unambiguously be copied and pasted into a computer algebra system. Being able to express and use expressions like this in a wide variety of systems is a major advantage, but one that comes with an apparent burden attached to it. Actually, even before we get into these details, just typing plain XML is awkward: those closing tags must be present and nested correctly, and any error may stop the application working.

We may set our sights lower, and not cater for such a wide range of systems. It certainly is true that MathML can be used in many ways, from quick but rather ambiguous mark-up that may have limited utility to painstakingly careful mark-up that would take a huge amount of time to write by hand. Whatever compromise is taken here though, it seems to me very necessary that there should be a way of entering the required data accurately enough so that an automatic system can apply appropriate defaults and add the necessary XML code.

One possibility is to use a LaTeX-to-MathML converter, such as TtM or one of the other text-based syntaxes for MathML (many of which use a syntax similar to LaTeX). I have rejected these for my own personal use because: (a) LaTeX source code does not contain enough information for any system to infer the correct output; (b) any use of macros in LaTeX can obfuscate or break the translation process; and (c) such translators never seem to work on my own documents, possibly because of macros, different fonts, or something else. However, as they say, your mileage may vary.

In this article I will look at the case of using GLOSS to author p-MathML embedded in a web page or similar document.

2 GLOSS

GLOSS is a general text-to-XML convertor. It is intended mainly for authors with some basic knowledge of both XML in general and the target XML application they are writing for. In its basic form it enables you to write any XML (including MathML, XHTML, etc.) saving considerably better than 50% of the time and effort. GLOSS uses a syntax based on indentation, like the computer language Python, but unlike TeX and LaTeX. Plain characters and text are delimited by square brackets. (This choice was made as square brackets rarely occur in text, and are easily accessible on most keyboards.) Everything else

in the input is a 'word' or 'token' or 'command', usually producing an XML element with the same name. So the example above would be coded in GLOSS as

```
mrow
  mrow
    mi[a]
    mo[+]
    mn[3]
  mo[=]
  mi[&amp;beta;]
```

To get this to work, save it in a text file called `example.xml.gloss` and run the command-line command `gloss-xml example.xml.gloss` and you should have beautiful XML in a new file called `example.xml`. Note: `&beta;` is the standard MathML name for the unicode character β. If you have a unicode editor you can use the unicode character itself instead.

Attributes are encoded in GLOSS with the construct `@name[text]` so the matrix equation

$$A = \begin{bmatrix} x & y \\ z & w \end{bmatrix}$$

could be encoded in GLOSS with

```
mrow
  mfenced @open[\[] @close[\]]
    mtable
      mtr
        mtd mi[x]
        mtd mi[y]
      mtr
        mtd mi[z]
        mtd mi[w]
```

which gives

```
<mrow>
  <mi>A</mi>
  <mo>=</mo>
  <mfenced open="[" close="]">
    <mtable>
      <mtr>
         <mtd><mi>x</mi></mtd>
         <mtd><mi>y</mi></mtd>
      </mtr>
      <mtr>
         <mtd><mi>z</mi></mtd>
         <mtd><mi>w</mi></mtd>
      </mtr>
    </mtable>
  </mfenced>
</mrow>
```

In p-MathML terminology a 'fence' is a pair of brackets that may change size according to context. Note the use of \ to 'escape' the] character, which would

otherwise be taken to be the end of an empty text block. The other characters that need to be escaped like this are [, {, }, and \. This provides a useful check built in to the system that you remembered the closing] character.

Indentation is very nice most of the time (and many text editors are already set up to utilise it) but sometimes more control is needed. Braces {...} are used in GLOSS to over-ride indentation. The rule is that an XML group cannot cross an open or close brace. So } closes all elements that were opened after the corresponding {. This means that the above example could be encoded as

```
mrow
  mfenced @open[\[] @close[\]] mtable {
      mtr {mtd mi[x]} {mtd mi[y]}
      mtr {mtd mi[z]} {mtd mi[w]}
    }
```

Note that if it wasn't for the { immediately following `mtable` the `mtr` elements would be children of `mfenced`, not `mtable`.

GLOSS also allows you to 'push back' into element-mode when in text mode, like TeX does—unlike normal XML. So, using GLOSS to write XHTML this time, you can write,

```
p [This text is part of an HTML paragraph. Let's
test HTML's [i[italics]] and [b[bold]] mark-up elements.]
```

giving

```
<p>This text is part of an HTML paragraph. Let's
test HTML's <i>italics</i> and <b>bold</b> mark-up elements.</p>
```

This feature is really like a combination of {..} and [..]. The above is equivalent to

```
p {[This text is part of an HTML paragraph. Let's
test HTML's ]{i[italics]}[ and ]{b[bold]}[ mark-up elements.]}
```

but a little clearer and easier to type.

GLOSS is a highly configurable and extensible system. That means you can write your own code (rather like 'macros') to deal with situations like matrices that occur many times over a group of documents to save even more typing. The whole idea of GLOSS is that your plain text is parsed by GLOSS in many different 'modes'; GLOSS will be in a different mode depending on the local context, and 'macros' will be context-dependent. So a new command in maths mode will not impact on what happens in text mode. What's more, you can have as many modes as you like.

I'm not going to explain how to write new modes here. That would be rather too technical for this article. However it is a feature that GLOSS's modes can be arranged into 'modules' and separate modules can be loaded according to needs. As well as the base XML module, there is a base XHTML module (using

the commands `gloss-html` or `gloss-xhtml` instead of `gloss-xml`) and several optional extension modules for XHTML including ones supporting: sections, subsections, etc., and automatic section numbers; definitions, theorems, propositions, lemmas, and proofs; p-MathML; some convenient syntactic 'extensions' to p-MathML which GLOSS maps to standard p-MathML; automatic detection of whether maths should be 'inline' or 'display'; and several more.

