New CD Release

James H. Davenport

Department of Computer Science
University of Bath
Bath BA2 7AY England
J.H.Davenport@bath.ac.uk

Deliverable D02

Contents

1 Introduction

Since OpenMath was last reported on [6], there have been several changes to the OpenMath Content Dictionaries available at `http://www.openmath.org`. Some of these changes are due to the release of MathML version 2 [9], since the OpenMath Society is committed to the principle that everything expressible in the content part of MathML should be expressible in OpenMath. Others are due to deliberate efforts to add functionality to OpenMath. We describe the changes under these headings, and finally look towards the future of OpenMath CDs.

We briefly recall the purpose of CDs. An OpenMath Content Dictionary lists a number of OpenMath symbols, together with their definitions. Note that the full name of a symbol comprises both the name of the symbol and the name of the CD it is from, so that overloading of symbol names is permitted, and encouraged where appropriate. A good example of this is provided by the two symbols

```
<OMS name="mean" cd="s-data1"/>
```

and

```
<OMS name="mean" cd="s-dist1"/>.
```

Both correspond to the MathML symbol `<mean/>`, depending on whether the MathML symbol is applied to a collection of data (the first case) or to a symbolic distribution (the second case). For more information about Content Dictionaries, see [1, 4].

There is a natural tendency to think of OpenMath symbols as describing computations. While this would be the natural meaning of them in an OpenMath interface to a computer algebra system, OpenMath *per se* does not ascribe any such semantics. and a type-setter, or a database, would not treat the symbols the same way. What the CD *does* prescribe is the semantics of the underlying mathematics, e.g. precisely which branch cut is meant for the log function [3]. The author has tried not to use words implying any operational semantics, but has probably not been perfect.

2 MathML-induced Changes

One significant change from the point of view of OpenMath $\leftrightarrow$ MathML compatibility in MathML 2 is the deprecation of the MathML 1 symbols `<reln>` and `<fn>`. This makes it easier to translate OpenMath into MathML, since `<OMA>` now translates more uniformly into `<apply>`.

2.1 MathML following OpenMath

Many of the changes in the OpenMath CD set were the result of MathML 2 adopting symbols that were already in OpenMath. Therefore all that was needed to follow suit was to re-arrange some CDs and CD groups. We therefore only describe these symbols briefly.

Arithmetic The new symbols are: `<arg/>`, `<real/>`, `<imaginary/>`, `<lcm/>`, `<floor/>` and `<ceiling/>`.

Relations The new symbols in this category are: `<equivalent/>`, `<approx/>`[1] and `<factorof/>`.

Set Theory Here the new symbols are `<card/>`, corresponding to the OpenMath symbol

```
<OMS name="size" cd="set1"/>,
```

and `<cartesianproduct/>` (spelled with an "_" in OpenMath).

Elementary Functions Here MathML completed the set, borrowing OpenMath's `<arccot/>`, `<arcsec/>` and `<arccsc/>`, as well as the hyperbolic equivalents.

Set Symbols MathML borrowed several of OpenMath's symbols for the standard sets of mathematics, which MathML called `<integers/>`, `<reals/>`, `<rationals/>`, `<naturalnumbers/>`, `<complexes/>` and `<primes/>`. In OpenMath, they are all of the format

```
<OMS name="R" cd="setname1"/>.
```

It should be noted that there are more such constructors in the `setname2` CD (see section 3.2) and in the various polynomial CDs (see section 4).

Constants Here MathML 2 added several new symbols: `<exponentiale/>`, `<imaginaryi/>`, `<notanumber/>`, `<true/>`, `<false/>`, `<infinity/>` (with the semantics[2] "... represents the concept of infinity. Proper interpretation depends on context."), `<pi/>` and `<eulergamma/>`. In OpenMath, some of the names are simpler, as in

```
<OMS name="e" cd="nums1"/>
```

and similarly `i`, `NaN` and `gamma`. `true`, `false` and `emptyset` of course live in CDs other than `nums1` (`boolean1` and `set1`).

[1] Though it is not precisely clear what the semantics of `<approx/>` in MathML or OpenMath are.

[2] OpenMath should probably introduce CDs for real and complex analysis, with their own infinities with tighter semantics, in due course.

2.2 New function-related symbols

MathML 2 introducted the symbols `domain`, `codomain` and `image`. While these were not in OpenMath, it was simple to add them to the `fns1` CD (as `domain`, `range` and `image`). It also introduced the symbol `domainofapplication`, with an example of $\int_C f$ as

```
<apply>
  <int/>
  <domainofapplication>
    <ci> C </ci>
  </domainofapplication>
  <ci> f </ci>
</apply>
```

This particular example was already catered for in OpenMath, as in

```
<OMOBJ>
  <OMA>
    <OMS name="defint" cd="calculus1"/>
    <OMV name="C"/>
    <OMV name="f"/>
  </OMA>
</OMOBJ>
```

However, the symbol `domainofapplication` was also added to the `fns1` CD in case there were other uses in MathML that were not already catered for in OpenMath.

