The OpenMath Standard

The OpenMath Society

Editors

S.Buswell, O.Caprotti, D.P.Carlisle, M.C.Dewar, M.Gaetano
and M.Kohlhase

Abstract

This document proposes *OpenMath* as a standard for the communication of semantically rich mathematical objects. This draft of the *OpenMath* standard comprises the following: a description of *OpenMath* objects, the grammar of XML and of the binary encoding of objects, a description of Content Dictionaries and an XML document type definition for validating Content Dictionaries. The non-normative Chapter 1 of this document briefly overviews the history of *OpenMath*.

Contents

List of Figures

Chapter 1

OpenMath Movement

This chapter is a historical account of *OpenMath* and should be regarded as non-normative.

OpenMath is a standard for representing mathematical objects, allowing them to be exchanged between computer programs, stored in databases, or published on the worldwide web. While the original designers were mainly developers of computer algebra systems, it is now attracting interest from other areas of scientific computation and from many publishers of electronic documents with a significant mathematical content. There is a strong relationship to the MathML recommendation [18] from the Worldwide Web Consortium, and a large overlap between the two developer communities. MathML deals principally with the *presentation* of mathematical objects, while *OpenMath* is solely concerned with their semantic meaning or *content*. While MathML does have some limited facilities for dealing with content, it also allows semantic information encoded in *OpenMath* to be embedded inside a MathML structure. Thus the two technologies may be seen as highly complementary.

1.1   History

OpenMath was originally developed through a series of workshops held in Zurich (1993 and 1996), Oxford (1994), Amsterdam (1995), Copenhagen (1995), Bath (1996), Dublin (1996), Nice (1997), Yorktown Heights (1997), Berlin (1998), and Tallahassee (1998). The participants in these workshops formed a global *OpenMath* community which was coordinated by a Steering Committee and operated through electronic mailing groups and ad-hoc working parties. This loose arrangement has been formalised through the establishment of an *OpenMath* Society. Up until the end of 1996 much of the work of the community was funded through a grant from the Human Capital and Mobility program of the European Union, the contributions of several institutions and individuals. A document outlining the objectives and basic design of *OpenMath* was produced (later published as [1]). By the end of 1996 a simplified specification had been agreed on and some prototype implementations have come about [5].

In 1996 a group of European participants in *OpenMath* decided to bid for funding under the Euro-

pean Union's Fourth Framework Programme for strategic research in information technology. This bid was successful and the project started in late 1997. The principal aims of the project are to formalise *OpenMath* as a standard and to develop it further through industrial applications; this document is a product of that process and draws heavily on the previous work described earlier. *OpenMath* participants from all over the world continue to meet regularly and cooperate on areas of mutual interest, and recent workshops in Tallahassee (November 1998) and Eindhoven (June 1999) endorsed drafts of this document as the current *OpenMath* standard.

1.2   *OpenMath* Society

In November 1998 the *OpenMath* Society has been established to coordinate all *OpenMath* activities. The society is based in Helsinki, Finland and is steered by the executive committee whose members are elected by the society. The official web page of the society is http://www.openmath.org.

Chapter 2

Introduction to *OpenMath*

This chapter briefly introduces *OpenMath* concepts and notions that are referred to in the rest of this document.

2.1 *OpenMath* Architecture

The architecture of *OpenMath* is described in Figure 2.1 and summarizes the interactions among the different *OpenMath* components. There are three layers of representation of a mathematical object [12]. A private layer that is the internal representation used by an application. An abstract layer that is the representation as an *OpenMath* object. Third is a communication layer that translates the *OpenMath* object representation to a stream of bytes. An application dependent program manipulates the mathematical objects using its internal representation, it can convert them to *OpenMath* objects and communicate them by using the byte stream representation of *OpenMath* objects.

2.2 *OpenMath* Objects and Encodings

OpenMath objects are representations of mathematical entities that can be communicated among various software applications in a meaningful way, that is, preserving their "semantics".

OpenMath objects and encodings are described in detail in Chapter 3 and Chapter 4.

The standard endorses encodings in XML and binary format. These are the encodings supported by the official *OpenMath* libraries. However they are not the only possible encodings of *OpenMath* objects. Users that wish to define their own encoding using some other specific language (e.g. Lisp) may do so provided there is an effective translation of this encoding to an official one.
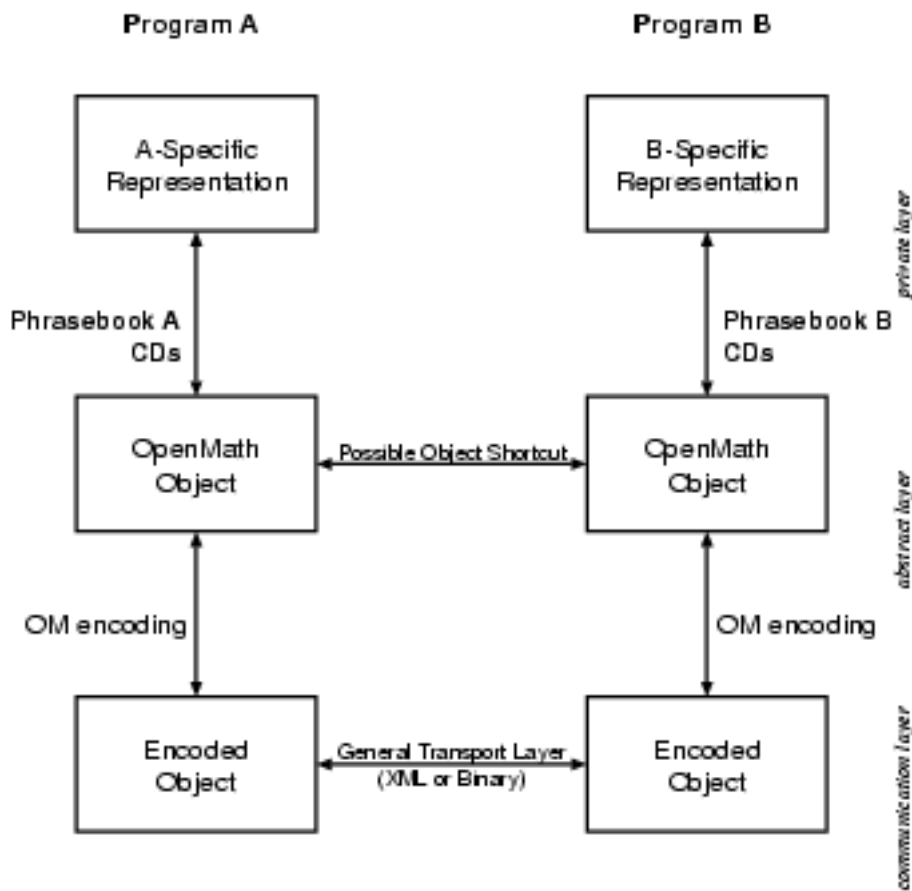
Figure 2.1: The *OpenMath* Architecture

2.3 Content Dictionaries

Content Dictionaries (CDs) are used to assign informal and formal semantics to all symbols used in the *OpenMath* objects. They define the symbols used to represent concepts arising in a particular area of mathematics.

The Content Dictionaries are public, they represent the actual common knowledge among *Open-Math* applications. Content Dictionaries fix the "meaning" of objects independently of the application. The application receiving the object may then recognize whether or not, according to the semantics of the symbols defined in the Content Dictionaries, the object can be transformed to the corresponding internal representation used by the application.

2.4 Additional Files

Several additional files are related to Content Dictionaries. Signature files contain the signatures of symbols defined in some *OpenMath* Content Dictionary and their format is endorsed by this standard.

Furthermore, the standard fixes how to define as a CDGroup a specific set of Content Dictionaries.

Auxiliary files that define presentation and rendering or that are used for manipulating and processing Content Dictionaries are not discussed by the standard.

2.5 Phrasebooks

The conversion of an *OpenMath* object to/from the internal representation in a software application is performed by an interface program called *Phrasebook*. The translation is governed by the Content Dictionaries and the specifics of the application. It is envisioned that a software application dealing with a specific area of mathematics declares which Content Dictionaries it understands. As a consequence, it is expected that the Phrasebook of the application is able to translate *OpenMath* objects built using symbols from these Content Dictionaries to/from the internal mathematical objects of the application.

OpenMath objects do not specify any computational behaviour, they merely represent mathematical expressions. Part of the *OpenMath* philosophy is to leave it to the application to decide what it does with an object once it has received it. *OpenMath* is not a query or programming language. Because of this, *OpenMath* does not prescribe a way of forcing "evaluation" or "simplification" of objects like $2 + 3$ or $sin(\pi)$. Thus, the same object $2 + 3$ could be transformed to 5 by a computer algebra system, or displayed as $2 + 3$ by a typesetting tool.

Chapter 3

OpenMath Objects

In this chapter we provide a self-contained description of *OpenMath* objects. We first do so by means of an abstract grammar description (Section 3.1) and then give a more informal description (Section 3.2).

3.1 Formal Definition of *OpenMath* Objects

OpenMath represents mathematical objects as terms or as labelled trees that are called *OpenMath* objects or *OpenMath* expressions. The definition of an abstract *OpenMath* object is then the following.

