>Call for Papers

>General Information

>Registration

>Accommodations

>Travel

>Tutorials

>Presentations

>Schedule

Content-Faithful Transformations for MathML

Sandy Huerter, Igor Rodionov, and Stephen Watt
The University of Western Ontario

Abstract

We present an implementation of transformations from MathML content to presentation form. These transformations preserve the MathML content in a recoverable fashion in the output, and are thus "content-faithful." Such transformations are critical for the exchange of MathML-encoded objects between complementary MathML applications that favor different MathML aspects (content vs. presentation). We show how our method of content-faithful transformation may be used to preserve semantics in extended MathML, that is MathML with additional application-defined elements.

Introduction

Mathematical Markup Language (or MathML, [1]) addresses the notational preferences and symbolic ambiguities of mathematical communication by providing two encoding schemes for mathematical objects --- semantic (or content) encoding, and notational (or presentation) encoding --- and defines a mechanism for binding one to the other.

MathML applications may be divided according to the type of encoding they favor; for example, presentation markup is not a priority in some computer algebra systems (e.g. [9]), while content markup is not featured in some equation editors (e.g. [10])). Transformations between the content and presentation encodings are therefore necessary in order for encoded math objects to be successfully shared between different types of MathML applications.

It is easier in practice to generate presentation MathML for given content MathML. The reverse process is not as algorithmic as it requires the use of heuristics (generally, there is a one-to-many correspondence between content and presentation encoding for a given object). Thus, a MathML-encoded object's potential for reuse depends on the amount of semantics it carries with it. It is impractical to communicate pure presentation MathML to an application requiring content MathML. Therefore, transformations that do not preserve MathML content are undesirable, since the range of use of the resulting object is reduced.

For example, the mathematical expressions that we put into scientific papers are very often examples of work that we have done in a particular content-oriented application. To communicate (and not just publish) such work over the Web, we not only want to have browsers render the expressions embedded in our Web document (which requires presentation markup), but also to allow colleagues the convenience of selecting such expressions from the online document and pasting them into their own preferred content-favoring applications for evaluation. Care must be taken, then, to conserve the original content markup when we transform our work into a format suitable for publication on the Web (e.g. into a notation that is more widely accepted by the target audience than the one that our content-favoring application is configured to support).

In the following sections we describe some techniques for implementing the "content-faithful" MathML transformations defined in [1]. We review three methods

for binding MathML semantics to MathML presentation, and describe the XSLT implementation of a content-faithful MathML transform that can preserve semantics in any of those three ways. Next, we describe how to conserve semantic content of extended MathML, that is MathML with elements from other namespaces. Finally, we discuss related work and draw some conclusions.

Parallel Markup

As described in [1], a "content-faithful" MathML transform is one that does not discard the original content encoding. This is made possible by the `semantics' construct of MathML, which allows content code to be carried along with its corresponding presentation. More generally, the `semantics' construct provides for a very flexible way to establish a correspondence among several alternative encodings for the same object, such as MathML-Content, MathML-Presentation, OpenMath, TeX, etc. An example of the `semantics' element is shown below.

```
<semantics>
   <!-- Some MathML here, e.g. MathML content markup -->
 <annotation-xml encoding="MathML-Presentation">
   <!-- Corresponding MathML presentation markup -->
 </annotation-xml>
 <annotation-xml encoding="OpenMath">
   <!-- Corresponding OpenMath markup -->
 </annotation-xml>
 <annotation encoding="TeX">
   <!-- Corresponding TeX markup -->
 </annotation>
</semantics>
```

We now review three ways that a MathML content tree and its corresponding presentation tree can be bound to each other (referred to as "parallel markup") by a content-faithful MathML transform.

Top-level Parallel Markup

In its simplest use, the `<semantics>` element appears at the top level only (i.e. no further nested 'semantics' elements), and it pairs a MathML presentation tree with the corresponding MathML content tree. The simplest example is to describe a single variable *x*:

```
<semantics>
  <mi> x </mi>
  <annotation-xml encoding="MathML">
    <ci> x </ci>
  </annotation-xml>
</semantics>
```

This style of parallel markup is most appropriate for situations where applications do not require access to subexpressions of the encoded expression. While individual subexpressions can still be selected, cut and pasted, their semantic meaning will not be readily available. The merit of this approach is that it preserves the original content encoding, and keeps the size of the output file within reasonable limits.