2.3 Piecewise definitions

MathML 2 introduced three additional symbols to encode piece-wise definitions of functions: `piecewise`, `piece` and `otherwise`. The example quoted in the MathML specification is

```
<apply>
  <eq/>
  <apply>
    <abs/>
    <ci> x </ci>
  </apply>
  <piecewise>
    <piece>
        <apply><minus/><ci> x </ci></apply>
        <apply><lt/><ci> x </ci> <cn> 0 </cn></apply>
    </piece>
```

```
        <piece>
            <cn> 0 </cn>
            <apply><eq/><ci> x </ci> <cn> 0 </cn></apply>
        </piece>
        <piece>
            <ci> x </ci>
            <apply><gt/><ci> x </ci> <cn> 0 </cn></apply>
        </piece>
      </piecewise>
    </apply>
```

These were encoded into OpenMath as corresponding elements of the new `piece1` CD. The real difficulty comes in giving these symbols a Small Type System [5] signature, since the specification in MathML is under-defined, and, for example, there is no way of discovering from the MathML what variable the `piecewise` object is a function of. It may be that new CDs such as `piece2` are required to give more precise definitions.

2.4 Vectors and Vector Calculus

MathML 2 introduced the symbols `divergence`, `grad`, `curl` and `laplacian`. These were easily introduced into the new `veccalc1` CD (with the OpenMath symbol being `Laplacian` to reflect the OpenMath naming rules).

Similarly, `vectorproduct`, `scalarproduct` and `outerproduct` were easily added to the existing `linalg1` CD.

2.5 Statistics

MathML 2 introduced the qualifier `momentabout`, to be used with the `moment` operator to specify that the moment is being taken about some point. MathML does not specify what the `moment` is taken about if this qualifier is omitted. In OpenMath, this is an explicit positional argument of the `moment` symbol (from either the `s-data1` or `s-dist1` CDs).

2.6 Bug fixes

From the beginning, OpenMath has had the symbol `diff` (from the `Calculus1` CD) to denote differentiation, so that $\frac{\mathrm{d}^2}{\mathrm{d}x^2}x^3$ can be expressed as

```
<OMA>
  <OMS name="diff" cd="calculus1"/>
  <OMA>
    <OMS name="diff" cd="calculus1"/>
```

```
    <OMBIND>
      <OMBVAR>
        <OMV name="x"/>
      </OMBVAR>
      <OMA>
        <OMS name="power" cd="arith1"/>
        <OMV name="x"/>
        <OMI> 3 </OMI>
    </OMBIND>
  </OMA>
</OMA>
```

However, there was no way to express in OpenMath constructs such as $\frac{\mathrm{d}^n}{\mathrm{d}x^n}$. The network is therefore proposing to the OpenMath Society that the `calculus1` CD be extended by an `nthdiff` symbol, so that the above example could also be expressed as follows.

```
<OMA>
  <OMS name="nthdiff" cd="calculus1"/>
  <OMI> 2 </OMI>
  <OMBIND>
    <OMBVAR>
      <OMV name="x"/>
    </OMBVAR>
    <OMA>
      <OMS name="power" cd="arith1"/>
      <OMV name="x"/>
      <OMI> 3 </OMI>
  </OMBIND>
</OMA>
```

3 Extensions to the MathML CDs

Various things are not expressible in MathML, but would be nice to have for completeness. These are listed in various groups below.

3.1 Arithmetic and Functional Operations

The `arith2` CD contains two symbols: `inverse` intended to represent the additive or multiplicative inverse of an element, and `times`, an explicitly commutative version of the `times` symbol in the `arith1` CD.

The `fns2` CD contains three symbols.

`apply_to_list` which represents the application of an n-ary function to all the elements of a list.

`kernel` which represents the usual algebraic object.

`right_compose` is defined with the following semantics[3].

```
<OMA>
  <OMS cd="relation1" name="eq"/>
  <OMA>
    <OMA>
      <OMS cd="fns2" name="right_compose"/>
      <OMV name="f"/>
      <OMV name="g"/>
    </OMA>
    <OMV name="x"/>
  </OMA>
  <OMA>
    <OMV name="g"/>
    <OMA>
      <OMV name="f"/>
      <OMV name="x"/>
    </OMA>
  </OMA>
</OMA>
```

`left_compose` is in the MathML-compatible `fns1` CD.

The `list2` CD contains three symbols whose meanings are familiar to any LISP programmer: `cons`, `first` and `rest`. Under pressure from OpenMath users[4], the author is proposing the addition of `nil`, `append` and `reverse`.

3.2 Set Names

The `setname2` CD contains several other names of algebraic sets that were not present in MathML 2. These are: `A` (the algebraic numbers), `Boolean`, `GFp`, `GFpn`, `H` (the Hamiltonian, or hyper-complex, numbers), `QuotientField` (which takes an integral domain as argument) and `Zm`.

[3] `right_compose` is logically redundant, in view of the existence of `left_compose`, but suits certain fields of mathematics. One recalls that Openmath encodings should be "reasonably natural".

[4] Notably Paul Libbrecht (`paul@ags.uni-sb.de`).

3.3 Linear Algebra

MathML, and OpenMath in the corresponding `linalg2` CD, define matrices as built up from rows. The `linalg3` CD defines a column-oriented view of matrices, via the three symbols `matrix`, `matrixcolumn` and `vector`.

