# OpenMath3 without conditions: A Proposal for a MathML3/OM3 Calculus Content Dictionary

Michael Kohlhase

September 6, 2008

**Abstract**

We propose a new way of encoding binding operators in OpenMath/MathML that alleviates the need to introduce `condition` elements into OpenMath3. We evaluate these ideas by providing a content dictionary `calculus3` that is more closely aligned with MathML2 representation intuitions as a replacement for the OpenMath standard CD `calculus1`.

## 1 Introduction

We are currently reworking the OpenMath content dictionaries from the "MathML group" in an attempt to align the OpenMath3 and MathML3 languages. One area of contention is the fact that MathML allows binding constructions where the bound variables are restricted by "qualifier elements", such as `domainofapplication`, `condition`, `uplimit`, `lowlimit`, `degree`, and `momentabout`.

Another bone of contention is that MathML often expresses functionals using binding operators over expressions with bound variables (and qualifiers), whereas OpenMath tends to apply the functionals themselves to functions represented with the help of the $\lambda$ operator. Probably the synchronized OpenMath3/MathML3 content dictionaries should support both styles, since they appeal to different communities of mathematicians. We propose a new content dictionary `calculus3`[1] that is more closely aligned with MathML2 representation intuitions as a replacement for the OpenMath standard CD `calculus1`. The new content dictionary can be found on the OpenMath3 development repository as `https://svn.openmath.org/OpenMath3/cd/MathML/calculu3.ocd`.

## 2 Derivatives

MathML interprets derivatives as operators on expressions in one bound variable and presents as paradigmatic examples:

[1] `calculus2` already exists as an experimental CD on `openmath.org`

| 1: $diff_x(x^2)$ | 2: $diff^2_x(x^5)$ |
|---|---|
| <apply><diff/><br><bvar><ci>x</ci></bvar><br><apply><br><power/><br><ci>x</ci><br><cn>2</cn><br></apply><br></apply> | <apply><diff/><br><bvar><br><ci>x</ci><br><degree><cn>2</cn></degree><br></bvar><br><apply><power/><ci>x</ci><cn>5</cn></apply><br></apply> |

but also allows differentiation over a function as in

```
<apply><eq/><apply><diff/><sin/></apply><cos/></apply>
```

In this, we use the `diff` element as a functional that is applied to the sin function. For OpenMath the functional view is primary: the content dictionary `calculus1` supplies a symbol `diff` that is a functional so that the latter expression can be directly represented as

```
<OMA><OMS cd="relation1" name="eq"/>
  <OMA><OMS cd="calculus1" name="diff"/><OMS cd="transc1" name="sin"/></OMA>
  <OMS cd="transc1" name="cos"/>
</OMA>
```

For the left hand expression in the table above, we would use the `lambda` symbol from the `fns1` CD and a special symbol `nthdiff` from the `calculus1` CD.

| 1: $diff_x(x^2)$ | 2: $diff^2_x(x^5)$ |
|---|---|
| <OMA><OMS cd="calculus1" name="diff"/><br><OMBIND><br><OMS cd="fns1" name="lambda"/><br><OMBVAR><OMV name="x"/></OMBVAR><br><OMA><br><OMS cd="arith1" name="power"/><br><OMV name="x"/><br><OMI>2</OMI><br></OMA><br></OMBIND><br></OMA> | <OMA><OMS cd="calculus1" name="nthdiff"/><br><OMI>2</OMI><br><OMBIND><br><OMS cd="fns1" name="lambda"/><br><OMBVAR><OMV name="x"/></OMBVAR><br><OMA><OMS cd="arith1" name="power"/><br><OMV name="x"/><br><OMI>5</OMI><br></OMA><br></OMBIND><br></OMA> |

While we lose the directly structural correspondence, this is quite natural. But for a partial derivative like $\frac{d^k}{dx^m dy^n} f(x,y)$ which can be expressed in MathML by

```
<apply>
  <partialdiff/>
    <bvar><ci>x</ci><degree><ci>m</ci></degree></bvar>
    <bvar><ci>y</ci><degree><ci>n</ci></degree></bvar>
    <degree><ci>k</ci></degree>
    <apply><ci type="function">f</ci><ci>x</ci><ci>y</ci></apply>
</apply>
```

we would obtain the following representation using a partial differentiation operator that takes a list of degrees and a total degree as an arguments.

```
<OMA>
  <OMS cd="calculus1" name="pdiffdegree"/>
  <OMA><OMS cd="list1" name="list"><OMV name="m"/><OMV name="m"/></OMA>
  <OMV name="k"/>
  <OMBIND>
    <OMS cd="fns1" name="lambda"/>
```

```
    <OMBVAR><OMV name="x"/><OMV name="y"/></OMBVAR>
    <OMA><OMV name="f"><OMV name="x"/><OMV name="y"/></OMA>
  </OMBIND>
</OMA>
```

Note that we are using a variant `pdiffdegree` of the `partialdiff` symbol that allows to specify the total degree as an extra argument. We propose to add this to the `calculus1` CD.