Fine-grained Parallel Markup

Some applications, such as equation editors, take a more hierarchical view of math objects in that they allow selection and manipulation of the object's parts, or subexpressions. In this case, a content-faithful transform needs to decompose the

generated presentation markup into units corresponding to the desired subexpression-decomposition of the content markup, and form the presentation-content pairs accordingly. This is can be done by the recursive use of the `<semantics>` element. Adding to the above example, the fine-grained parallel markup that would be generated for the content encoding of the expression "*x* divided by 13" is:

```
<semantics>
 <mrow>
  <semantics>
   <mi> x </mi>
   <annotation-xml encoding="MathML">
    <ci> x </ci>
   </annotation-xml>
  </semantics>
  <mo> / </mo>
  <semantics>
   <mn> 13 </mn>
   <annotation-xml encoding="MathML">
    <cn> 13 </cn>
   </annotation-xml>
  </semantics>
 </mrow>
 <annotation-xml encoding="MathML">
  <apply>
   </divide>
   <ci> x </ci>
   <cn> 13 </cn>
  </apply>
 </annotation-xml>
</semantics>
```

While this approach is very flexible (works for non-MathML annotations and multiple annotations), it is not very efficient as it results in a quadratic increase in document size.

Parallel Markup via Cross-References

A more efficient approach (in terms of required space) is for the content-faithful transform to generate two parallel trees --- the original content-encoded tree, and the new presentation-encoded tree --- and bind them together by means of the 'semantics' element (just like in the first approach so far), and then to generate references from the generated presentation markup to the original content markup. Our example now gives:

```
<semantics>
 <mrow xref="abc1">
  <mi xref="abc3"> x </mi>
  <mo xref="abc2"> / </mo>
  <mn xref="abc4"> 13 </mn>
 </mrow>
 <annotation-xml encoding="MathML-Content">
  <apply id="abc1">
   <divide id="abc2"/>
   <ci id="abc3"> x </ci>
   <cn id="abc4"> 13 </cn>
  </apply>
 </annotation-xml>
</semantics>
```

The only drawback of this approach is that every node in the original tree must have a unique id, so that a reference can be made to it. In practice, this means that normally two passes will be needed for this approach to work (unless ids are already present in the content markup): ids must be generated for every node during the first pass (which can be easily done with a straightforward XSLT stylesheet), and during the second pass the output will be generated with links to the augmented input tree. Refer to [6] for more information.

Implementing Content-Faithful Transforms in XSLT

Just as Cascading Stylesheets (CSS) are used to transform HTML documents, XSLT (Extensible Stylesheet Langueage Transformations) stylesheets are used with XML documents. In this section, we describe the implementation of content-faithful MathML transformations in XSLT.

The XSLT stylesheet we have implemented works in several `semantics-handling' modes, some of which correspond to the three parallel markup techniques described in the previous section. The desired mode can be chosen by the application using the stylesheet.

The first mode simply generates pure presentation MathML markup corresponding to the input content markup, so no content markup whatsoever is included into the output (it is completely stripped, even if it shows up as the 'annotation-xml' in the 'semantics' element). This mode can be useful for output transformations. The second (default) mode is similar to the first (does not place input content markup in the output), but preserves all the 'annotation-xml' markup from the input. The remaining three modes of the stylesheet are content-faithful and correspond to the three parallel markup styles described in the previous section.