The `linalg4` CD contains some additional linear algebra symbols representing abstract concepts: `characteristic_eqn`, `columncount`, `eigenvalue` (this takes two arguments: the first should be the matrix, the second should be an index to specify the eigenvalue — the ordering imposed on the eigenvalues is first on the modulus of the value, and second on the argument of the value), `eigenvector` (similar to `eigenvalue`), `rank`, `rowcount` and `size`.

The `linalg5` CD contains various symbols for defining matrices of special shapes. They are: `anti-Hermitian`, `banded`, `constant`, `diagonal_matrix`, `Hermitian`, `identity`, `lower-Hessenberg`, `lower-triangular`, `scalar`, `skew-symmetric`, `symmetric`, `tridiagonal`, `upper-Hessenberg`, `upper-triangular` and `zero`.

4 Polynomials

Various CDs had been written during the previous OpenMath project to support an explicit view of polynomials in specific algebraic structures. The `polysts` CD defines a symbol `polynomial_ring`, which is the type of all polynomial rings. Specific polynomial rings are constructed via constructors from the various specific polynomial CDs (`polyr`, `polyd` etc.). The author held a review of these with John Abbott (from the algebra system CoCoA) and Hans Schonemann (SINGULAR) in October 2001. This led to various improvements in the CD (marked **(new)**), and the network will now propose the adoption of these CDs to the OpenMath Society.

4.1 The `poly` CD

The `poly` CD supports generic views of polynomials. Most operators take arguments in any polynomial ring, and return results in the same polynomial ring. The symbols in it are listed below.

`convert` This takes a polynomial in one polynomial ring, and the specification of a second polynomial ring (i.e. an object of type `polynomial_ring`), and expresses the polynomial represented in that second ring.

`degree` The total degree function.

`degree_wrt` The degree with respect to a specific variable (the second argument to the symbol).

`expand` This symbol represents the conversion of a `factored` or `squarefreed` form into an expanded polynomial over the same ring, so that, for example, factored(recursive) → recursive.

`factor` Unlike the next symbol, this is a call for a factorisation. The result (if this symbol is being used computationally) should be an expression built with `factored`.

`factored` The constructor for a factorization. Its arguments are formal powers (see the `power` operator below), where the polynomials are supposed to be irreducible (except possibly for a content from the ground ring) and relatively prime. Note that `factored` is not a call to factorise something, rather a statement that we know a (complete) factorisation.

`partially_factored` The constructor for a factorization. Its arguments are formal powers (see the `power` operator below), as in `factored`, but we do not assume that the polynomials are irreducible or relatively prime.

`gcd` This is an n-ary symbol, representing the greatest common divisor of its polynomial arguments.

`lcm` This is an n-ary symbol, representing the least common multiple of its polynomial arguments.

`power` Takes a polynomial and a (non-negative) integer and produces a formal power. Although OpenMath does not specify operational semantics, the idea here is that these powers are not evaluated. We note that the `power` from `arith1` would suggest the expanded form. Expressions built with `power` are the children of `factored` and `squarefreed`.

`discriminant` (**new**). This takes a polynomial and a variable, and represents the discriminant of the polynomial with respect to that variable.

`resultant` This takes two polynomials and a variable as arguments, and represents the resultant of the two polynomials with respect to that variable.

`squarefree` Unlike the next symbol, this is a call for a square free decomposition. The result should be an expression built with `squarefreed`.

`squarefreed` As for `factored` above.

`leading_coefficient` (**new**). This takes a polynomial and a variable, and represents the leading coefficient of the polynomial with respect to that variable, which is therefore a polynomial in all the other variables.

`coefficient` (**new**). This takes a polynomial, a list of variables and a corresponding list of exponents, and returns the coefficient of the monomial corresponding to those powers of those variables. This would be a polynomial in the remaining variables.

4.2 The polyr CD

The `polyr` CD deals with polynomials described in recursive format, so that the polynomial $2 * y^3 * z^5 + x + 1$ in $\mathbf{Z}[z][y][x]$ can be conceptually encoded as

```
poly_r_rep(x,
          term(1,1),
          term(0,poly_r_rep(y,
                            term(3,poly_r_rep(z,
                                              term(5,2))),
                            term(0,1))))
```

The symbols defined in the CD are the following.

`poly_r_rep` This takes a variable and then any number of `term` arguments in decreasing degree order, and constructs a polynomial in that variable with those terms.

`term` Takes two arguments: a degree (from $\mathbf{N}$) and a coefficient, and makes a term.

`polynomial_ring_r` This constructs the data type of a (recursive) polynomial ring, e.g. $\mathbf{Z}[x, y, z]$ (implemented as $\mathbf{Z}[z][y][x]$) would be described as follows.

```
<OMOBJ>
  <OMA>
    <OMS name="polynomial_ring_r" cd="polyr"/>
    <OMS name="Z" cd="setname1"/>
    <OMV name="x"/>
    <OMV name="y"/>
    <OMV name="z"/>
  </OMA>
</OMOBJ>
```

As can be seen, the first argument is the coefficient ring (which could itself be a polynomial domain) and the rest are variables.

`polynomial_r` This constructs a polynomial in a specific ring: the first argument is a `polynomial_ring_r` (see above) and the second is a `poly_r_rep` in that ring.