3.1.1 Basic *OpenMath* objects

The Basic *OpenMath* Objects form the leaves of the *OpenMath* Object tree. A Basic *OpenMath* Object is of one of the following.

- (i) Integer.

  Integers in the mathematical sense, with no predefined range. They are "infinite precision" integers (also called "bignums" in computer algebra).

- (ii) IEEE floating point number.

  Double precision floating-point numbers following the IEEE 754-1985 standard [8].

- (iii) Character string.

  A Unicode Character string. This also corresponds to 'characters' in XML.

- (iv) Bytearray.

  A sequence of bytes.

- (v) Symbol.

  A Symbol encodes three fields of information, a *name*, a *Content Dictionary*, and (optionally) a role. The name of a symbol is a sequence of characters matching the regular expression described in Section 3.3. The Content Dictionary is the location of the definition of the symbol, consisting of a name (a sequence of characters matching the regular expression described in Section 3.3) and, optionally, a unique prefix called a *cdbase* which is used to disambiguate multiple Content Dictionaries of the same name. The role is a restriction on where the symbol may appear in an *OpenMath* object. The possible roles are described in Section 3.1.4.

- (vi) Variable.

  A Variable must have a *name* which is a sequence of characters matching a regular expression, as described in Section 3.3. Where the variable is a member of an enumerated set it also has a child which is an *OpenMath* object representing its indices.

3.1.2 Derived *OpenMath* Objects

A derived*OpenMath* object is built as follows:

- (i) If A is *not* an *OpenMath* object, then **foreign**(A) is an *OpenMath foreign object*.

3.1.3 Compound *OpenMath* Objects

OpenMath objects are built recursively as follows.

- (i) Basic *OpenMath* objects are *OpenMath* objects. Derived *OpenMath* objects are *not OpenMath* objects.
- (ii) If $A_1, \ldots, A_n$ $(n > 0)$ are *OpenMath* objects, then

$$\mathbf{application}(A_1, \ldots, A_n)$$

  is an *OpenMath application object*.
- (iii) If $S_1, \ldots, S_n$ are *OpenMath* symbols, and A is an *OpenMath* object, and $A_1, \ldots, A_n$ $(n > 0)$ are *OpenMath* objects or *OpenMath* derived objects, then

$$\mathbf{attribution}(A, S_1\ A_1, \ \ldots \ , S_n\ A_n)$$

  is an *OpenMath attribution object*. A is the object *stripped of attributions*. $S_1, \ldots, S_n$ are referred to as *keys* and $A_1, \ldots, A_n$ as their associated *values*. The operation of recursively applying stripping to the stripped object is called *flattening of the attribution*. When the stripped object after flattening is a variable, the attributed object is called *attributed variable*.
- (iv) If B and C are *OpenMath* objects, and $v_1, \ldots, v_n$ $(n \geq 0)$ are *OpenMath* variables or attributed variables, then

$$\mathbf{binding}(B, v_1, \ldots, v_n, C)$$

  is an *OpenMath binding object*.

- (v) If *S* is an *OpenMath* symbol and $A_1, \ldots, A_n$ $(n \geq 0)$ are *OpenMath* objects, then

$$\mathbf{error}(S, A_1, \ldots, A_n)$$

is an *OpenMath error object*.

3.1.4 *OpenMath* Symbol Rôles

The *rôle* of an *OpenMath* symbol is a restriction on where it can appear in an *OpenMath* object. A symbol cannot have more than one role. If no role is indicated then the symbol can be used anywhere. Possible roles are:

  (i) *binder* The symbol may only appear as the first child of an *OpenMath* binding object.
 (ii) *attribution* The symbol may only be used as key in an *OpenMath* attribution object, i.e. as the first element of a key-value pair, or in an equivalent context (for example to refer to the value of an attribution). This form of attribution may be ignored by an application, so should be used for information which does not change the meaning of the attributed *OpenMath* object.
(iii) *semantic-attribution* This is the same as *attribution* except that it modifies the meaning of the attributed *OpenMath* object and thus cannot be ignored by an application.
(iv) *error* The symbol can only appear as the first child of an *OpenMath* error object.
 (v) *default* The symbol can appear anywhere not defined in the previous four cases.

3.2 Further Description of *OpenMath* Objects

Informally, an *OpenMath object* can be viewed as a tree and is also referred to as a term. The objects at the leaves of *OpenMath* trees are called *basic objects*. The basic objects supported by *OpenMath* are:

Integer Arbitrary Precision integers.

Float *OpenMath* floats are IEEE 754 Double precision floating-point numbers. Other types of floating point number may be encoded in *OpenMath* by the use of suitable content dictionaries.

Character strings are sequences of characters. These characters come from the Unicode standard [14].

Bytearrays are sequences of bytes. There is no "byte" in *OpenMath* as an object of its own. However, a single byte can of course be represented by a bytearray of length 1. The difference between strings and bytearrays is the following: a character string is a sequence of bytes with a fixed interpretation (as characters, Unicode texts may require several bytes to code one character), whereas a bytearray is an uninterpreted sequence of bytes with no intrinsic meaning. Bytearrays could be used inside *OpenMath* errors to provide information to, for example, a debugger; they could also contain intermediate results of calculations, or 'handles' into computations or databases.
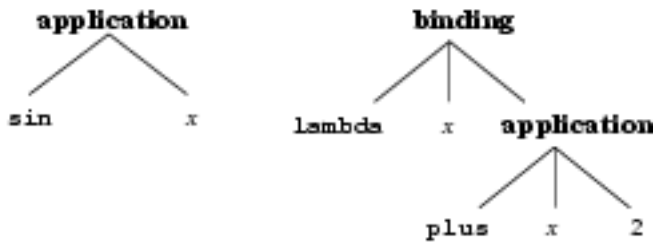
Figure 3.1: The *OpenMath* application and binding objects for $sin(x)$ and $\lambda x.x + 2$ in tree-like notation.

Symbols are uniquely defined by the Content Dictionary in which they occur and by a name. The form of these definitions is explained in Chapter 5. Each symbol has no more than one definition in a Content Dictionary. Many Content Dictionaries may define differently a symbol with the same name (e.g. the symbol `union` is defined as associative-commutative set theoretic union in a Content Dictionary `set1` but another Content Dictionary, `multiset1` might define a symbol `union` as the union of multi-sets).

Variables are meant to denote parameters, variables or indeterminates (such as bound variables of function definitions, variables in summations and integrals, independent variables of derivatives).

A variable may have one or more children which are themselves *OpenMath* objects and are treated as scripts. Thus it is possible to create a variable with cardinality such as x_1 or x_i.

Currently there is one way of making a derived *OpenMath* object.

Foreign is used to import a non-*OpenMath* object into an *OpenMath* attribution. Examples of its use could be to annotate a formula with a visual or aural rendering, an animation etc.

The four following constructs can be used to make compound *OpenMath* objects.

Application constructs an *OpenMath* object from a sequence of one or more *OpenMath* objects. The first argument of application is referred to as "head" while the remaining objects are called "arguments". An *OpenMath* application object can be used to convey the mathematical notion of application of a function to a set of arguments. For instance, suppose that the *OpenMath* symbol *sin* is defined in a Content Dictionary for trigonometry, then **application**(sin, x) is the abstract *OpenMath* object corresponding to $sin(x)$. More generally, an *OpenMath* application object can be used as a constructor to convey a mathematical object built from other objects such as a polynomial constructed from a set of monomials. Constructors build inhabitants of some symbolic type, for instance the type of rational numbers or the type of polynomials. The rational number, usually denoted as $1/2$, is represented by the *OpenMath* application object **application**$(Rational, 1, 2)$. The symbol *Rational* must be defined, by a Content Dictionary, as a constructor symbol for the rational numbers.

Binding objects are constructed from an *OpenMath* object, and from a sequence of zero or more variables followed by another *OpenMath* object. The first *OpenMath* object is the "binder" object. Arguments 2 to $n-1$ are always variables to be bound in the "body" which is the n^{th} argument object. It is allowed to have no bound variables, but the binder object and the body should be present. Binding can be used to express functions or logical statements. The function $\lambda x.x + 2$, in which the variable x is bound by λ, corresponds to a binding object having as binder the *OpenMath* symbol *lambda*:

$$\textbf{binding}(lambda, x, \textbf{application}(plus, x, 2)).$$

Phrasebooks are allowed to use α conversion in order to avoid clashes of variable names. Suppose an object Ω contains an occurrence of the object **binding**(B, v, C). This object **binding**(B, v, C) can be replaced in Ω by **binding**(B, z, C') where z is a variable not occurring free in C and C' is obtained from C by replacing each free (i.e., not bound by any intermediate **binding** construct) occurrence of v by z. This operation preserves the semantics of the object Ω. In the above example, a phrasebook is thus allowed to transform the object to, e.g.

$$\textbf{binding}(lambda, z, \textbf{application}(plus, z, 2)).$$

Repeated occurrences of the same variable in a binding operator are allowed. An *OpenMath* application should treat a binding with multiple occurrences of the same variable as equivalent to the binding in which all but the last occurrence of each variable is replaced by a new variable which does not occur free in the body of the binding.

$$\textbf{binding}(lambda, v, v, \textbf{application}(times, v, v))$$

is semantically equivalent to:

$$\textbf{binding}(lambda, v', v, \textbf{application}(times, v, v))$$

so that the resulting function is actually a constant in its first argument (v' does not occur free in the body **application**$(times, v, v)$)).