The stylesheet is modular in structure and takes one parameter, `SEM_SW` indicating the mode it is to use. The stylesheet consists of a number of templates, the first of which is the one that matches the topmost element of the input content-MathML fragment. Once it is matched, it uses the value of the `SEM_SW` to decide whether to add a top level `<semantics>` element to the result tree. Next, the `semantics-handling' template is invoked, and is applied recursively. At each level in the source tree, this template first uses `SEM_SW` to decide whether to strip off, pass, or add semantics at that level, and then it applies the element-specific template to the node. A sketch of the semantics-handling template is given below. For brevity, we have omitted the use of some extra parameters which are used to correctly group and parenthesize operands in the generated MathML presentation markup.

```
<xsl:template match = "mml:*" mode = "semantics">
 <xsl:choose>
  <xsl:when test="$SEM_SW=$SEM_STRIP
            and self::mml:semantics">
   <xsl:apply-templates
     select="mml:annotation-xml[@encoding='MathML']"/>
  </xsl:when>
  <xsl:when test="($SEM_SW=$SEM_PASS or $SEM_SW=$SEM_TOP)
          and self::mml:semantics">
   <semantics>
    <xsl:apply-templates select="*[1]"/>
    <xsl:copy-of select="mml:annotation-xml"/>
   </semantics>
  </xsl:when>
  <xsl:when test="$SEM_SW=$SEM_ALL">
   <semantics>
```

```
      <xsl:choose>
       <xsl:when test="self::mml:semantics">
        <xsl:apply-templates select="*[1]"/>
        <xsl:copy-of select="mml:annotation-xml"/>
       </xsl:when>
       <xsl:otherwise>
        <xsl:apply-templates select="."/>
        <annotation-xml encoding="MathML">
         <xsl:copy-of select="."/>
        </annotation-xml>
       </xsl:otherwise>
      </xsl:choose>
     </semantics>
    </xsl:when>
    <xsl:when test="$SEM_SW=$SEM_XREF and @id">
     <xsl:choose>
      <xsl:when test="self::mml:semantics">
       <xsl:copy>
        <xsl:copy-of select="@*"/>
        <xsl:attribute name="xref">
         <xsl:value-of select="@id"/>
        </xsl:attribute>
        <xsl:copy-of select="*[1]"/>
        <xsl:copy-of select="mml:annotation-xml"/>
       </xsl:copy>
      </xsl:when>
      <xsl:otherwise>
       <xsl:apply-templates select="."/>
      </xsl:otherwise>
     </xsl:choose>
    </xsl:when>
    <xsl:otherwise>
     <xsl:choose>
      <xsl:when test="self::mml:semantics">
       <xsl:copy-of select="."/>
      </xsl:when>
      <xsl:otherwise>
       <xsl:apply-templates select="."/>
      </xsl:otherwise>
     </xsl:choose>
    </xsl:otherwise>
   </xsl:choose>
</xsl:template>
```

For every MathML content element, our stylesheet includes a template rule which, once matched, does the necessary processing for the current node, and then invokes the semantics-handling template on the children of the matched node. For example, the rule for the <transpose> element is shown below.

```
<xsl:template match = "mml:apply[mml:transpose[1]]">
 <msup>
  <xsl:if test="($SEM_SW=$SEM_XREF
            or $SEM_SW=$SEM_XREF_EXT) and @id">
   <xsl:attribute name="xref">
    <xsl:value-of select="@id"/>
   </xsl:attribute>
  </xsl:if>
  <xsl:apply-templates select="*[2]" mode="semantics"/>
  <mo>T</mo>
 </msup>
```

```
</xsl:template>
```

This template rule is typical of most of our rules for the MathML content elements: the presentation to be generated is first tagged with a cross-reference to the corresponding content node (if the semantics mode is using cross-references). Next, the necessary presentation markup directly corresponding to the current (matched) content node is added to the result tree, and then the children are processed (by going through the above-mentioned semantics-handling template first).

Not all transformations happen to be so straightforward. Some of them are ``non-local'' ([1]) in the sense that they require examination of the context of the node in question, in order to generate the correct presentation for it. For example, the second power of the greatest common divisor of *x* and *y* should be represented as $\gcd^2(x, y)$ rather than $\gcd(x, y)^2$. For this reason, the template below tests the context of the application of the *gcd* operator as shown here.

```
<xsl:template match = "mml:apply[mml:gcd[1]]">
 <mrow>
  <xsl:if test="$SEM_SW=$SEM_XREF and @id">
   <xsl:attribute name="xref">
    <xsl:value-of select="@id"/>
   </xsl:attribute>
  </xsl:if>
  <xsl:if test="not(parent::mml:apply[mml:power[1]])">
   <mo>gcd</mo>
  </xsl:if>
  <xsl:if test="parent::mml:apply[mml:power[1]]">
   <msup>
    <mo>gcd</mo>
     <xsl:apply-templates select = "../*[3]"
                          mode = "semantics"/>
    </msup>
  </xsl:if>
  <mfenced separators=",">
   <xsl:for-each select = "*[position()>1]">
    <xsl:apply-templates select = "." mode="semantics"/>
   </xsl:for-each>
  </mfenced>
 </mrow>
</xsl:template>
```

Another good example of non-locality would be any transformation involving any of the arithmetic operations, such as addition, subtraction, division, multiplication, modulo, etc. In the input markup, the shape of the tree unambiguously defines the order of evaluation of the subexpressions. In the output, their default precedences have to be overridden by means of use of parentheses where appropriate. An added challenge is that some operands can be negative, and in some situations they have to be parenthesized, while in other cases they do not. The difficult part is that sometimes these operands can be buried arbitrarily deeply in the tree, like the number - 13 in the following example: x + (-13) / a , and x - (-13 / a) . All these issues are recognized, and dealt with accordingly. For more examples of non-local MathML transformations, see [1].