4.3 The polyd CD

The `polyd` CD deals with polynomials described in distributed format, so that the polynomial $x^2y^6 + 3y^5$ can be encoded (including the type of the ring to which it belongs) as

```
DMP(poly_ring_d(Z, 2),
    SDMP(term(1, 2, 6), term(3, 0, 5)))
```

This does not store the names of the variables, and the (**new**) constructor `poly_ring_d_named` will allow that, as in the following representation of the same polynomial.

```
DMP(poly_ring_d_named(Z, x, y),
    SDMP(term(1, 2, 6), term(3, 0, 5)))
```

`completely_reduced` (**new**). This is an attribute which can be attached to a `groebnered` object to indicate that the basis is completely reduced, i.e. no monomial is divisble by the leading monomial of any polynomial in the basis.

`DMP` This symbol takes two arguments: a distributed polynomial ring (built with the `poly_ring_d` symbol) and a polynomial (built with `SDMP`) and returns the polynomial in that ring.

`DMPL` As DMP, except that it takes an arbitrary number of `SDMP`s, and returns a list of polynomials (all in the same ring).

`elimination` One of the possible values for the `ordering` attribute, or the first argument of `groebnered`. It takes three arguments: the first is a number k of variables, the second is an ordering to apply to the first k variables, and the third is an ordering to apply as a tie-breaker to the rest of the variables, as in the following example.

```
<OMA>
  <OMS name="elimination" cd="polyd"/>
  <OMI> 1 </OMI>
  <OMS name="lexicographic" cd="polyd"/>
  <OMS name="graded_reverse_lexicographic" cd="polyd"/>
</OMA>
```

`graded_lexicographic` One of the possible values for the `ordering` attribute, or the first argument of `groebnered`.

`graded_reverse_lexicographic` One of the possible values for the `ordering` attribute, or the first argument of `groebnered`.

`groebner` This symbol represents the construction of a Gröbner basis: the first argument is an ordering, and the second a list of polynomials (i.e. a `DMPL`). If sent to a computational engine, the result should be a `groebnered` object.

`groebnered` This[5] is the constructor for an auto-reduced Gröbner basis. The first argument to this symbol is an ordering, and the second is a `DPML` representing the basis.

`lexicographic` One of the possible values for the `ordering` attribute, or the first argument of `groebnered`.

`matrix_ordering` (**new**). One of the possible values for the `ordering` attribute, or the first argument of `groebnered`. The argument is a matrix with as many columns as there are indeterminates (which should also be the rank of the matrix). Each row of the matrix is used in turn to decide the relative ranking of two monomials, by comparing the inner product of the row and the exponents of the monomials.

[5] Originally called `groebner_basis`.

`ordering` This is intended for use as an OpenMath attribute to specify how the monomials are ordered. Thus the polynomial $x^2y^6 + 3y^5$ can be more fully encoded as follows

```
<OMOBJ>
  <OMATTR>
    <OMATP>
      <OMS name="ordering" cd="polyd"/>
      <OMS name="graded_lexicographic" cd="polyd"/>
    </OMATP>
    <OMA>
      <OMS name="DMP" cd="polyd"/>
      <OMA>
        <OMS name="poly_ring_d" cd="polyd"/>
        <OMS name="Z" cd="setname1"/>
        <OMI> 2 </OMI>
      </OMA>
      <OMA>
        <OMS name="SDMP" cd="polyd"/>
        <OMA>
          <OMS name="term" cd="polyd"/>
          <OMI> 1 </OMI>
          <OMI> 2 </OMI>
          <OMI> 6 </OMI>
        </OMA>
        <OMA>
          <OMS name="term" cd="polyd"/>
          <OMI> 3 </OMI>
          <OMI> 0 </OMI>
          <OMI> 5 </OMI>
        </OMA>
      </OMA>
    </OMA>
  </OMATTR>
</OMOBJ>
```

`plus` This takes a `DMPL` as its (single) argument, and returns a `DMP` (in the same ring) representing the sum of the polynomials in the `DMPL`.

`poly_ring_d` This constructs a distributed polynomial ring (i.e. an object of type `polynomial_ring`). Its two arguments are the coefficient ring and the *number* of variables. Hence these are essentially polynomials in anonymous variables.

`poly_ring_d` (**new**). This constructs a distributed polynomial ring (i.e. an object of type `polynomial_ring`). Its first argument is the coefficient ring, and the rest are the variables. These will generally be `OMV`'s, but can also be the special symbol `anonymous` to indicate an anonymous variable.

`power` Takes two arguments, a `DMP` and a non-negative integer, and should return a `DMP` representing the appropriate power of the input `DMP`.

`reduce` The represents the reduction of the first argument, a polynomial (i.e. a `DMP`) with respect to the second argument, a Gröbner basis (i.e. a `groebnered` object). The result, if this is passed to a computational agent, should be a `DMP`.

`reverse_lexicographic` One of the possible values for the `ordering` attribute, or the first argument of `groebnered`.

`SDMP` The constructor for multivariate polynomials without any indication of variables or domain for the coefficients. Its arguments are just "monomial"s, built with the `term` constructor. No monomials should differ only by the coefficient (i.e. it is not permitted to have both $2xy$ and xy as monomials in a SDMP). SDMPs can be attributed with the "ordering" symbol to indicate a particular ordering of its monomials. This attribute shall not be set if the SDMP is part of a DMP or DMPL that has this attribute set.