Attribution decorates an object with a sequence of one or more pairs made up of an *OpenMath* symbol, the "attribute", and an associated object, the "value of the attribute". The value of the attribute can be an *OpenMath* attribution object itself. As an example of this, consider the *OpenMath* objects representing groups, automorphism groups, and group dimensions. It is then possible to attribute an *OpenMath* object representing a group by its automorphism group, itself attributed by its dimension.

OpenMath objects can be attributed with *OpenMath* foreign objects, which are containers for non-*OpenMath* structures. For example a mathematical expression could be attributed with its spoken or visual rendering.

Composition of attributions, as in

$$\textbf{attribution}(\textbf{attribution}(A, S_1\ A_1, \ldots, S_h\ A_h), S_{h+1}\ A_{h+1}, \ldots, S_n\ A_n)$$

is semantically equivalent to a single attribution, that is

$$\textbf{attribution}(A, S_1 \ A_1, \ldots, S_h \ A_h, S_{h+1} \ A_{h+1}, \ldots, S_n \ A_n).$$

The operation that produces an object with a single layer of attribution is called *flattening*.

Multiple attributes with the same name are allowed. While the order of the given attributes does not imply any notion of priority, potentially it could be significant. For instance, consider the case in which $S_h = S_n \ (h < n)$ in the example above. Then, the object is to be interpreted as if the value A_n overwrites the value A_h. (*OpenMath* however does not mandate that an application preserves the attributes or their order.)

Attribution acts as either adornment annotation or as semantical annotation. When the key has rôle *attribution*, then replacement of the attributed object by the object itself is not harmful and preserves the semantics. When the key has rôle *semantic-attribution* then the attributed object is modified by the attribution and cannot be viewed as semantically equivalent to the stripped object. If the attribute lacks the rôle specification then attribution is acting as adornment annotation.

Objects can be decorated in a multitude of ways. In [3], typing of *OpenMath* objects is expressed by using an attribution. The object $\textbf{attribution}(A, type \ t)$ represents the judgment stating that object A has type t. Note that both A and t are *OpenMath* objects.

Error is made up of an *OpenMath* symbol and a sequence of zero or more *OpenMath* objects. This object has no direct mathematical meaning. Errors occur as the result of some treatment on an *OpenMath* object and are thus of real interest only when some sort of communication is taking place. Errors may occur inside other objects and also inside other errors. Error objects might consist only of a symbol as in the object: $\textbf{error}(S)$.

3.3   Names

The names of symbols, variables and content dictionaries must conform to the following rules, which are designed to be compatible with standards such as Unicode and XML. These standards group individual characters into letters, digits, combining characters and extenders. Informally, these are defined as follows:

Letters are elements of an alphabet such as latin, cyrillic, kanji etc.

Digits are atomic numbers, from which compound numbers can be constructed (for example '1' is a digit but '11' is not).

Combining Characters are used to combine several characters to produce a new one, for example an accented character.

Extenders are characters which are neither letters nor combining characters but modify the appearance of other characters in some way.

Formally, we use the precise definitions given in the Unicode standard [14].

Then a legal *OpenMath* name is defined by the following grammar:

Name	$\longrightarrow$	(Letter | '_') (Char)*
Char	$\longrightarrow$	Letter | Digit | '.' | '-' | '_' | CombiningChar | Extender

3.3.0.0.1 CD Base A cdbase must conform to the grammar for URIs described in [9]. Note that if non-ASCII characters are used in a CD or symbol name then when a URI for that symbol is constructed it will be necessary to map the non-ASCII characters to a sequence of octets. The precise mechanism for doing this depends on the URI scheme.

3.3.0.0.2 Note on content dictionary names It is a common convention to store a Content Dictionary in a file of the same name, which can cause difficulties on many file systems. If this convention is to be followed then *OpenMath recommends* that the name be restricted to the subset of the above grammar which is a legal POSIX [7] filename, namely:

Name	$\longrightarrow$	(PosixLetter | '_') (Char)*
Char	$\longrightarrow$	PosixLetter | Digit | '.' | '-' | '_'
PosixLetter	$\longrightarrow$	'a' | 'b' | ... | 'z' | 'A' | 'B' | ... | 'Z'

3.4 Summary

- *OpenMath* supports basic objects like integers, symbols, floating-point numbers, character strings, bytearrays, and variables.
- *OpenMath* compound objects are of four kinds: applications, bindings, errors, and attributions.
- *OpenMath* objects may be attributed with non-*OpenMath* objects via the use of foreign *OpenMath* objects.
- *OpenMath* objects have the expressive power to cover all areas of computational mathematics.

Observe that an *OpenMath* application object is viewed as a "tree" by software applications that do not understand Content Dictionaries, whereas a Phrasebook that understands the semantics of the symbols, as defined in the Content Dictionaries, should interpret the object as functional application, constructor, or binding accordingly. Thus, for example, for some applications, the *OpenMath* object corresponding to $2 + 5$ may result in a command that writes 7.

Chapter 4

OpenMath Encodings

In this chapter, two encodings are defined that map between *OpenMath* objects and byte streams. These byte streams constitute a low level representation that can be easily exchanged between processes (via almost any communication method) or stored and retrieved from files.

The first encoding is a character-based encoding in XML format. In previous versions of the *OpenMath* Standard this encoding was a restricted subset of the full legal XML syntax. In this version, however, we have removed all these restrictions so that the earlier encoding is a strict subset of the existing one. The XML encoding can be used, for example, to send *OpenMath* objects via e-mail, cut-and-paste, etc. and to embed *OpenMath* objects in XML documents or to have *OpenMath* objects processed by XML-aware applications.

The second encoding is a binary encoding that is meant to be used when the compactness of the encoding is important (interprocess communications over a network is an example).

Note that these two encodings are sufficiently different for autodetection to be effective: an application reading the bytes can very easily determine which encoding is used.

4.1   The XML Encoding

This encoding has been designed with two main goals in mind:

1. to provide an encoding that uses common character sets (so that it can be easily included in most documents and transport protocols) and that is both readable and writable by a human.
2. to provide an encoding that can be included (embedded) in XML documents or processed by XML-aware applications.

4.1.1 A Schema for the XML Encoding

The XML encoding of an *OpenMath* object is defined by the Relax NG schema [11] given below. Relax NG has a number of advantages over the older XSD Schema format [15], in particular it allows for tighter control of attributes and has a modular, extensible structure. Although we have made the XML form, which is given in Appendix B normative, it is generated from the compact syntax given below. It is also very easy to restrict the schema to allow a limited set of *OpenMath* symbols as described in Appendix C.

Standard tools exist for generating a DTD or an XSD schema from a Relax NG Schema. Examples of such documents are given in Appendix E and Appendix D respectively.

```
# RELAX NG Schema for OpenMath 2


default namespace om = "http://www.openmath.org/OpenMath"
namespace xlink = "http://www.w3.org/1999/xlink"

# OM2: allow OMR
omel =
  OMS | OMV | OMI | OMB | OMSTR | OMF | OMA | OMBIND | OME | OMATTR |OMR

# things which can be variables
omvar = OMV | attvar

attvar = element OMATTR { common.attributes,(OMATP , (OMV | attvar))}

#OM2: common attributes
cdbase = attribute cdbase { xsd:anyURI}?
common.attributes = (attribute id { xsd:ID })?
compound.attributes = common.attributes,cdbase

# symbol
OMS = element OMS { common.attributes, attlist.OMS}
attlist.OMS =
  attribute name { xsd:NCName},
  attribute cd { xsd:NCName},
  cdbase

# variable
OMV = element OMV { common.attributes, attlist.OMV,omel?}
attlist.OMV = attribute name { xsd:NCName}

# integer
```

```
OMI = element OMI { common.attributes,
                    xsd:string {pattern = "\s*(-\s?)?[0-9]+(\s[0-9]+)*\s*"}}
# byte array
OMB = element OMB { common.attributes, xsd:base64Binary }


# string
OMSTR = element OMSTR { common.attributes, text }


# floating point
OMF = element OMF { common.attributes, attlist.OMF}
attlist.OMF =
  attribute dec { xsd:string
          {pattern = "(-?)([0-9]+)?(\.[0-9]+)?(e([+\-]?)[0-9]+)?"}}|
  attribute hex { xsd:string {pattern = "[0-9A-F]+"}}


# apply constructor
OMA = element OMA { compound.attributes, omel+ }
# binding constructor and variable
OMBIND = element OMBIND { compound.attributes, omel, OMBVAR, omel }
OMBVAR = element OMBVAR { common.attributes, omvar+ }


# error
OME = element OME { common.attributes, OMS, omel* }


# attribution constructor and attribute pair constructor
OMATTR = element OMATTR { compound.attributes, OMATP, omel }


# OM2: allow OMFOREIGN
OMATP = element OMATP { compound.attributes, (OMS, (omel | OMFOREIGN) )+ }


# OM2: OMFOREIGN
OMFOREIGN =  element OMFOREIGN { compound.attributes, (omel|notom)* }


# Any elements not in the om namespace (valid om is allowed as a descendant)
notom =
  (element * - om:* {attribute * { text }*,(omel|notom)*}
   | text)


# OM object constructor
OMOBJ = element OMOBJ { compound.attributes, omel }
 attlist.OMOBJ = attribute version { "1.0" | "1.2" | "2.0" }



# OM2: OMR
```

```
OMR = element OMR { common.attributes, attlist.OMR }
attlist.OMR =
        attribute xlink:href { text },
        attribute xlink:type {"simple"},
        attribute xlink:show {"embed"}

start = OMOBJ
```

4.1.2 Informal description of the XML Encoding

An encoded *OpenMath* object is placed inside an `OMOBJ` element. This element can contain the elements (and integers) described above. It can take an optional `version` XML-attribute which indicates to which version of the *OpenMath* standard it conforms. This would allow an older *OpenMath* application which, for example, could not parse the full XML syntax, to accept objects with version "1" or "1.1" but reject objects with version "2".

