Limitations in Content MathML

Kevin Kofler, Peter Schodl, Arnold Neumaier
Faculty of Mathematics
University of Vienna, Austria
Nordbergstr. 15, 1090 Wien, Austria
kevin.kofler@chello.at,
{peter.schodl,arnold.neumaier}@univie.ac.at

July 26, 2009

Support by the Austrian Science Fund (FWF) under contract number P20631 is gratefully acknowledged.

Abstract

This draft technical report is part of the FMathL research project at the University of Vienna, Austria, headed by Prof. Arnold Neumaier.

While working on an internal representation for formulas, Peter Schodl collected some example formulas which reflect common usage in mathematical texts and should thus be representable by our internal representation. Kevin Kofler tried to represent these formulas in Content MathML. This technical report summarizes the issues and limitations with Content MathML we encountered during this effort, leaving us to conclude that Content MathML in its current state is not an adequate representation for our goals.

We encountered the following issues and limitations:

- We consider the syntax of operations in Content MathML to be less than ideal: all the operations are bounded by `<apply>`, e.g.:

  ```
  <apply>
    <plus/>
    <cn> 2 </cn>
    <cn> 3 </cn>
  </apply>
  ```

  It would be much more natural to write:

  ```
  <plus>
    <cn> 2 </cn>
    <cn> 3 </cn>
  </plus>
  ```

  While we understand the advantages of a generic notation for all functions, including basic operators, and while `<apply>` can be interpreted as mapping to `(` and `</apply>` to `)`, we would prefer the second notation, which matches the real-world notation more closely.

- There are no row vectors, only column vectors and matrices, or lists (which should not be used in vector computations). One has to transpose a column vector to get a row (which is the exact opposite of the common notation in Numerical Analysis, where we write all vectors as rows and add a $^\mathsf{T}$ sign for column vectors), or resort to single-row matrices.

  An additional issue is that the current draft (20090604) of MathML 3 appears to contradict itself on whether a vector is a row or a column! The following sentence was added to section 4.4.9.1:
  Where orientation is important, such as for pre or post multiplication by a matrix a vector is treated as a row vector and its transpose is treated a column vector.
  whereas the next sentence, which was already present in MathML 2, says:
  For purposes of interaction with matrices and matrix multiplication, vectors are regarded as equivalent to a matrix consisting of a single column, and the transpose of a vector behaves the same as a matrix consisting of a single row.
  We have reported this inconsistency to the MathML working group. [1]

- The notation for quantification is less than satisfactory: to avoid running into paradoxes, we would like all quantifications to inherently be over a set, e.g. $\forall x \in X$. Content MathML does the exact opposite: there is no direct way to express $\forall x \in X$, instead you have to write something to the effect of $\forall x$ where $x \in X$. When quantifying over several variables, this becomes: $\forall x, z$ where $x \in X \wedge z \in X$:

```
<apply>
  <forall/>
  <bvar>
    <ci> x </ci>
  </bvar>
  <bvar>
    <ci> z </ci>
  </bvar>
  <condition>
    <apply>
      <and/>
      <apply>
        <in/>
        <ci> x </ci>
        <ci type="set"> X </ci>
      </apply>
      <apply>
        <in/>
        <ci> z </ci>
        <ci type="set"> X </ci>
      </apply>
    </apply>
  </condition>
```

[1] Update (Feb 23, 2010): This has been fixed in the 20091215 draft.

```
  ...
</apply>
```

The Content MathML specification says it is up to the renderer to render this as $\forall x, z \in X$.

Note that we have encountered a similar limitation in the subset of natural language interpreted by the NAPROCHE project.

- There appears to be no notation for block matrices such as

$$\begin{pmatrix} (\lambda - x)I & * \\ 0 & * \end{pmatrix}.$$

While it is possible to abuse the notation for regular matrices, the result is not valid Content MathML. A related issue is that there are no predefined constants for identity (I) or zero (0) matrices.

- There is no way to represent something like

$$a \overset{\text{assmpt.}}{=} b$$

in pure Content MathML. Content MathML specifies a tag called `<semantics>` which allows to specify equivalent Presentation MathML (which can represent the above, at least graphically) or unspecified arbitrary data, but the semantics of such arbitrary data are unspecified and completely up to the software reading that file (and Presentation MathML in particular is not designed to carry semantics, Content MathML is). A MathML renderer is allowed to render Presentation MathML annotations and other annotations it understands, but it is not required to, even if the annotation is in Presentation MathML!