`term` This symbol takes $n + 1$ arguments (where n is the number of variables in the relevant `poly_ring_d`): the first is the coefficient, and the rest are non-negative integers representing the exponents of the various variables.

`times` This takes a `DMPL` as its (single) argument, and returns a `DMP` (in the same ring) representing the product of the polynomials in the `DMPL`.

`weighted` (**new**). One of the possible values for the `ordering` attribute, or the first argument of `groebnered`. The first argument is a list of integers to act as variable weights, and the second is an ordering.

`weighted_degree` (**new**). This returns the maximum degree of any term *after* taking into account any weights supplied by means of the `weighted` ordering.

4.4 The `polyslp` CD

The `polyslp` CD deals with polynomials described in straight-line program format [7], so that the polynomial x^2y^2 can be represented as follows.

```
<OMOBJ>
  <OMA>
    <OMS cd="polyslp" name="polynomial_SLP"/>
    <OMA>
      <OMS cd="polyslp" name="poly_ring_SLP"/>
      <OMS cd="setname1" name="Z"/>
      <OMV name="x"/>
      <OMV name="y"/>
    </OMA>
    <OMA>
      <OMS cd="polyslp" name="prog_body"/>
      <OMA>
        <OMS cd="polyslp" name="inp_node"/>
```

```
        <OMV name="x"/>
      </OMA>
      <OMA>
        <OMS cd="polyslp" name="inp_node"/>
        <OMV name="y"/>
      </OMA>
      <OMA>
        <OMS cd="polyslp" name="op_node"/>
        <OMS cd="opnode" name="times"/>
        <OMI> 1 </OMI>
        <OMI> 1 </OMI>
      </OMA>
      <OMA>
        <OMS cd="polyslp" name="op_node"/>
        <OMS cd="opnode" name="times"/>
        <OMI> 2 </OMI>
        <OMI> 2 </OMI>
      </OMA>
      <OMA>
        <OMS cd="opnode" name="return"/>
        <OMA>
          <OMS cd="polyslp" name="op_node"/>
          <OMS cd="opnode" name="plus"/>
          <OMI> 3 </OMI>
          <OMI> 4 </OMI>
        </OMA>
      </OMA>
    </OMA>
  </OMA>
</OMOBJ>
```

The following are the symbls defined in the `polyslp` CD.

`const_node` This takes one argument, which is a value in the coefficient ring of the `poly_ring_SLP`.

`depth` This unary symbol represents the maximum depth of an SLP, i.e. the longest path from any node to a return node.

`inp_node` This takes one argument, which is the name of one of the variables in the `poly_ring_SLP`.

`left_ref` Takes as argument a node of an slp. Returns the value of the left hand pointer of the node.

`length` This unary symbol represents the length (number of arguments to `prog_body`) in an SLP.

`monte_carlo_eq` This represents a Monte-Carlo equality test, it takes three arguments, the first two are slps representing polynomials, the third argument is the maximum probability of incorrectness that is required of the equality test.

`node_selector` Takes an slp as the first argument, the second argument is the position of the required node. Returns the node of the slp at this position.

`op_node` This constructor takes three arguments. The first argument is a symbol from the `opnode` CD, meant to specify whether the node is a plus, minus, times or divide node, the second and third arguments are integers, which are the numbers of the lines which are the arguments of the operation.

`poly_ring_SLP` The constructor of the polynomial ring. The first argument is a ring, (the ring of the coefficients), the rest are the variables, in any order.

`polynomial_SLP` This actually builds a polynomial in a given SLP ring (the first argument). The second argument has to be a `prog_body`.

`prog_body` This takes n arguments, which are the instructions of a straight-line program. In particular they must be of types `const_node`, `inp_node` or `op_node`, possibly wrapped inside the `return` symbol from the `opnode` CD.

`quotient` A quotient function for polynomials represented by SLPs. It is a requirement that this is an exact division.

`return_node` Takes an slp as the argument, and returns the return node of the slp.

`right_ref` Takes as argument a node of an slp. Returns the value of the right hand pointer of the node.

`slp_degree` A unary symbol taking an SLP as argument and representing the apparent multiplicative degree of the SLP, without performing any cancellation.

The related `opnode` CD contains the symbols for the four binary arithmetic operations (`divide`, `minus`, `plus` and `times`), as well as the unary `return` symbol.

5 Dimensions and Units

There have been several well-publicised problems with the misunderstanding of units. However, before units can be formalised, dimensions have to be.

5.1 Dimensions

The CD `dimensions1` contains some fundamental and derived dimensions. The fundamental ones are `charge`, `length`, `mass`, `temperature` and `time`. The derived ones are `area`, `volume`, `speed`, `velocity`, `acceleration`, `force`, `pressure`, `current` and `voltage`. Formal Mathematical Properties link the derived ones to the fundamental ones.

5.2 Units

There are two CDs currently that capture units: `units_metric1` and `units_imperial1`. The definitions in these are fairly obvious. Though this has not yet been done, the conversion of imperial units to metric (for the fundamental dimensions) should be encoded as Formal Mathematical Properties, so that conversions could then be deduced for the other units by means of the Formal Mathematical Properties in the `dimensions1` CD.