We briefly discuss the XML encoding for each type of *OpenMath* object starting from the basic objects.

Integers are encoded using the `OMI` element around the sequence of their digits in base 10 or 16 (most significant digit first). White space may be inserted between the characters of the integer representation, this will be ignored. After ignoring white space, integers written in base 10 match the regular expression `-?[0-9]+`. Integers written in base 16 match `-?x[0-9A-F]+`. The integer 10 can be thus encoded as `<OMI> 10 </OMI>` or as `<OMI> xA </OMI>` but neither `<OMI> +10 </OMI>` nor `<OMI> +xA </OMI>` can be used.

The negative integer -120 can be encoded as either as decimal `<OMI> -120 </OMI>` or as hexadecimal `<OMI> -x78 </OMI>`.

Symbols are encoded using the `OMS` element. This element has three XML-attributes `cd`, `name`, and `cdbase`. The value of `cd` is the name of the Content Dictionary in which the symbol is defined and the value of `name` is the name of the symbol. The optional `cdbase` attribute is a URI that can be used to disambiguate between two content dictionaries with the same name. The `cdbase` attribute may be used on any ancester element of the current `OMS`, any `OMS` elements without an explicit `cdbase` attribute which is a descendent of such an element is taken to encode an OpenMath symbol with this CD base. For example:

```
<OMS cdbase="http://www.openmath.org/cd" cd="transc1" name="sin"/>
```

is the encoding of the symbol named `sin` in the Content Dictionary named `transc1`, which is part of the collection maintained by the *OpenMath* Society.

The three attributes of the `OMS` can be used to build a URI reference for the symbol, for use in contexts where URI-based referencing mechanisms are used. This canonical URI reference is constructed as follows:

```
        URI = cdbase-value + '/' + cd-value + '#' + name-value
```

For example

```
    <OMS name="plus" cd="arith1" cdbase="http://www.openmath.org/cd"/>
```

gives the URI "`http://www.openmath.org/cd/arith1#plus`" This would allow us to refer uniquely to an openmath symbol from a MathML document [18].

```
    <mathml:csymbol xmlns:mathml="http://www.w3.org/1998/Math/MathML/"
               definitionURL="http://www.openmath.org/cd/arith1#plus">
  Z
</csymbol>
```

Note that the role attribute described in Section 3.1.4 is contained in the Content Dictionary and is not part of the encoding of a symbol, also the `cdbase` attribute need not be explicit on each `OMS` as it is inherited from any ancestor element.

Variables are encoded using the `OMV` element, with only one XML-attribute, `name`, whose value is the variable name. For instance, the encoding of the object representing the variable x is: `<OMV name="x"/>`

For an enumerated variable the index or indices of the variable are encoded as a child. So for example the encoding of the object representing the variable x_{i+1} is:

```
<OMV name="x">
  <OMA>
    <OMS cdbase="http://www.openmath.org/cd" cd="arith1" name="plus"/>
    <OMV name="i"/>
    <OMI>1</OMI>
  </OMA>
</OMV>
```

Floating-point numbers are encoded using the `OMF` element that has either the XML-attribute `dec` or the XML-attribute `hex`. The two XML-attributes cannot be present simultaneously. The value of `dec` is the floating-point number expressed in base 10, using the common syntax:

```
    (-?)([0-9]+)?("."[0-9]+)?(e(-?)[0-9]+)?.
```

The value of `hex` is the digits of the floating-point number expressed in base 16, with digits `0-9`, `A-F` (mantissa, exponent, and sign from lowest to highest bits) using a least significant byte ordering. For example, `<OMF dec="1.0e-10"/>` is a valid floating-point number.

Character strings are encoded using the `OMSTR` element. Its content is a Unicode text . Note that as always in XML the characters < and & need to be represented by the entity references `<` and `&` respectively.

Bytearrays are encoded using the `OMB` element. Its content is a sequence of characters that is a base64 encoding of the data. The base64 encoding is defined in RFC 1521 [2]. Basically, it represents an arbitrary sequence of octets using 64 "digits" (`A` through `Z`, `a` through `z`, `0` through `9`, `+` and `/`, in order of increasing value). Three octets are represented as four digits (the `=` character for padding to the right at the end of the data). All line breaks and carriage return, space, form feed and horizontal tabulation characters are ignored. The reader is refered to [2] for more detailed information.

In detail the encoding of an *OpenMath* object is described below.

Applications are encoded using the `OMA` element. The application whose head is the *OpenMath* object e_0 and whose arguments are the *OpenMath* objects $e_1, \ldots, e_n$ is encoded as `<OMA>` C_0 $C_1 \ldots C_n$ `</OMA>` where C_i is the encoding of e_i.

For example, **application**(*sin*, *x*) is encoded as:

```
<OMA>
  <OMS cdbase="http://www.openmath.org/cd" cd="transc1" name="sin"/>
  <OMV name="x"/>
</OMA>
```

provided that the symbol `sin` is defined to be a function symbol in a Content Dictionary named `transc1`.

Binding is encoded using the `OMBIND` element. The binding by the *OpenMath* object b of the *OpenMath* variables $x_1, x_2, \ldots, x_n$ in the object c is encoded as `<OMBIND>` B `<OMBVAR>` $X_1 \ldots$ X_n `</OMBVAR>` C `</OMBIND>` where B, C, and X_i are the encodings of b, c and x_i, respectively.

For instance the encoding of **binding**(*lambda*, *x*, **application**(*sin*, *x*)) is:

```
<OMBIND>
  <OMS cdbase="http://www.openmath.org/cd" cd="fns1" name="lambda"/>
  <OMBVAR><OMV name="x"/></OMBVAR>
  <OMA>
    <OMS cdbase="http://www.openmath.org/cd" cd="transc1" name="sin"/>
    <OMV name="x"/>
  </OMA>
</OMBIND>
```

Binders are defined in Content Dictionaries, in particular, the symbol `lambda` is defined in the Content Dictionary `fns1` for functions over functions.

Attributions are encoded using the `OMATTR` element. If the *OpenMath* object e is attributed with $(s_1, e_1), \ldots, (s_n, e_n)$ pairs (where s_i are the attributes), it is encoded as `<OMATTR>` `<OMATP>` S_1 $C_1 \ldots S_n$ C_n `</OMATP>` E `</OMATTR>` where S_i is the encoding of the symbol s_i, C_i of the object e_i and E is the encoding of e.

Examples are the use of attribution to decorate a group by its automorphism group:

```
<OMATTR>
  <OMATP>
    <OMS cd="groups" name="automorphism_group" />
    [..group-encoding..]
  </OMATP>
  [..group-encoding..]
</OMATTR>
```

or to express the type of a variable:

```
<OMATTR>
  <OMATP>
    <OMS cd="ecc" name="type" />
    <OMS cd="ecc" name="real" />
  </OMATP>
  <OMV name="x" />
</OMATTR>
```

A special use of attributions is to associate non-*OpenMath* data with an *OpenMath* object. This is done using the OMFOREIGN element. The children of this element must be well-formed XML. For example the attribution of the *OpenMath* object *sin(x)* with its representation in Presentation MathML is:

```
<OMATTR>
  <OMATP>
    <OMS cd="presentation1" name="mathml"/>
    <OMFOREIGN>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <mi>sin</mi><mfenced><mi>x</mi></mfenced>
      </math>
    </OMFOREIGN>
  </OMATP>
  <OMA>
   <OMS cdbase="http://www.openmath.org/cd" cd="transc1" name="sin"/>
   <OMV name="x"/>
  </OMA>
</OMATTR>
```

Of course not everything has a natural XML encoding in this way and often the contents of a OMFOREIGN will just be data or some kind of encoded string. For example the attribution of the previous object with its LaTeX representation could be achieved as follows:

```
<OMATTR>
  <OMATP>
```

```
      <OMS cd="presentation1" name="latex"/>
      <OMFOREIGN>sin\,(x)</OMFOREIGN>
    </OMATP>
    <OMA>
      <OMS cdbase="http://www.openmath.org/cd" cd="transc1" name="sin"/>
      <OMV name="x"/>
    </OMA>
</OMATTR>
```

Errors are encoded using the `OME` element. The error whose symbol is *s* and whose arguments are the *OpenMath* objects e_1, …, e_n is encoded as `<OME>` C_s $C_1 \ldots C_n$ `</OME>` where C_s is the encoding of *s* and C_i the encoding of e_i.