Preserving Extended Content

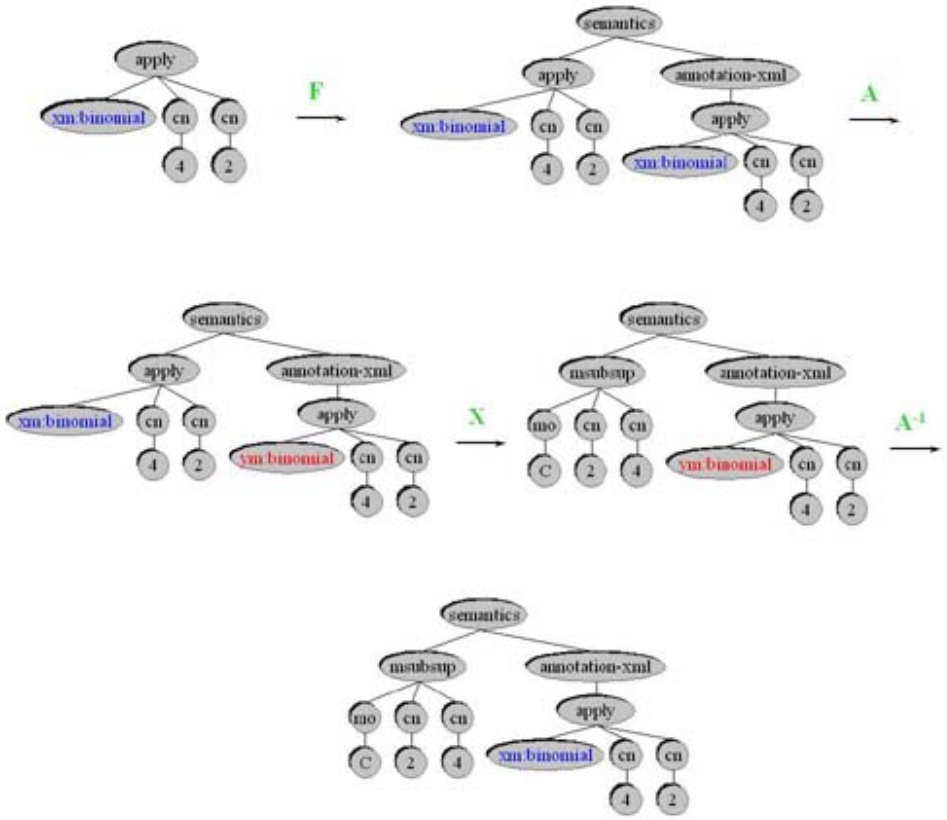
The existing set of MathML content tags does not attempt to cover our vast and ever-expanding body of mathematical discourse. In fact, it is only designed to cover mathematics up to and including the secondary school level. It is therefore

necessary to use external definitions (e.g. provided by some *definition server* such as OpenMath ([8]) in order to represent concepts not pre-defined in MathML. In this section, we explore content-faithful transformations of *extended MathML* content.

Consider a content encoding over an extended MathML set of symbols, for which presentation is to be generated in a ``content-faithful'' manner. Not only do we have to faithfully ``transform'' the standard MathML symbols, but the non-MathML symbols, as well. Assuming a framework such as that described in [7], in which transforms exist (or can be generated) to expand each extended symbol to a desired notation (just as standard MathML content elements, extended elements can be transformed into content, presentation, or a combination of both; for our purposes, we assume the extension transform produces notation.), we can combine our existing content-faithful transform together with the extension-transforms, to effect the transformation.