In the proposed `calculus2` CD, we would model the `diff` and `partialdiff` as binding operator constructors and thereby make use of the fact that OpenMath allows the first child of an `OMBIND` to be an `OMA`, not just an `OMS` as is predominantly used. This gives us a much better structural similarity in the

| | MathML2 | strict cMathML3 |
|---|---|---|
| 1 | `<apply><diff/>`<br>`  <bvar><ci>x</ci></bvar>`<br>`  <apply>`<br>`    <power/>`<br>`    <ci>x</ci>`<br>`    <cn>2</cn>`<br>`  </apply>`<br>`</apply>` | `<bind><csymbol cd="calculus2">diff</csymbol>`<br>`  <bvar><ci>x</ci></bvar>`<br>`  <apply>`<br>`    <csymbol cd="arith1">power</csymbol>`<br>`    <ci>x</ci>`<br>`    <cn>2</cn>`<br>`  </apply>`<br>`</apply>` |
| 2 | `<apply><diff/>`<br>`  <bvar>`<br>`    <ci>x</ci>`<br>`    <degree><cn>2</cn></degree>`<br>`  </bvar>`<br>`  <apply>`<br>`    <power/>`<br>`    <ci>x</ci>`<br>`    <cn>5</cn>`<br>`  </apply>`<br>`</apply>` | `<bind>`<br>`  <apply>`<br>`    <nthdiff/>`<br>`    <cn>2</cn>`<br>`  </apply>`<br>`  <bvar><ci>x</ci></bvar>`<br>`  <apply>`<br>`    <power/>`<br>`    <ci>x</ci>`<br>`    <cn>5</cn>`<br>`  </apply>`<br>`</apply>` |

We have used strict content MathML to highlight the correspondence for the partial differentiation example we obtain

```
<OMBIND>
  <OMA><OMS cd="calculus3" name="pdiffdegree"/>
    <OMV name="m"/><OMV name="m"/><OMV name="k"/>
  </OMA>
  <OMBVAR><OMV name="x"/><OMV name="y"/></OMBVAR>
  <OMA><OMV name="f"><OMV name="x"/><OMV name="y"/></OMA>
</OMBIND>
```

# 3 Integrals

For integrals, the situation is similar, MathML interprets derivatives as operators on expressions in one bound variable and presents as paradigmatic examples the following three expressions, which differ in which ways the bound variables are handled.

3: $\int_0^a f(x)dx$

```
<apply>
  <int/>
  <bvar><ci>x</ci></bvar>
  <lowlimit><cn>0</cn></lowlimit>
  <uplimit><ci>a</ci></uplimit>
  <apply><ci>f</ci>
    <ci>x</ci>
  </apply>
</apply>
```

4: $\int_{x \in D} f(x)dx$

```
<apply>
  <int/>
  <bvar><ci>x</ci></bvar>
  <condition>
    <apply><in/>
      <ci>x</ci>
      <ci>D</ci>
    </apply>
  </condition>
  <apply><ci>f</ci>
    <ci>x</ci>
  </apply>
</apply>
```

5: $\int_D f(x)dx$

```
<apply>
  <int/>
  <bvar><ci>x</ci></bvar>
  <domainofapplication>
    <ci>D</ci>
  </domainofapplication>
  <apply><ci>f</ci>
    <ci>x</ci>
  </apply>
</apply>
```

Example 3. uses the `lowlimit` `uplimit` qualifiers that specify an ordered range of integration by allowing the bound variable to range from 0 to $a$. Example 4. uses a general `condition` qualifier that allows to place restrictions on the bound variable (this is possible on any binding operator), and finally example 5. uses the `domainofapplication` qualifier element that restricts the bound variable to range over a set.

MathML2 also allows integration over a function as in

6: $\int_{[a,b]} \cos$

```
<apply>
  <int/>
  <interval><ci>a</ci><ci>b</ci></interval>
  <cos/>
</apply>
```

7: $\int \sin = \cos$

```
<apply>
  <eq/>
  <apply><int/><sin/></apply>
  <cos/>
</apply>
```

Examples 5. and 6. can be represented in OpenMath using the `defint` symbol from the `calculus1` CD using similar representational intuitions as above: we apply `defint` symbol that takes a set and a function as arguments to the range of integration provided and construct a function as a $\lambda$-term. Example 7. works analogously using the `int` symbol for indefinite integration from `calculus1`.