If an `aritherror` Content Dictionary contained a `DivisionByZero` symbol, then the object **error**(*DivisionByZero*, **application**(*divide*, *x*, 0)) would be encoded as follows:

```
<OME>
  <OMS cd="aritherror" name="DivisionByZero"/>
  <OMA>
    <OMS cd="arith1" name="divide" />
    <OMV name="x"/>
    <OMI> 0 </OMI>
  </OMA>
</OME>
```

References *OpenMath* integers, floating point numbers, character strings, byearrays, applications, binding, attributions can also be encoded as an empty `OMR` element with an `xlink:href` attribute whose value is the value of an id attribute of an *OpenMath* object of that type. The *OpenMath* element represented by this `OMR` element is a copy of the *OpenMath* element pointed to in the `xlink:xref` attribute. Note that the representation of the `OMR` element is *structurally equal*, but not identical to the element it points to.

For instance, the *OpenMath* object

application(f, **application**(f, **application**(f, a, a), **application**(f, a, a)), **application**(f, **application**(f, a, a), **applica**

can be encoded in the XML encoding as either one of the XML encodings below (and some intermedidate versions as well).

We say that an *OpenMath* element dominates all its children and all elements they dominate. An `OMR` element dominates its target, i.e. the element that carries the id attribute pointed to by the `xref` attribute. For instance in the representation in Figure Figure 4.1, the `OMA` element with id="t1" and also the second `OMR` dominate the `OMA` element with id="t11".

```
<OMOBJ>                            <OMOBJ>
  <OMA>                              <OMA xmlns:xlink="http://www.w3.org/1999/xlink">
    <OMV name="f"/>                    <OMV name="f"/>
    <OMA>                              <OMA id="t1">
      <OMV name="f"/>                    <OMV name="f"/>
      <OMA>                              <OMA id="t11">
        <OMV name="f"/>                    <OMV name="f"/>
        <OMV name="a"/>                    <OMV name="a"/>
        <OMV name="a"/>                    <OMV name="a"/>
      </OMA>                            </OMA>
      <OMA>                              <OMR xlink:href="t11"/>
        <OMV name="f"/>
        <OMV name="a"/>
        <OMV name="a"/>
      </OMA>
    </OMA>                            </OMA>
    <OMA>                              <OMR xlink:href="t1"/>
      <OMV name="f"/>
      <OMA>
        <OMV name="f"/>
        <OMV name="a"/>
        <OMV name="a"/>
      </OMA>
      <OMA>
        <OMV name="f"/>
        <OMV name="a"/>
        <OMV name="a"/>
      </OMA>
    </OMA>
  </OMA>
</OMOBJ>                           </OMOBJ>
```

Figure 4.1: Shared vs. unshared representations

4.1.2.1 An Acyclicity Constraint

The occurrences of the OMR element must obey the following global *acyclicity constraint*: An *Open-Math* element may not dominate itself.

Consider for instance the following (illegal) XML representation

```
<OMOBJ>
  <OMA id="foo">
    <OMS cdbase="http://www.openmath.org/cd" cd="arith1" name="divide"/>
    <OMI>1</OMI>
    <OMA>
       <OMS cdbase="http://www.openmath.org/cd" cd="arith1" name="plus"/>
       <OMI>1</OMI>
       <OMR xref="foo"/>
    </OMA>
  </OMA>
</OMOBJ>
```

Here, the OMA element with id="foo" dominates its third child, which dominates the OMR element, which dominates its target: the element with id="foo". So by transitivity, this element dominates itself, and by the acyclicity constraint, it is not the XML representation of an *OpenMath* element. Even though it could be given the interpretation of the continued fraction

$$\cfrac{1}{1+\cfrac{1}{1+\frac{1}{\cdots}}}$$

this would correspond to an infinite tree of applications, which is not admitted by the structure of *OpenMath* objects described in Chapter 3.

Note that the acyclicity constraints is not restricted to such simple cases, as the following example shows.

```
<OMOBJ>                                <OMOBJ>
  <OMA id="bar">                         <OMA id="baz">
    <OMS cd="arith1" name="plus"/>         <OMS cd="arith1" name="plus"/>
    <OMI>1</OMI>                            <OMI>1</OMI>
    <OMR xref="baz"/>                      <OMR xref="bar"/>
  </OMA>                                 </OMA>
</OMOBJ>                                </OMOBJ>
```

Here, the OMA with id="bar" dominates its third child, the OMR with xref="baz", which dominates its target OMA with id="baz", which in turn dominates its third child, the OMR with xref="bar", this finally dominates its target, the original OMA element with id="bar". So this pair of *OpenMath* objects violates the acyclicity constraint and is not the XML representation of an *OpenMath* object.

4.1.2.2 Sharing and Bound Variables

Note that the OMR element is a *syntactic* referencing mechanism: an OMR element stands for the exact XML element it points to. In particular, referencing does not interact with binding in a semantically intuitive way, since it allows for variable capture. Consider for instance the following XML representation:

```
<OMBIND id="outer">
  <OMS cdbase="http://www.openmath.org/cd" cd="fns1" name="lambda"/>
  <OMBVAR><OMV name="X"/></OMBVAR>
  <OMA>
    <OMV name="f"/>
    <OMBIND id="inner">
      <OMS cdbase="http://www.openmath.org/cd" cd="fns1" name="lambda"/>
      <OMBVAR><OMV name="X"/></OMBVAR>
      <OMR id="copy" xlink:href="orig"/>
    </OMBIND>
    <OMA id="orig"><OMV name="g"/><OMV name="X"/></OMA>
  </OMA>
</OMBIND>
```

it represents the *OpenMath* object

$$\textbf{binding}(\lambda, X, \textbf{application}(f, \textbf{binding}(\lambda, X, \textbf{application}(g, X)), \textbf{application}(g, X))))$$

which has two subterms of the form **application**(g, X), one with id="orig" (the one explicitly represented) and one with id="copy", represented by the OMR element. In the original, the variable *X* is bound by the *outer* OMBIND element, and in the copy, the variable *X* is bound by the *inner* OMBIND element. We say that the inner OMBIND has captured the variable *X*.

It is well-known that variable capture does not conserve semantics. For instance, we could use α-conversion to rename the inner occurrence of *x* into - say - *y* arriving at the (same) object

$$\textbf{binding}(\lambda, X, \textbf{application}(f, \textbf{binding}(\lambda, Y, \textbf{application}(g, Y)), \textbf{application}(g, X))))$$

Using references that capture variables in this way can easily lead to representation errors, and is not recommended.

4.1.3 Embedding *OpenMath* in XML Documents

The above encoding of XML encoded *OpenMath* specifies the grammar to be used in files that encode a single *OpenMath* object, and specifies the character streams that a conforming *OpenMath* application should be able to accept or produce.

When embedding XML encoded *OpenMath* objects into a larger XML document one may wish, or need, to use other XML features. For example use of extra XML attributes to specify XML Namespaces [16] or xml:lang attributes to specify the language used in strings [17].

If such XML features are used then the XML application controlling the document must, if passing the *OpenMath* fragment to an *OpenMath* application, remove any such extra attributes and must ensure that the fragment is encoded according to the grammar specified above.

4.2 The Binary Encoding

The binary encoding was essentially designed to be more compact than the XML encodings, so that it can be more efficient if large amounts of data are involved. For the current encoding, we tried to keep the right balance between compactness, speed of encoding and decoding and simplicity (to allow a simple specification and easy implementations).

4.2.1 A Grammar for the Binary Encoding

Figure 4.2: Grammar of the binary encoding of *OpenMath* objects.

Figure Figure 4.2 gives a grammar for the binary encoding ("start" is the start symbol)..

The following conventions are used in this section: [*n*] denotes a byte whose value is the integer *n* (*n* can range from 0 to 255), {*m*} denotes four bytes representing the (unsigned) integer *m* in network byte order, [_] denotes an arbitrary byte, {_} denotes an arbitrary sequence of four bytes.

xxxx:*n*, where *xxxx* is one of *symbname*, *cdname*, *varname*, *uri*, *id*, *digits*, or *bytes* denotes a sequence of *n* bytes that conforms to the constraints on *xxxx* strings. For instance, for *symbname*, *varname*, or *cdname*this is the regular expression described in Section 3.3, for *uri* it is the grammar for URIs in [9].

4.2.2 Description of the Grammar

An *OpenMath* object is encoded as a sequence of bytes starting with the begin object tag (values 24 and 88) and ending with the end object tag (value 25). These are similar to the <OMOBJ> and </OMOBJ> tags of the XML encoding.

The encoding of each kind of *OpenMath* object begins with a tag that is a single byte, holding a *token identifier* that describes the kind of object and two flags, the *long* flag and the *shared* flag. The identifier is stored in the first 6 bits (1 to 6). The long flag is the eighth bit and the shared flag is the seventh bit. If the long flag is set, this signifies that the names, strings, and data fields in the encoded *OpenMath* object are longer than 255 byte or characters. The sharing flag indicates that the encoded object is shared in another (part of an) object somewhere else (see Section 4.2.3).

Note that if the sharing flag is set (in the right column of the grammar in Figure Figure 4.2, then the encoding includes a representation of the identifier.