6 Proofs

It is often said that OpenMath[6] cannot represent proofs. While this may well be true of the informal style of proof that much mathematics is written in, it ought not to be true of formal proofs. The CDs described in [6] had no specific support for proofs, so the author wrote the `logic3` CD, which essentially implements the following definition [8, p. 28]

Definition 1 *A proof of a theorem is a sequence of well-formed formulae, each of which is either an instance of an axiom, or follows from the previous formulae by one of the rules of inference, such that the last one is the theorem.*

The *syntactic* nature of "well-formed formulae", i.e. the rules that make $(p \wedge q)$ acceptable but $p \wedge ()q$ not acceptable in classicla logic, is not a problem for us, as the OpenMath DTD handles much of this problem, and the arity rules implied by the Small Type System [5] handle the rest.

In general, a proof of this nature will be laid about somewhat as follows.

$$\begin{array}{cr} ((a \Rightarrow ((a \Rightarrow a) \Rightarrow a)) \Rightarrow ((a \Rightarrow (a \Rightarrow a)) \Rightarrow (a \Rightarrow a))) & \text{Axiom 2} \\ (a \Rightarrow ((a \Rightarrow a) \Rightarrow a)) & \text{Axiom 1} \\ ((a \Rightarrow (a \Rightarrow a)) \Rightarrow (a \Rightarrow a)) & \text{MP 2,1} \\ ((a \Rightarrow (a \Rightarrow a)) & \text{Axiom 1} \\ (a \Rightarrow a) & \text{MP 4,3} \end{array}$$

where Axiom 2 is

$$((a \Rightarrow (b \Rightarrow c)) \Rightarrow ((a \Rightarrow b) \Rightarrow (a \Rightarrow c))). \tag{1}$$

One fundamental question is whether the OpenMath representation of a proof should contain the "justification", as in the textbook example above. In accordance with the spirit of keeping OpenMath objects self-contained, we decided that the justifications should be present, since otherwise a proof is only valid in some context declaring what the axioms are. This leads us to define the OpenMath symbol `axiom_instance`, whose first argument is the line of the proof, and whose second is the axiom being used, i.e. equation (1). Similarly, the rule of inference Modus Ponens, often written as

$$\frac{a \quad (a \rightarrow b)}{b},$$

[6]Why this shuld be a criticism of OpenMath, but not of MathML, is unclear.

is represented by the OpenMath symbol `ModusPonens`, whose three children are, in order, the line of the proof (i.e. b), the number of the line containing a and the number of the line containing b. In terms of the Small Type System [5], both `axiom_instance` and `ModusPonens` return objects of type `ProofLine`.

Since the key component is a sequence of weff-formed formulae, refered to, in the rules of inference, by their line numbers, it seems natural to this of a proof as a `list` of `ProofLine` items, so the OpenMath symbol asserting that one has a proof, `proof`, takes such a list as its argument.