- There is no notation for an ellipsis as in

$$\begin{pmatrix} \sigma_1 & & & \\ & \ddots & & \\ & & \sigma_r & \\ & & & 0 \end{pmatrix}.$$

- There is no straightforward notation for integer ranges such as $i = 1, \ldots, n$. They have to be given as

```
<bvar>
  <ci type="integer"> i </ci>
</bvar>
<lowbound>
  <cn type="integer"> 1 </cn>
</lowbound>
<upbound>
  <ci type="integer"> n </ci>
</upbound>
```

and the most obvious way to specify the range itself appears to be to specify the above in a `<set>` tag.

- There is no direct representation for restrictions such as:

$$\sum_{k=1}^{n} A_{ik} = b_i \qquad (i = 1, \ldots, n)$$

  These need to be rewritten as $\forall$ quantifiers, e.g.:

$$\forall i = 1, \ldots, n : \sum_{k=1}^{n} A_{ik} = b_i$$

  (where the integer range has to be represented as discussed above).

- There is no predefined function, and thus no default rendering, for conditional probabilities such as $\Pr(i|k)$.

- There are no predefined norms (other than the absolute value), and in particular no subscripted norms such as $\|x\|_1$, $\|x\|_2$, $\|x\|_\infty$, $\|A\|_F$ (the Frobenius norm) etc., and thus no default rendering for such norms.

- There is no assignment operator :=. There is the `<declare>` tag, but its purpose is just to internally define something, not to be rendered as an assignment, and thus it is normally not rendered.

- The `<sum>` tag appears not to allow multiple sums such as $\sum_{\substack{i=1,\ldots,m \\ k=1,\ldots,n}}$, they have to be separated into nested sums, e.g. $\sum_{i=1}^{m}\sum_{k=1}^{n}$.

- The `<selector/>` operation is limited to 2 indices, i.e. triple or higher indices, e.g. ${}^{2}\Phi^{ij}$, cannot be represented. There are also no data types for tensors or multidimensional arrays of dimensions higher than 2.

- There is no notation other than lambdas for evaluating an expression at one:

$$f(x)|_{x=a}$$

  or two:

$$\frac{t^2}{2}\Big|_0^x$$

  places. This can be approximated using lambda expressions, e.g. (for the second example, using `<declare>` to define the `eval` function):

$$\texttt{eval} := \lambda(f, x, y, f(y) - f(x));$$

$$\texttt{eval}(\lambda(t, t^2), 0, x),$$

  but this requires 2 pages of XML and the resulting default rendering will be very far from the intended one.

- There is no representation for comma derivatives and covariant derivatives, i.e. for the , and ; operators in e.g.:

$$A^i{}_{;k} = A^i{}_{,k} + \Gamma^i{}_{ka} A^a.$$

  This is a symptom of a general problem: specialized concepts can be represented only poorly or not at all in Content MathML, its extensibility is limited.

- Presentation information which mathematicians will want to preserve gets lost unless Presentation MathML annotations are systematically provided. For example, Content MathML cannot distinguish between an index above or below: assuming they represent indices of a vector (and not powers, derivatives or a single variable), x_k, x^k and $x^{(k)}$ will all be converted to the same representation. When converting it back, a default rendering will be used. This is by design, as MathML considers this distinction a presentation information and thus inappropriate in a content language. However, in the real world, mathematicians will expect their internal representation to preserve this kind of notation.

  While it is possible to provide both Content MathML and Presentation MathML for a given expression (using e.g. the `<semantics>` tag) and while this can also work around some of the above limitations (e.g. the lack of builtin operators for some concepts: it is possible to define custom functions for them and specify the rendering using Presentation MathML), this is less than satisfactory for several reasons: it generates redundancy (as some content information will be repeated in the presentation), it requires specifying the presentation each time, it is hard to automatically match up a given presentation element to its corresponding content element, and the renderer is not required to actually use the given Presentation MathML for rendering (it is also allowed to render based on the Content MathML only).

This leads us to the conclusion that Content MathML is not an adequate representation for our goals in its current state and would need significant changes to fulfill them. Therefore, we will be using our own internal representation. While it will be possible to export to MathML, such export functionality will be subject to the above limitations.