H. Hagen
PRAGMA ADE

MathML

It is a well known fact that TeX can do a pretty good job on typesetting math. This is one reason why many scientific articles, papers and books are typeset using TeX. However, in these days of triumphing angle brackets, coding in TeX looks more and more out of place.

From the point of view of an author, coding in TeX is quite natural, given that some time is spent on reading the manuals. This is because not only the natural flow of the definition suits the way mathematicians think, but also because the author has quite some control over the way his thoughts end up on paper. It will be no surprise that switching to a more restricted way of coding, which also demands more keystrokes, is not on forehand considered to be better.

There are however circumstances that one wants to share formulas (or formula like specifications) between several applications, one of which is a typesetting engine. In that case, a bit more work now, later saves you some headaches due to keeping the different source documents in sync.

As soon as coding math in angle brackets is discussed, those in favour stress that coding can be eased by using appropriate editors. Here we encounter a dilemma. For optimal usage, one should code in terms of content, that is, the principles that are expressed in a formula. Editors are not that strong in this area, and if they would be, editing would be not that much different from traditionally editing formulas: just keying in ideas using code that at first sight looks obscure. A more graphical oriented editor can help authors to compose formulas, but the under laying coding will mainly be in terms of placing glyphs and boxes, and as a result the code will hardly be usable in other applications.

So either we code in terms of concepts, which permits sharing code among applications, and poses strong limitations on the influence of authors on the visual appearance. Or we use an interactive editor to fine tune the appearance of a formula and take for granted that reuse will be minimal or suboptimal.

In this presentation I will discuss the mathematical language MATHML in the perspective of typography as well as demonstrate how formulas coded in MATHML can be transformed into readable output.

Welcome to MathML

how to present your math content

Hans Hagen

Introduction

What is TEX

TEX is a general purpose typographic (macro) programming language.

TEX has a pretty good reputation for typesetting (complicated) math formulas.

TEX uses \'s and $'s and other funny characters to signal special actions.

TEX is almost 20 years old, and that's makes its users a weird species.

Although TEX can be used to make beautiful documents, it is (no longer) a first choice for typesetting jobs.

Many TEX documents sources look awful.

What is XML

XML is a language for coding all kind of documents.

XML looks like HTML and thereby it is pretty hip and popular.

XML is based on SGML and thereby also quite old.

XML also uses funny characters, like <, >, and &, but can look quite structured (apart from other funny characters).

Thanks to some good public relations, XML is now seen as the ultimate solution for coding documents.

Math in TEX

In TEX you enter math in a rather natural flow.

TEX permits you to optimize the visual appearance of a formula.

It's non-trivial to make a document consistent, especially when more authors are involved.

For simple formulas, coding in TEX is fast.

Math in XML

Coding math in XML is more verbose than in TEX, and is called MathML.

You can code in either presentational, or in content MathML.

In presentational markup, you compose a formula of its base typographic components.

In content markup, you define a formula in terms of what it means (represents).

Coding in presentational markup is not so much different from coding in TEX, although it's more strict and verbose.

Coding in content markup is less flexible, but more promising from the point of view of reusing information and consistent typography.

Exploration

Presentational Markup

You can summarize presentational markup as: what you key is what you get.

$x = 1$

```
<math>
  <mrow>
    <mi>x</mi> <mo>=</mo> <mn>1</mn>
  </mrow>
</math>
```

$x \leq 1$

```
<math>
  <mrow>
    <mi>x</mi> <mo>&le;</mo> <mn>1</mn>
  </mrow>
</math>
```

$\sin x^2$

```
<math>
  <mrow>
    <mi>sin</mi> <mo>&ApplyFunction;</mo>
    <msup> <mi>x</mi> <mn>2</mn> </msup>
  </mrow>
</math>
```

$(\sin x)^2$

```
<math>
  <msup>
    <mfenced> <mi>sin</mi> <mi>x</mi> </mfenced>
    <mn>2</mn>
  </msup>
</math>
```

Content Markup

You can summarize content markup as: you get typeset what you think.

$x = 1$

```
<math>
  <apply> <eq/>
    <ci>x</ci> <cn>1</cn>
  </apply>
</math>
```

$x \leq 1$

```
<math>
  <apply> <leq/>
    <ci>x</ci> <cn>1</cn>
  </apply>
</math>
```

$\sin(x^2)$

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

$\sin^2 x$

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

Mixed Markup

Occasionally (or for some frequently) MathML is not rich enough. In that case you add TEX code to your formula.

$x_i = 5$

```
<math>
  <semantics>
    <apply> <eq/>
      <ci> x </ci> <cn> 5 </cn>
    </apply>
    <annotation encoding="TeX">
      x_i = 5
    </annotation>
  </semantics>
</math>
```

As an alternative, you can (mildly) enhance a formula marked up in content MathML with presentational elements.

$x_1 = 5$

```
<math>
  <apply> <eq/>
    <ci> <msub> <mi>x</mi> <mn>1</mn> </msub> </ci>
    <cn> 5 </cn>
  </apply>
</math>
```

Processing Instruction

Given that the formulas are coded consistently, you can influence the layout by providing local or global processing instructions.

$\log_7 x$

```
<math>
  <apply> <log/>
    <logbase> <ci>7</ci> </logbase> <ci>x</ci>
  </apply>
</math>
```

${}^7\log x$

```
<math>
  <?context-mathml-directive log location left?>
  <apply> <log/>
    <logbase> <ci>7</ci> </logbase> <ci>x</ci>
  </apply>
</math>
```

MathML in ConTEXt

In ConTEXt, XML support is build into the kernel.

MathML support is supported by core `xtag` filters, to be loaded at runtime.

You can embed MathML in normal ConTEXt documents:

```
\startXMLdata <math> ... </math> \stopXMLdata
\XMLdata {<math> ... </math>}
```

Such mixed documents can be converted to pure XML quite easily, which provides a nice migration path.

There will be much more layout options, as well as support for units, chemistry, and complex formula building.

There is a MathML manual, an example suite, and a experimentation site. And there will be more.

Some observations

Presentational MathML is not that useful.

Content MathML is not rich enough.

For inline math we need something different.

For presentational MathML we need editors with restrictions.

For content MathML we need conceptual editors.

So … we're not yet there.