For another example, consider

$$\begin{bmatrix} \alpha & \beta \\ -1 & \nabla \end{bmatrix} + A = \begin{bmatrix} x & 45 \\ 3.14159E-2 & w \end{bmatrix}$$

The p-MathML extension module knows all the standard MathML names for individual characters, such as `beta`. It also has names for all the single-letter alphabetical characters and can recognise numbers. It also has a default way to wrap each of these with the appropriate tag from `mi`, `mo`, `mn`. So using XHTML and p-MathML, the paragraph you are reading right now is encoded as

```
p [For another example, consider]
math
  mrow
    mfenced @open[\[] @close[\]]
      mtable
        mtr alpha beta
        mtr -1 nabla
    +
    A =
    mfenced @open[\[] @close[\]]
      mtable
        mtr x 45
        mtr 3.14159E-2 w
p [The p-MathML extension module ... is encoded as]
pre [p \[For another example, consider\]
math
  mrow
     ...
]
```

Note also the use of the `math` command to enter maths mode and insert the MathML `math` element.

I have discovered that, with careful use of the standard XHTML tags, the HTML `class` attribute, some of GLOSS's HTML extension modules and CSS style-sheets, it turns out that standards-compliant XHTML can be used as an excellent format for shorter mathematics papers. That is how I have typed this paper for example, as well as all of my first-year real analysis pages at `http://web.mat.bham.ac.uk/R.W.Kaye/seqser/`. (GLOSS sources for all these pages are available from the web-site.) For longer papers or books there are many other XML formats available to choose from. These include DOCBOOK, TEI, OMDoc—all with XSLT style-sheets to transform to HTML or paper-based formats. Gloss can of course be used to write sources for any of these. Or you devise your own format (based on HTML for exmaple) and tailored to your particular application, as I did for a book I am currently working on—also written using GLOSS.

3 Serving the document

Once you have a beautiful XHTML+MathML document you should be able to view it locally (with Firefox, say). It is also a good idea to *validate* your document. This involves running a standard XML tool that makes some basic checks against a document-type definiton (DTD). The DTD contains basic structural details of the format such as: your root `html` element should have only two children `head` and `body`; you are not taking the square root (`msqrt`) of an HTML anchor; and so on. GLOSS's html modules automatically include references to the correct DTDs, and the GLOSS distribution also contains a simple validator: you may already have a better one on your system. When the document is fully checked and ready, it is time to put this on your web server for others to read.

This turned out to be slightly non-trivial on my system. You may need to check and change the way your web server is set up: ask your web-master to make changes or make changes in your `.htaccess` file. XHTML pages with embedded MathML should be served as mime-type `application/xhtml+xml` and ordinary HTML should be served as `text/html`. I use the file-extension 'xhtml' for the former to distinguish them from the latter, though there doesn't seem to be any consensus on this. Also, to ensure that the maximum number of people (and search engines) can read your pages, a technique known as content-negotiation is useful. This is rather easy to set up in Apache, but requires you to get out of the habit of including the suffix `.html` or `.xhtml` in your web links. See the references below for more on content-negotiation, and the 'installation' notes in my Sequences and Series web pages for an example.

4 Further topics and references

I have only touched on the basics of GLOSS for HTML and p-MathML here in this article. In particular I haven't said anything about how to define semantic content of maths expressions (content-MathML or OpenMath) or how to define other transformations either in GLOSS itself or in XSLT, or in using some other program. These are important topics sadly outside the scope of this article.

One of the design decisions that influenced GLOSS is that it is intended for authors with some basic knowledge of both XML in general and the target XML application they are writing for. There is a somewhat steep learning curve at the beginning, and there are a number of pitfalls for the beginner, but once the system is well-understood productivity should be as good or better than with LaTeX. (It certainly has been for me!) There are many freely available web pages and other resources to help a beginner. Some of the ones I found helpful are also listed here.

- `http://web.mat.bham.ac.uk/R.W.Kaye/gloss/`, the main GLOSS web-pages, including all documentation, and downloadable sources and compiled files for any platform. (It is hoped that these pages will migrate to somewhere more memorable soon, to `http://gloss.bham.ac.uk` perhaps. If so, the first page will remain operational as long as possible, and

will contain links to the 'real' home page.)

- `http://www.w3.org/Math`, the W3C's MathML pages. In particular the page `http://www.w3.org/TR/MathML2/` contains the specification for MathML 2.0. Chapters 1–3 make very good reading for the details of presentation MathML, which is what I have used. Other chapters cover content MathML, which may also be of interest.
- `http://www.w3.org/2003/entities/`, information on the standard names for characters used in XHTML and MathML (and GLOSS). Useful character tables are provided.
- `http://www.unicode.org`, the unicode consortium. With many more character tables, and lists of characters that can be referred to by number rather than name, provided you have the appropriate fonts on your system of course!
- `http://www.w3.org/2003/01/xhtml-mimetype/content-negotiation`, some information and advice from the W3C on content negotiation.
- `http://www.mozilla.org/projects/mathml/`, the Mozilla MathML page.
- `http://hutchinson.belmont.ma.us/tth/mml/`, TtM, a TeX to MathML translator.

Richard Kaye
School of Mathematics
University of Birmingham