In some circumstances we may wish to assert that P is a theorem, and give its proof, in other circumstances we may merely wish to assert, or even question, that a proof of the theoem exists. Having quoted earlier the principle that OpenMath objects should be self-contained, we may also ask "a proof with respect to which rules of inference". The `logic3` CD defines two such systems of rules of inference: propositional calculus (Modus Ponens is the only rule) and predicate calculus (Modus Ponens and `Generalisation` — $\frac{P(x)}{\forall x \quad P(x)}$). This therefore means that there are four symbols in the `logic3` CD for stating theorems: `pred_theorem` and `prop_theorem`, for stating that theorems exist, and `complete_pred_theorem` and `complete_prop_theorem` for stating that they exist and quoting the proof. There is also the Boolean-valued symbol `is_theorem` for asking whether a given theorem is true, i.e. whether there exists a proof of it. On an object of type `complete_????_theorem`, it should therefore always be `true`. A complete proof in OpenMath would therefore be as follows.

```
<OMOBJ>
  <OMA>
    <OMS cd="logic3" name="complete_prop_theorem"/>
    <OMA>
      <OMS cd="logic1" name="implies"/>
      <OMV name="A"/>
      <OMV name="A"/>
    </OMA>
    <OMA>
      <OMS cd="logic3" name="proof"/>
      <OMA>
        <OMS cd="list1" name="list"/>
        <OMA>
          <OMS cd="logic3" name="axiom_instance"/>
          <OMA>
            <OMS cd="logic1" name="implies"/>
            <OMA>
              <OMS cd="logic1" name="implies"/>
              <OMV name="a"/>
              <OMA>
                <OMS cd="logic1" name="implies"/>
                <OMA>
```

```
          <OMS cd="logic1" name="implies"/>
          <OMV name="a"/>
          <OMV name="a"/>
        </OMA>
        <OMV name="a"/>
      </OMA>
    </OMA>
    <OMA>
      <OMS cd="logic1" name="implies"/>
      <OMA>
        <OMS cd="logic1" name="implies"/>
        <OMV name="a"/>
        <OMA>
          <OMS cd="logic1" name="implies"/>
          <OMV name="a"/>
          <OMV name="a"/>
        </OMA>
      </OMA>
      <OMA>
        <OMS cd="logic1" name="implies"/>
        <OMV name="a"/>
        <OMV name="a"/>
      </OMA>
    </OMA>
  </OMA>
  <OMA>
    <OMS cd="logic1" name="implies"/>
    <OMA>
      <OMS cd="logic1" name="implies"/>
      <OMV name="a"/>
      <OMA>
        <OMS cd="logic1" name="implies"/>
        <OMV name="b"/>
        <OMV name="c"/>
      </OMA>
    </OMA>
    <OMA>
      <OMS cd="logic1" name="implies"/>
      <OMA>
        <OMS cd="logic1" name="implies"/>
        <OMV name="a"/>
        <OMV name="b"/>
      </OMA>
      <OMA>
```

```
        <OMS cd="logic1" name="implies"/>
        <OMV name="a"/>
        <OMV name="c"/>
      </OMA>
    </OMA>
  </OMA>
</OMA>
<OMA>
  <OMS cd="logic3" name="axiom_instance"/>
  <OMA>
    <OMS cd="logic1" name="implies"/>
    <OMV name="a"/>
    <OMA>
      <OMS cd="logic1" name="implies"/>
      <OMA>
        <OMS cd="logic1" name="implies"/>
        <OMV name="a"/>
        <OMV name="a"/>
      </OMA>
      <OMV name="a"/>
    </OMA>
  </OMA>
  <OMA>
    <OMS cd="logic1" name="implies"/>
    <OMV name="a"/>
    <OMA>
      <OMS cd="logic1" name="implies"/>
      <OMV name="b"/>
      <OMV name="a"/>
    </OMA>
  </OMA>
</OMA>
<OMA>
  <OMS cd="logic3" name="ModusPonens"/>
  <OMA>
    <OMS cd="logic1" name="implies"/>
    <OMA>
      <OMS cd="logic1" name="implies"/>
      <OMV name="a"/>
      <OMA>
        <OMS cd="logic1" name="implies"/>
        <OMV name="a"/>
        <OMV name="a"/>
      </OMA>
```

```
          </OMA>
          <OMA>
            <OMS cd="logic1" name="implies"/>
            <OMV name="a"/>
            <OMV name="a"/>
          </OMA>
        </OMA>
        <OMI> 2 </OMI>
        <OMI> 1 </OMI>
      </OMA>
      <OMA>
        <OMS cd="logic3" name="axiom_instance"/>
        <OMA>
          <OMS cd="logic1" name="implies"/>
          <OMV name="a"/>
          <OMA>
            <OMS cd="logic1" name="implies"/>
            <OMV name="a"/>
            <OMV name="a"/>
          </OMA>
        </OMA>
        <OMA>
          <OMS cd="logic1" name="implies"/>
          <OMV name="a"/>
          <OMA>
            <OMS cd="logic1" name="implies"/>
            <OMV name="b"/>
            <OMV name="a"/>
          </OMA>
        </OMA>
      </OMA>
      <OMA>
        <OMS cd="logic3" name="ModusPonens"/>
        <OMA>
          <OMS cd="logic1" name="implies"/>
          <OMV name="a"/>
          <OMV name="a"/>
        </OMA>
      </OMA>
    </OMA>
  </OMA>
</OMA>
</OMOBJ>
```

Relatively early on in formal logic, one comes across deductions, generally defined as follows.

Definition 2 *A deduction of a conclusion from a given set of hypotheses is a sequence of well-formed formulae, each of which is an instance of an axiom, one of the hypotheses, or follows from the previous formulae by one of the rules of inference, such that the last one is the conclusion.*

The concept of hypothesis is catered for by adding the extra symbol `Hypothesis`, whose single argument is the hypothesis being quoted, and which returns a `ProofLine` object. There are four symbols in the `logic3` CD for stating deductions: `pred_deduction` and `prop_deduction`, for stating that deductions exist, and `complete_pred_deduction` and `complete_prop_deduction` for stating that they exist and quoting the proof. There is also the Boolean-valued symbol `is_deduction` for asking wheher a given deduction is true, i.e. whether there exists a proof of it. On an object of type `complete_????_deduction`, it should therefore always be `true`. A complete, though trivial, deduction in OpenMath would therefore be as follows.

```
<OMA>
  <OMS name="complete_prop_deduction" cd="logic3"/>
  <OMA name="a"/>
  <OMA>
    <OMS name="set" cd="set1"/>
    <OMA name="a"/>
  </OMA>
  <OMA>
    <OMS name="proof" cd="logic3"/>
    <OMA>
      <OMS name="list" cd="list1"/>
      <OMA>
        <OMS name="Hypothesis" cd="logic3"/>
        <OMA name="a"/>
      </OMA>
    </OMA>
  </OMA>
</OMA>
```

