Limitations in OpenMath

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

April 27, 2010

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 OpenMath. This technical report summarizes the issues and limitations with OpenMath we encountered during this effort, leaving us to conclude that OpenMath in its current state is not an adequate representation for our goals.

When we evaluated[1] Content MathML in a similar way, we found the main issues to be lack of support for some common mathematical concepts which were used in our formulas. With OpenMath, the problems we found were of a somewhat different nature:

- OpenMath has basically no default lexicon. Instead, it relies on so-called "content dictionaries" to define all the functions used. While this sounds like a good thing as first, as it gives us the flexibility to represent all the concepts we need, in practice this is awful for interoperability, as symbols in namespaces defined by some application will in general not be recognized by other applications, making it of only limited use as an interchange format. Addition of arbitrary new functions not defined in the standard is also possible in other representations (e.g. Content MathML), but this will not lead to interoperability with other applications.

- A few content dictionaries are defined in the OpenMath standard (forming the closest thing to a default lexicon). However, those default content dictionaries are insufficient to encode all the content of our formulas. For example, there are no predefined operations for block matrices or for the := assignment operator.

[1]K. Kofler, P. Schodl, A. Neumaier: Limitations in Content MathML.
`http://www.mat.univie.ac.at/~neum/FMathL/mathml0.pdf`

- The default set of operations is split into many separate content dictionaries (over two dozen are needed to reach a functionality level comparable to Content MathML) and not all applications support all of them.

- Many of the official content dictionaries are still marked “experimental”, which means they can change incompatibly in the future. This hinders application support. The subset specified by the already stable content dictionaries is even more incomplete.

- The operations in the default content dictionaries tend to be underspecified. In particular, they are usually as general as possible, e.g. `arith1` just wants an Abelian semigroup. For items such as constants, this is an underspecification: zero can be the zero element of anything! Interestingly, there is a specific function for a zero matrix (with the dimensions as parameters), but that also does not specify what field the matrix is over. While it is possible to specify a type using an attribute, the type systems used with OpenMath are themselves underspecified, e.g. a type can be a variable (such as `Abelian_semigroup`) with unspecified meaning. (The meaning can only be understood by a human or knowledge system who already knows what e.g. an Abelian semigroup is.)

- The syntax is very verbose, even compared to Content MathML, and carries a high level of redundancy. The simple `+` symbol is rendered as `<plus/>` in Content MathML. In OpenMath, one also has to specify the content dictionary for each operation, resulting in `<OMS cd="arith1" name="plus"/>`. And similarly to Content MathML, all the function calls have to be wrapped with `<OMA>` (OpenMath Apply) and `</OMA>`. Having the arguments be child nodes of the function would be more natural. Another source of redundancy is that the default transitive relations (e.g. `=`, `<`, `>` etc.) are only binary, not *n*ary. While defining additional *n*ary versions would be possible, it would lose interoperability.

- The representation for symbol names is insufficient to encode commonly-used mathematical variable and function names. OpenMath expects identifier names to be strings of a subset of Unicode consisting of the usual ASCII identifier characters and a fairly arbitrary set of letterlike Unicode symbols. That set does not include characters such as the caret `^` (which would be useful to encode superscripts) or the mathematical $'$ (prime) character U+2032. (As a workaround, U+02B9 MODIFIER LETTER PRIME (`&#697;` in XML) could be used, which is found in the arbitrary allowed subset, but that is not the recommended character to use for the prime symbol.) This is also a problem when converting from Content MathML, as Content MathML allows arbitrary Presentation MathML for symbol names.

- OpenMath is only a pure content representation. It is not designed for rendering and OpenMath operations do not have any inherent default rendering (other than the basic function notation, which is inadequate for many operations, e.g. `plus(2,3)` is a poor way to write $2+3$). The ability to represent arbitrary operations by defining arbitrary content dictionaries makes rendering troublesome, especially as those content dictionaries have no standard way to specify a default rendering. There are a few approaches to rendering OpenMath, but neither is satisfying:

- It is possible to go through MathML, i.e. convert the OpenMath to Content MathML which is subsequently rendered to Presentation MathML. However, this is subject to Content MathML's limitations; in particular, it will work poorly (i.e. just use the default function representation) for any content dictionaries which do not map directly to native Content MathML operations.
- Some automated direct renderers exist, e.g. `http://www.maths.tcd.ie/~dwmalone/om2la.html`. But those renderers also only support fixed sets of content dictionaries, and adding more is often not trivial. E.g., in `oml2a`, adding a pretty rendering for a new operation is only possible by editing the Perl code directly.
- It would in principle be possible to define a rendering (e.g. in Presentation MathML) for all the operations in a content dictionary. The problem is that there is no standard way to do this.
- It is possible to attach annotations to OpenMath objects, so we could annotate all objects with a rendering (e.g. in Presentation MathML). However, this would result in a high redundancy, and the rendering would still have to come from somewhere.

This leads us to the conclusion that OpenMath 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 OpenMath, such export functionality will be subject to the above limitations; in particular, we have strong concerns about interoperability. The more interoperable the output should be, the fewer formulas we will be able to represent.