To facilitate the stremaing *OpenMath* objects, some basic objects (integers, strings, bytearrays, and foreign objects) have variant token identifiers with the fifth bit set. The idea behind this is that these basic objects can be split into packets. If the fifth bit is not set, this packet is the final packet of the basic object. If the bit is set, then more packets of the basic object will follow directly after this one. Note that all packets making up a basic object must have the same token identifier (up to the fifth bit).

Here is a description of the binary encodings of every kind of *OpenMath* object:

Integers are encoded depending on how large they are. There are four possible formats. Integers between -128 and 127 are encoded as the small integer tags (token identifier 1) followed by a single byte that is the value of the integer (interpreted as a signed character). For example 16 is encoded as `0x01 0x10`. Integers between -2^{31} (-2147483648) and $2^{31} - 1$ (2147483647) are encoded as the small integer tag with the long flag set followed by the integer encoded in little endian format on four bytes (network byte order: the most significant byte comes first). For example, 128 is encoded as `0x81 0x00000080`. The most general encoding begins with the big integer tag (token identifier 2) with the long flag set if the number of bytes in the encoding of the digits is greater or equal than 256. It is followed by the length (in bytes) of the sequence of digits, encoded on one byte (0 to 255, if the long flag was not set) or four bytes (network byte order, if the long flag was set). It is then followed by a byte describing the sign and the base. This 'sign/base' byte is + (0x2B) or – (0x2D) for the sign ored with the base mask bits that can be 0 for base 10 or 0x40 for base 16. It is followed by the strings of digits (as characters) in their natural order (as in the XML encoding). For example, 8589934592 (2^{33}) is encoded `0x02 0x0A 0x2B 0x3835383939393334353932` and xffffffff1 is encoded as `0x02 0x08 0x6b 0x6666666666666631`. Note that it is permitted to encode a "small" integer in any "bigger" format.

To splice sequences of integer packets into integers, we have to consider three cases: In the case of token identifiers 1, 33, and 65 the sequence of packets is treated as a sequence of integer digits to the base of 2^7 (most significant first). The case of token identifiers 129, 161, and 193 is analogous with digits of base 2^{31}. In the case of token identifiers 2, 34, 66, 130, 162, and 194 the integer is assembled by concatenating the string of decimal digits in the packets in sequence order (which corresponds to most significant first). Note that in all cases only the sequence-initial packet may contain a signed integer. The sign of this packet determines the sign of the overall integer.

Symbols are encoded as the symbol tags (token identifier 8) with the long flag set if the maximum of the length in bytes in the UTF-8 encoding of the Content Dictionary name, the symbol name or the CD base is greater than or equal to 256 . The symbol tags [8] and [8+128] are deprecated in *OpenMath*2 since symbols now have an optional `cdbase` field, but are kept for backwards compatibility. The symbol tag is followed by the length in bytes in the UTF-8 encoding of the Content Dictionary name, the symbol name, and the CD base as a byte (if the long flag was not set) or a four byte integers (in network byte order). These are followed

by the

bytes of the UTF-8 encoding of the Content Dictionary name, the symbol name, and the CD base.

Variables are encoded using the variable tags (token identifiers 5) with the long flag set if the number of bytes in the UTF-8 encoding of the variable name is greater than or equal to 256 (this should never happen if the rules on variables are followed). Then, there is the number of characters as a byte (if the long flag was not set) or a four byte integer (in network byte order), followed by the characters of the name of the variable. For example, the variable x is encoded as `0x05 0x01 0x78`.

Indexed Variables are encoded using the indexed variable tags (token identifiers 10 and 11) as begin and end tags. The long flag set on the begin tag if the number of bytes (characters) in the variable name is greater than or equal to 256. The start tag is followed by the number of bytes as a byte (if the long flag was not set) or a four byte integer (in network byte order), followed by the bytes of the UTF-8 encoding of the name of the variable. This is followed by the encoding of an *OpenMath* object and the end tag [11].

Floating-point number are encoded using the floating-point number tags (token identifier 3) followed by eight bytes that are the IEEE 754 representation [8], most significant bytes first. For example, 0.1 is encoded as `0x03 0x000000000000f03f`.

Character string are encoded in two ways depending on whether , the string is encoded in UTF-16 or ISO-859-1 (LATIN-1). In the case of LATIN-1 it is encoded as the one byte character string tags (token identifier 6) with the long flag set if the number of bytes (characters) in the string is greater than or equal to 256. Then, there is the number of characters as a byte (if the length flag was not set) or a four byte integer (in network byte order), followed by the characters in the string. If the string is encoded in UTF-16, it is encoded as the two byte character string tags (token identifier 7) with the long flag set if the number of characters in the string is greater or equal to 256. Then, there is the number of UTF-16 units, which will be the number of characters unless characters in the higher planes of Unicode are used, as a byte (if the long flag was not set) or a four byte integer (in network byte order), followed by the characters (UTF-16 encoded Unicode).

Sequences of string packets are assembled into strings by concatenating the strings in the packets in sequence order.

Bytearrays are encoded using the bytearray tags (token identifier 4) with the long flag set if the number elements is greater than or equal to 256. Then, there is the number of elements, as a byte (if the long flag was not set) or a four byte integer (in network byte order), followed by the elements of the arrays in their normal order.

Sequences of bytearray packets are assembled into byte arrays by concatenating the bytearrays in the packets in sequence order.

Foreign Objects are encoded using the foreign object tags (token identifier 12) with the long flag set if the number of bytes is greater than or equal to 256. Then, there is the number of elements, as a byte (if the long flag was not set) or a four byte integer (in network byte order), followed by the elements of the arrays in their normal order.

Sequences of foreign object packets are assembled into foreign objects by concatenating the bytes in the packets in sequence order.

Applications are encoded using the application tags (token identifiers 16 and 17). More precisely, the application of E_0 to $E_1 \ldots E_n$ is encoded using the application tags (token identifier 16), the sequence of the encodings of E_0 to E_n and the end application tags (token identifier 17).

Bindings are encoded using the binding tags (token identifiers 26 and 27). More precisely, the binding by B of variables $V_1 \ldots V_n$ in C is encoded as the binding tags (token identifier 26), followed by the encoding of B, followed by the binding variables tags (token identifier 28), followed by the encodings of the variables $V_1 \ldots V_n$, followed by the end binding variables tags (token identifier 29), followed by the encoding of C, followed by the end binding tags (token identifier 27).

Attribution are encoded using the attribution tags (token identifiers 18 and 19). More precisely, attribution of the object E with (S_1, E_1), $\ldots$ (S_n, E_n) pairs (where S_i are the attributes) is encoded as the attributed object tags (token identifier 18), followed by the encoding of the attribute pairs as the attribute pairs tags (token identifier 20), followed by the encoding of each symbol and value, followed by the end attribute pairs tags (token identifier 21), followed by the encoding of E, followed by the end attributed object tags (token identifier 19).

Error are encoded using the error tags (token identifiers 22 and 23). More precisely, S_0 applied to $E_1 \ldots E_n$ is encoded as the error tags (token identifier 22), the encoding of S_0, the sequence of the encodings of E_0 to E_n and the end error tags (token identifier 23).

Internal References are encoded using the internal reference tags [30] and [30+128] (the sharing flag cannot be set on this tag, since chains of references are not allowed in the *OpenMath* binary encoding.) with long flag set if the number of *OpenMath* sub-objects in the encoded *OpenMath* is greater than or equal to 256. Then, there is the ordinal number of the referenced *OpenMath* object as a byte (if the long flag was not set) or a four byte integer (in network byte order).

External References are encoded using the external reference tags [31] and [31+128] (the sharing flag cannot be set on this tag, since chains of references are not allowed in the *OpenMath* binary encoding) with the long flag set if the number of bytes in the reference URI is greater than or equal to 256. Then, there is the number of bytes in the URI used for the external reference as a byte (if the long flag was not set) or a four byte integer (in network byte order), followed by the URI.

4.2.2.1 Sharing in Objects beginning with the identifier [24]

This binary encoding supports the sharing of symbols, variables and strings (up to a certain length for strings) within one object. That is, sharing between objects is not supported. A reference to a shared symbol, variable or string is encoded as the corresponding tag with the long flag not set and the shared flag set, followed by a positive integer n coded on one byte (0 to 255). This integer references the $n+1$-th such sharable sub-object (symbol, variable or string up to 255 characters) in the current *OpenMath* object (counted in the order they are generated by the encoding). For example, `0x48 0x01` references a symbol that is identical to the second symbol that was found in

the current object. Strings with 8 bit characters and strings with 16 bit characters are two different kinds of objects for this sharing. Only strings containing less than 256 characters can be shared (i.e. only strings up to 255 characters).

4.2.3   Sharing with References (beginning with [24+64])

In the binary encoding specified in the last section (which we keep for compatibility reasons, but deprecate in favor of the more efficient binary encoding specified in this section) only symbols, variables, and short strings could be shared. In this section, we will present a second binary encoding, which shares most of the identifiers with the one in the last one, but handles sharing differently. This encoding is signaled by the shared object tags [88].

The main difference is the interpretation of the sharing flag (bit 7), which can be set on all objects that allow it. Instead of encoding a reference to a previous occurrence of an object of the same type, it indicates whether an object will be referenced later in the encoding. This corresponds to the information, whether an id attribute is set in the XML encoding. On the object identifier (where sharing does not make sense), the shared flag signifies the encoding described here ([88]=[24+64]).