The deduction theorem of propositional calculus, often stated as

$$H \vdash P \text{ implies } \vdash (H \rightarrow P),$$

but which is more correctly written as

$$\{H\} \vdash P \text{ implies } \vdash (H \rightarrow P),$$

would therefore be written as follows in OpenMath.

```
<OMA>
  <OMS name="implies" cd="logic1"/>
  <OMA>
    <OMS name="is_deduction" cd="logic3"/>
    <OMA>
      <OMS name="prop_deduction"/>
      <OMV name="P"/>
      <OMA>
        <OMS name="set" cd="set1"/>
        <OMV name="H"/>
      </OMA>
    </OMA>
  </OMA>
  <OMA>
    <OMS name="is_theorem" cd="logic3"/>
    <OMA>
      <OMS name="prop_theorem"/>
      <OMA>
        <OMS name="implies" cd="logic1"/>
        <OMV name="H"/>
        <OMV name="P"/>
      </OMA>
    </OMA>
  </OMA>
</OMA>
```

We should note that it is much harder to express in OpenMath the deduction theorem of the predicate calculus, whose standard statement is as follows

Theorem 1 *Assume that in some deducion showing that $\Gamma, \mathcal{A} \vdash \mathcal{B}$, no application of generalisation to a well-formed formula that depends on $\mathcal{A}$ has as its quantified variable a free variable of $\mathcal{A}$. Then $\Gamma \vdash \mathcal{A} \Rightarrow \mathcal{B}$.*

[8, p. 59].

7 Special Functions

Some work had been done on these, both at INRIA and at the University of Western Ontario, but these efforts needed to be brought together and unified. As with elementary functions [3], there is a great need for precision over branch cuts etc. It is desirable (and planned) to co-operate with NIST's revisions and computerisation of Abramowitz & Stegun.

Table 1: Style A

```
<OMA>
  <OMA>
    <OMS name="J" cd="Bessel"/>
    <OMV name="nu"/>
  </OMA>
  <OMV name="z"/>
</OMA>
```

Table 2: Style B

```
<OMA>
  <OMS name="BesselJ" cd="specfun"/>
  <OMV name="nu"/>
  <OMV name="z"/>
</OMA>
```

There are two design choices to be made here, both of which are illustrated in the two fragments above for $J_\nu(z)$.

- Do we have many CDs of special functions, as in style A, or one large CD, as in style B. The name `BesselJ` corresponds to several computer algebra systems, but `J` to mathematical notation. At the first meeting of the network in Berlin in August 2001, it was decided to adopt the "many CDs" approach.
- To curry (as in style A) or not to curry (as in style B): that is the question. Currying means that one can easily talk about the function J_ν in the abstract, whereas the uncurried style means that one has to use $\lambda z.J_\nu(z)$. It was decided to adopt the curried approach.

8 Future work on CDs

In a very real sense, like mathematics itself, CDs will never be finished. However, it is possible to identify some tasks that are relatively urgent from the point of view of having a more usable set of CDs. The following list partly reflects the author's prejudices, and other contributions would be welcome.

Abstract Algebra Many more CDs need to be written and/or formalised in this area. Here too, there are problems of consistency:

Degree S_{12}, M_{12} are permutation groups acting on 12 symbols;
Size F_{20} is a permutation group of size 20, normally acting on 5 elements;
Unclear Is D_{12} a group with 12 elements acting on six points, or a group with 24 elements acting on 12 points? One can find both in the literature, though the second is probably more common. It would make sense to follow [2]

OpenMath also needs to deal with ideals etc., rather than just lists of polynomials (different lists can represent the same ideal).

Algorithms A CD to describe algorithmic concepts would be useful, partly from the point of view of the wider publication-related aspects of OpenMath, and partly for use in concepts such as symbolic differentiation, i.e. differentiating an algorithm, where co-operation between software packages is important.

Logics While basic classical logic (propositional, predicate) is catered for, there is nothing on proofs, or other forms of logic (intuitionistic etc.). Different concepts of equality also need to be handled.

References

[1] Caprotti,O., Carlisle,D.P. & Cohen,A.M. (Eds), The OpenMath Standard version 1.0. The OpenMath Esprit Consortium, Feb. 2000. `http://www.nag.co.uk/projects/OpenMath/omstd/omstd.pdf`.

[2] Conway,J.H., Hulpke,A. & McKay,J., On Transitive Permutation Groups. *LMS J. Computation and Math.* **1** (1998) pp. 1–8.

[3] Corless,R.M., Davenport,J.H., Jeffrey,D.J. & Watt,S.M., According to Abramowitz & Stegun. *ACM SIGSAM Bulletin* **30** (2000) 2 pp. 58–65.

[4] Davenport,J.H., On Writing OpenMath Content Dictionaries. *ACM SIGSAM Bulletin* **30** (2000) 2 pp. 12–15.

[5] Davenport,J.H., A Small OpenMath Type System. *ACM SIGSAM Bulletin* **30** (2000) 2 pp. 16–21.

[6] Dewar,M.C., OpenMath: An Overview. *ACM SIGSAM Bulletin* **30** (2000) 2 pp. 2–5.

[7] Freeman,T., Imirzian,G. & Kaltofen,E., A System for Manipulating Polynomials Given by Straight-Line Programs. Proc. SYMSAC 86 (ACM, New York, 1986) pp. 169–175.

[8] Mendelson,E., *Introduction to Mathematical logic*. Wadsworth, Monterey CA, 1987.

[9] Mathematical Markup Language (MathML) Version 2.0. World-Wide Web Consortium, 21 February 2001. `http://www.w3c.org/TR/MathML2`.