In general, we cannot assume that the extension templates are compatible (i.e. operate under the same content-faithful modes as our standard set) with our rules for standard MathML; this prevents the parallel application of the extended and standard rules. We note also that if the extensions are expanded first (by their corresponding templates), and the standard rules are then applied, then we lose the original semantics of the object, since the semantics of the object resides in the extended tag (i.e. we can't recover it from the presentation markup it expanded to). However, it is possible to slightly modify our standard transform so that it is able to handle extended MathML content. For example, any extension could be treated as a standard MathML element, by generating both a presentation tree and a content tree for it (depending on the desired semantics-handling mode). However, our transform would simply copy the extended content through, to temporarily hold a place in each tree. A simple intermediate transform could then be used, which relabels the extended elements in the content part of the parallel markup generated by the standard transform (say, with a different namespace prefix), so that they are ignored by the expansion transforms. Then the expansion transforms could be applied to the result, to expand the extended tags in the presentation tree. Finally, we can use the inverse of the relabelling transform in order to recover the original extended element down in the content tree, which denotes the intended semantics of the math object. This process ensures that we don't lose the extended content.

For example, if **xm** is the namespace denoting some MathML content extensions, our standard MathML transform is denoted by **F**, the notational transform for the extended elements is denoted by **X**, the relabeling transform and its inverse are denoted by **A** and **A'** respectively, and we are interested in preserving top-level semantics of our extended MathML, the process would be as shown below. The result is the transformation $\mathbf{X_A \circ F}$, where we define the conjugate of **X** with respect to **A** as $\mathbf{X_A = A^{-1} \circ X \circ A}$.



Related Work

The stylesheet that we implemented has been used in several projects, some of which are briefly described below.

Notation Selection Tool

As was mentioned earlier, transformation of content MathML to presentation is a relatively easy task (although involving a fair bit of programming effort - our stylesheet contains over three thousand lines of XSLT code). However, a given stylesheet will only generate one particular kind of presentation for given content markup. Because very often different notations are used in different geographical areas to represent the same mathematical notion (e.g. an operator, such as division), it may be desirable to use slightly different versions of a stylesheet depending on user's preferences. The only problem with this approach is that there would have to exist many different versions of such stylesheet (potentially, up to the number of combinations of different representations of all of the MathML operators).

Fortunately, there are solutions to this problem. The idea is to dynamically generate the stylesheet itself. In essence, a XSLT stylesheet is a collection of templates that specify the rules how a particular XML (in this case MathML) element should be transformed when it is matched in the input document. Therefore, once we have one (default) version of a stylesheet, we can easily generate a new version of it by simply substituting a particular template in it with the one that we need. One does not even need to understand XSLT to be able to do such substitutions, because there exist tools with user-friendly graphical front-ends allowing to compile a stylesheet according to the user's preferences ([5]). Yet another way to generate XSLT stylesheets using metastylesheets is described in [7].

The HELM Project An earlier version of the content-faithful stylesheet we developed and described in this paper plays an important role in the Helm project at the University of Bologna ([11]). The aim of the HELM project is to develop a suitable technology for the creation and maintenance of a virtual distributed hyper-textual library of structured mathematical knowledge implemented in XML.

Conclusions

We have described the motivation, advantages and implementation of content-faithful content-to-presentation transformations for ``native'' MathML content, and suggested a method for handling extended MathML content in a content-faithful manner. Such transformations are critical to the flow of MathML communication, since most nodes in the network of MathML applications are not fluent in both dialects of MathML.

References

[1] MathML spec: http://www.w3.org/TR/MathML2/

[2] XSLT spec: http://www.w3.org/TR/xslt

[3] XML spec: http://www.w3.org/XML

[4] XPath spec: http://www.w3.org/XPath

[5] "A Notation Selection Tool for MathML Stylesheets", Dicheng Liu, MSc Project, U Western Ontario 2001.

[6] "Tools for MathML", Igor Rodionov, MSc Thesis, U Western Ontario 2001.

[7] "Meta stylesheets for the conversion of mathematical documents in to multiple forms", Bill Naylor and Stephen Watt, Proc 2001 Conference on Mathematical Knowledge Management, Linz, Austria.

[8] "On the Relationship between OpenMath and MathML", .Bill Naylor and Stephen Watt, Proc 2001 Conference on Internet Accessible Mathematical Communication, London, Ontario

[9] REDUCE-MathML Interface, http://www.zib.de/Symbolik/reduce/moredocs/mathml.pdf

[10] Amaya Home Page: http://www.w3.org/Amaya/

[11] The HELM Project at the University of Bologna, http://www.cs.unibo.it/helm/