Otherwise integers, floats, variables, symbols, strings, bytearrays, and constructs are treated exactly as in the binary encoding described in the last section.

The binary encoding with references uses the additional reference tags [30] for (short) internal references, [30+128] for long internal references, [31] for (short) external references, [31+128] for long external references. Internal references are used to share sub-objects in the encoded object (see Figure 4.3 for an example) by referencing their position; external references allow to reference *OpenMath* objects in other documents by a URI.

Identifiers [30+64] and [30+64+128] are not used, since they would encode references that are shared themselves. Chains of references are redundant, and decrease both space and time efficiency, therefore they are not allowed in the *OpenMath* binary encoding.

References consist of the identifier [30] ([30+128] for long references) followed by a positive integer n coded on one byte (4 bytes for long references). This integer references the $n + 1$th shared sub-object (one where the shared flag is set) in the current object (counted in the order they are generated in the encoding). For example `0x7E 0x01` references the second shared sub-object. Figure Figure 4.3 shows the binary encoding of the object in figure Figure 4.1 above.

It is easy to see that in this binary encoding, the size of the encoding is $13 + 7(d - 1)$ bytes, where d is the depth of the tree, while a totally unshared encoding is $8 * 2^d - 8$ bytes (sharing variables saves up to 256 bytes for trees up to depth 8 and wastes space for greater depths). The shared XML encoding only uses $32d + 29$ bytes, which is more space efficient starting at depth 9.

Note that in conversion to binary encoding the identifiers on the objects are not preserved. Morever, even though the XML encoding allows references across objects, as in figure ???, the binary encoding does not (the binary encoding has no notion of a multi-object collection, which is implicit by embedding *OpenMath* objects into XML documents in the XML encoding.).

Hex	Meaning	Hex	Meaning
58	begin object tag	05	variable tag
10	begin application tag	01	variable length
05	variable tag	61	a (variable name)
01	variable length	05	variable tag
66	f (variable name)	01	variable length
50	begin application tag (shared)	61	a (variable name)
05	variable tag	11	end application tag
01	variable length	1E	short reference
66	f (variable name)	00	to the first shared object
50	begin application tag (shared)	11	end application tag
05	variable tag	1E	short reference
01	variable length	00	to the second shared object
66	f (variable name)	11	end application tag
		19	end object tag

Figure 4.3: A binary encoding of the *OpenMath* object from figure Figure 4.1.

Note that objects need not be fully shared (or shared at all) in the binary encoding with sharing.

4.2.4 Implementation Note

A typical implementation of the binary encoding comes in two parts. The first part deals with the unshared encodings, i.e. objects starting with the identifier `[24]`.

This part uses four tables, each of 256 entries, for symbol, variables, 8 bit character strings whose lengths are less than 256 characters and 16 bit character strings whose lengths are less than 256 characters. When an object is read, all the tables are first flushed. Each time a sharable sub-object is read, it is entered in the corresponding table if it is not full. When a reference to the shared *i*-th object of a given type is read, it stands for the *i*-th entry in the corresponding table. It is an encoding error if the *i*-th position in the table has not already been assigned (i.e. forward references are not allowed). Sharing is not mandatory, there may be duplicate entries in the tables (if the application that wrote the object chose not to share optimally).

The part for the shared representations of *OpenMath* objects uses an unbounded array for storing shared sub-objects. Whenever an object has the shared flag set, then it is read and a pointer to the generated data structure is stored at the next position of the array. Whenever a reference of the form `[30]` `[_]` is encountered, the array is queried for the value at `[_]` and analogously for `[30+128]` `{_}`. Note that the application can decide to copy the value or share it among subterms as long as it respects the identity conditions given by the tree-nature of the *OpenMath* objects. The implementation must take care to ensure that no variables are captured during this process (see section Section 4.1.2.2), and possibly have methods for recovering from cyclic dependency relations (this can be done by standard loop-checking methods).

Writing an object is simple. The tables are first flushed. Each time a sharable sub-object is encountered (in the natural order of output given by the encoding), it is either entered in the corresponding table (if it is not full) and output in the normal way or replaced by the right reference if it is already present in the table.

4.2.5 Example of Binary Encoding

As an example of this binary encoding, we can consider the *OpenMath* object whose XML encoding is

```
<OMOBJ>
  <OMA>
    <OMS cdbase="http://www.openmath.org/cd" name="times" cd="arith1"/>
    <OMA>
      <OMS cdbase="http://www.openmath.org/cd" name="plus" cd="arith1"/>
      <OMV name="x"/>
      <OMV name="y"/>
    </OMA>
    <OMA>
      <OMS cdbase="http://www.openmath.org/cd" name="plus" cd="arith1"/>
      <OMV name="x"/>
      <OMV name="z"/>
    </OMA>
  </OMA>
</OMOBJ>
```

It is binary encoded as the sequence of bytes given by the following table.

Hex	Meaning	Hex	Meaning
18	begin object tag	68	h .
10	begin application tag	31	1 .)
08	symbol tag	70	p (symbol name begin
06	cd length	6c	l .
05	name length	75	u .
61	a (cd name begin	73	s .)
72	r .	05	variable tag
69	i .	01	name length
74	t .	78	x (name)
68	h .	05	variable tag
31	1 .)	01	name length
74	t (symbol name begin	79	y (variable name)
69	i .	11	end application tag
6d	m .	10	begin application tag
65	e .	48	symbol tag (with share bit on)
73	s .)	01	reference to second symbol seen (arith1:plus)
10	begin application tag	45	variable tag (with share bit on)
08	symbol tag	00	reference to first variable seen (x)
06	cd length	05	variable tag
04	name length	01	name length
61	a (cd name begin	7a	z (variable name)
72	r .	11	end application tag
69	i .	11	end application tag
74	t .	19	end object tag

4.2.6 Relation to the *OpenMath*1 binary encoding

The *OpenMath*2 binary encoding significantly extends the *OpenMath*1 binary encoding to acco-modate the new features and in particular sharing of sub-objects. The tags and structure of the *OpenMath*1 binary encoding are still present in the current *OpenMath* binary encoding, so that binary encoded *OpenMath*1 objects are still valid in the *OpenMath*2 binary encoding and correspond to the same abstract *OpenMath* objects. In some cases, the binary encoding tags without the shared flag can still be used as more compact representations of the objects (which are not shared, and do not have an identifier).

As the binary encoding is geared towards compactness, *OpenMath* objects should be maximally internally shared (if computationally feasible). Note that since sharing is done only at the encoding level, this does not alter the meaning of an *OpenMath* object, only allow to represent it more compactly.

4.3   Summary

The key points of this chapter are:

- The XML encoding for *OpenMath* objects uses most common character sets.
- The XML encoding is readable, writable and can be embedded in most documents and transport protocols.
- The binary encoding for *OpenMath* objects should be used when efficiency is a key issue. It is compact yet simple enough to allow fast encoding and decoding of objects.

Chapter 5

Content Dictionaries

In this chapter we give a brief overview of Content Dictionaries before explicitly stating their functionality and encoding.

5.1 Introduction

Content Dictionaries (CDs) are central to the *OpenMath* philosophy of transmitting mathematical information. It is the *OpenMath* Content Dictionaries which actually hold the meanings of the objects being transmitted.

For example if application *A* is talking to application *B*, and sends, say, an equation involving multiplication of matrices, then *A* and *B* must agree on what a matrix is, and on what matrix multiplication is, and even on what constitutes an equation. All this information is held within some Content Dictionaries which both applications agree upon.

A *Content Dictionary* holds the meanings of (various) mathematical "words". These words are *OpenMath* basic objects referred to as *symbols* in Section 3.1.

With a set of symbol definitions (perhaps from several Content Dictionaries), *A* and *B* can now talk in a common "language".

It is important to stress that it is not Content Dictionaries themselves which are being transmitted, but some "mathematics" whose definitions are held within the Content Dictionaries. This means that the applications must have already agreed on a set of Content Dictionaries which they "understand" (i.e., can cope with to some degree).

In many cases, the Content Dictionaries that an application understands will be constant, and be intrinsic to the application's mathematical use. However the above approach can also be used for applications which can handle every Content Dictionary (such as an *OpenMath* parser, or perhaps a typesetting system), or alternatively for applications which understand a changeable number of Content Dictionaries (perhaps after being sent Content Dictionaries in some way).

The primary use of Content Dictionaries is thought to be for designers of Phrasebooks,the programs which translate between the *OpenMath* mathematical object and the corresponding (often internal) structure of the particular application in question. For such a use the Content Dictionaries have themselves been designed to be as readable and precise as possible.

Another possible use for *OpenMath* Content Dictionaries could rely on their automatic comprehension by a machine (e.g., when given definitions of objects defined in terms of previously understood ones), in which case Content Dictionaries may have to be passed as data. Towards this end, a Content Dictionary has been written which contains a set of symbols sufficient to represent any other Content Dictionary. This means that Content Dictionaries may be passed in the same way as other (*OpenMath*) mathematical data.

Finally, the syntax of the reference encoding for Content Dictionaries has been designed to be relatively easy to learn and to write, and also free from the need for any specialist software. This is because it is acknowledged that there is an enormous amount of mathematical information to represent, and so most of the Content Dictionaries will be written by "ordinary" mathematicians, encoding their particular fields of expertise. A further reason is that the mathematics conveyed by a specific Content Dictionary should be understandable independently of any application.