MathML2 claims that `uplimit` and `lowlimit` can be reduced to `domainofapplication` or `condition`, but the convention $\int_a^b f(x)dx = -\int_b^a f(x)dx$ shows that this is not directly possible via the claimed intuition that $\int_a^b$ can be represented by $\int_{[a,b]}$, since either the interval $[a, b]$ or $[b, a]$ is nonsensical, or we would be forced to come up with a general notion of "reversed interval" only to fix the integration convention. Thus we have to do something else for example 3. In particular, the `FMP` representation of the convention above in the `calculus1` CD is nonsensical and should be eliminated.

For example 4. we are also in trouble, as we have to construct a set from the `condition` in order to use `int` or `defint`. This would in principle be possible using the `suchthat` symbol from `set1`, but we need a base set for separation here. But this is not given in example 3. and taking the "universal set" is set-theoretically problematic.

In this situation we propose to take the representational distinctions in examples 3. - 6. seriously model the definite integrals as distinct binding constructor as we did for differentiation. Our new `calculus3` content dictionary supplies a symbol `defint` that takes two real numbers as arguments: the lower and upper limits of the range of integration, a symbol `defintset` that takes a set as an argument, and finally a symbol `defintcond` that takes an expression involving the bound variable as an argument.

With this, we directly get the following strict content MathML representations (we have abbreviated `calculus3` to `calc3` to save space):

3: $\int_0^a f(x)dx$

```
<bind>
  <apply>
    <csymbol cd="calc3">defint</csymbol>
    <cn>0</cn><ci>a</ci>
  </apply>
  <bvar><ci>x</ci></bvar>
  <apply><ci>f</ci><ci>x</ci></apply>
</bind>
```

4: $\int_{x \in D} f(x)dx$

```
<bind>
  <apply>
    <csymbol cd="calc3">defintcond</csymbol>
    <apply><in/>
      <ci>x</ci>
      <ci>D</ci>
    </apply>
  </apply>
  <bvar><ci>x</ci></bvar>
  <apply><ci>f</ci><ci>x</ci></apply>
</bind>
```

5: $\int_D f(x)dx$

```
<bind>
  <apply>
    <csymbol cd="calc3>defintset </csymbol>
␣␣␣␣<ci>D</ci>
␣␣</apply>
␣␣<bvar><ci>x</ci></bvar>
␣␣<apply><ci>f</ci><ci>x</ci></apply>
</bind>
```

6: $\int_{[a,b]} \cos$

```
<apply>
  <csymbol cd="calc1">defint</csymbol>
  <interval><ci>a</ci><ci>b</ci></interval>
  <cos/>
</apply>
```

Note that we also propose to extend the old `calculus1` content dictionary with a symbol `defintbounds` that takes two real numbers as arguments: the lower and upper limits of the range of integration. The symbol `defint` for definite integration over functions in `calculus1` already takes the role analogous to `defintset` as example 6 shows.

# 4 Conclusions

We propose a new content dictionary `calculus3` that is more closely aligned with MathML2 representation intuitions as a replacement for the OpenMath standard CD `calculus1`.

For differentiation and integration over expressions with bound variables we should use the `calculus3` symbols, and for differentiation and integration over functions we should use `calculus1` symbols. This is directly analogous to the situation between the content dictionaries `s_data1` and `s_dist1`, where the underspecified usage with data sets and random variables in MathML has been specified into two different content dictionaries in OpenMath.

Finally, we remark that the use of binding constructors like we use them in the `calculus3` content dictionary allows us to move the MathML `condition` elements into (suitably defined) binding constructors, so that the core OpenMath format need not be extended to achieve synchronization with MathML. In particular note that an analog to the new symbol `defintcond` that takes an expression involving the bound variable as an argument cannot be added to `calculus1`, since the bound variable in the condition cannot be accessed outside the $\lambda$ term that binds it.

The only thing that has be be changed/clarified in the OpenMath3 standard is that scope of the bound variable (and thus replacement in $\alpha$-renaming) extends to the binding operator. As we have to clarify alpha renaming for attributions anyway, this seems like the lesser evil in comparison with extending the OpenMath format with a condition element, in particular since existing OpenMath only uses symbols as binding operators.

# 5 Acknowledgements

This proposal has been greatly influenced by discussions with Florian Rabe in the context of the development of the OMDoc 1.6 notation definitions.