The key points from this section are:

- Content Dictionaries should be readable and precise to help Phrasebook designers,
- Content Dictionaries should be readily write-able to encourage widespread use,
- It ought to be possible for a machine to understand a Content Dictionary to some degree.

5.2   Abstract Content Dictionaries

In this section we define the abstract structure of Content Dictionaries.

A Content Dictionary consists of the following mandatory pieces of information:

1. A *name* corresponding to the rules described in Section 3.3.
2. A *description* of the Content Dictionary.
3. A *revision date*, the date of the last change to the Content Dictionary. Dates should be stored in the ISO-compliant format YYYY-MM-DD, e.g. 1966-02-03.
4. A *review date*, a date until which the content dictionary is guarenteed to remain unchanged.
5. A *version number* which consists of a major and minor part (see Section 5.2.2).
6. A *status*, as described in Section 5.2.1.
7. A *CD base* which is a unique identifier for the Content Dictionary, and may or may not refer to an actual location from which it can be retrieved.
8. A series of *symbol definitions* as described below.

A symbol definition consists of the following pieces of information:

1. A mandatory *name* corresponding to the rules described in Section 3.3.
2. A mandatory *description* of the symbol, which can be as formal or informal as the author likes.
3. An optional *role* as described in Section 3.1.4.
4. Zero or more *commented mathematical properties* which are mathematical properties of the symbol expressed in a mechanism other than *OpenMath*.
5. Zero or more *formal mathematical properties* which are mathematical properties of the symbol expressed in *OpenMath*. Note that it is common for commented and formal mathematical properties to be introduced in pairs, with the former describing the latter.
6. Zero or more *examples* which are intended to demonstrate the use of the symbol within an *OpenMath* object.

Some pieces of information which might logically be thought to be part of a Content Dictionary, such as the types or signatures of symbols, are better represented externally. This allows for new variants to be associated with Content Dictionaries without the Dictionaries themselves being changed. A model for signatures is given in Section 5.4.1.

Content Dictionaries may be grouped into *CD Groups*. These groups allow applications to easily refer to collections of Content Dictionaries. One particular CDGroup of interest is the "MathML CDGroup". This group expresses the collection of the core Content Dictionaries that is designed to have the same semantic scope as the content elements of MathML 2 [18]. *OpenMath* objects built from symbols that come from Content Dictionaries in this CDGroup may be expected to be eaily transformed between *OpenMath* and MathML encodings. The detailed structure of a CDGroup is described in Section 5.4.2 below.

5.2.1 Content Dictionary Status

The status of a Content Dictionary can be either

- `official`: approved by the *OpenMath* Society according to the procedure outlined in Section 5.5;
- `experimental`: under development and thus liable to change;
- `private`: used by a private group of *OpenMath* users;
- `obsolete`: an obsolete Content Dictionary kept only for archival purposes.

5.2.2 Content Dictionary Version Numbers

A version number must consist of two parts, a major version and a revision, both of which should be non-negative integers. In CDs that do not have status *experimental*, the version number should be a positive integer.

Unless a CD has status *experimental*, no changes should ever be introduced that invalidate objects built with previous versions. Any change that influences phrasebook compliance, like adding a new symbol to a Content Dictionary, is considered a major change and should be reflected by an increase in the major version number. Other changes, like adding an example or correcting a description, are considered minor changes. For minor changes the version number is not changed, but an increase should be made to the revision number. Note that a change such as removing a symbol should not be made unless the CD has status *experimental*. Should this be required then a new CD with a different name should be produced so as not to invalidate existing objects.

When the major version number is increased, the revision number is normally reset to zero.

As detailed in chapter Chapter 6, *OpenMath* compliant applications state which versions of which CDs they support.

5.3 The Reference Encoding for Content Dictionaries

The reference encoding of Content Dictionaries are as XML documents. A valid Content Dictionary document should conform to the Relax NG Schema for Content Dictionaries given in Section 5.3.1.

An example of a complete Content Dictionary is given in Appendix Appendix A.1, which is the Meta Content Dictionary for describing Content Dictionaries themselves. A more typical Content Dictionary is given in Appendix Appendix A.2, the arith1 Content Dictionary for basic arithmetic functions.

5.3.1 The RelaxNG Schema for Content Dictionaries

```
# **********************************************
#
# Relax NG Schema for OpenMath CD
#
# **********************************************

default namespace = "http://www.openmath.org/OpenMathCD"


include "openmath2.rnc" {start = CD}
```

```
CDComment = element CDComment { text }
CDName = element CDName { text }
CDUses = element CDUses { CDName* }
CDURL = element CDURL { text }
CDBASE = element CDBASE { text }
text-or-om = (text | OMOBJ)*
CDReviewDate = element CDReviewDate { text }
CDDate = element CDDate { text }
CDVersion = element CDVersion { text }
CDRevision = element CDRevision { text }
CDStatus = element CDStatus { text }
Description = element Description { text }
Name = element Name { text }
Role = element Role { text }
CMP = element CMP { text }
FMP = element FMP { text-or-om }
# allow embedded OM
Example = element Example { text-or-om }
CDDefinition =
  element CDDefinition {
    (CDComment | Name | Description | Example | FMP | CMP)*
  }
CD =
  element CD {
    CDComment*,
    (CDName & CDURL &
     CDReviewDate & CDDate & CDStatus &
     CDUses &
     CDVersion & CDRevision),
    (CDComment | Description | CDDefinition | Example)*
  }
```

5.3.2 Further Description of the CD Schema

We now describe the elements used in the above schema in terms of the abstract description of CDs in Section 5.2. Unless stated otherwise the information is encoded as the content of the element.

CDName The name of the Content Dictionary.

Description The text occurring in the `Description` element is used to give a description of the enclosing element, which could be a symbol or the entire Content Dictionary. The content of this element can be any XML text.

CDReviewDate The review date of the Content Dictionary.

CDDate The revision date of this version of the Content Dictionary.

CDVersion The major version number of the CD.

CDRevision The minor version number of the CD.

CDStatus The status of the Content Dictionary.

CDBASE The CD base of the CD.

CDURL The text occurring in the `CDURL` element should be a valid URL where the source file for the Content Dictionary encoding can be found (if it exists). The filename should conform to ISO 9660 [13].

CDUses The content of this element should be a series of `CDName` elements, each naming a Content Dictionary used in the `Example` and `FMP`s of the current Content Dictionary. This element is optional and deprecated since the information can easily be extracted automatically.

CDComment The content of this element should be text that does not convey any crucial information concerning the current Content Dictionary. It can be used in the Content Dictionary header to report the author of the Content Dictionary and to log change information. In the body of the Content Dictionary, it can be used to attach extra remarks to certain symbols.

CDDefinition The element which contains the definition of an individual symbol.

Name The name of a symbol.

Example The text occurring in the `Example` element is used to give examples of the enclosing symbol, and can be any XML text. In addition to text the element may contain examples as XML encoded *OpenMath*, inside `OMOBJ` elements. Note that `Examples` must be with respect to some symbol and cannot be "loose" in the Content Dictionary.

CMP A Commented Mathematical Property.

FMP A Formal Mathematical Property.

5.4 Additional Information

Content Dictionaries contain just one part of the information that can be associated to a symbol in order to define its meaning and its functionality. *OpenMath* Signature dictionaries, CDGroups, and possibly collections of extra mathematical properties, are used to convey the different aspects that as a whole make up a mathematical definition.

5.4.1 Signature Dictionaries

OpenMath may be used with any type system. One just needs to produce a Content Dictionary which gives the constructors of the type system, and then one may build *OpenMath* objects representing types in the given type system. These are typically associated with *OpenMath* objects via the *OpenMath* **attribution** constructor.

A Small Type System, called STS, has been designed to give semi-formal signatures to *OpenMath* symbols and is documented in [6]. The signature file given in Appendix A.3 is based on this formalism. Using the same mechanism, [4] shows how pure type systems can also be employed to assign types to *OpenMath* symbols.

5.4.1.1 Abstract Specification of a Signature File

Signature files have a header which specifies the type system being used, and the Content Dictionary which contains the symbols for which the signatures are being given. Each signature takes the form of an *OpenMath* object in an appropriate encoding.

CDSignatures The outermost element of the Signature File is characterized by two required attributes that identify the type system and the Content Dictionary whose signatures are defined. The value of the XML attribute `type` is the name of the Content Dictionary or of the CDGroup (cfg. Section 5.4.2) that represents the type system. The value of the XML attribute `cd` is the name of the Content Dictionary whose symbols are assigned signatures in this Signature File. Both values are of the form specified in Chapter 4.

CDSComment See `CDComment` in Section 5.3.3.

CDSreviewDate The text occurring in the `CDSReviewDate` element corresponds to the earliest possible revision date of the Signature File. The date formats should be ISO-compliant in the form YYYY-MM-DD, e.g. 2000-02-29.

CDSStatus The text occurring in the `CDSStatus` element corresponds to the status of the Signature File, and can be either `official` (approved by the *OpenMath* Society according to the procedure outlined in Section 5.5), `experimental` (currently being tested), `private` (used by a private group of *OpenMath* users) or `obsolete` (an obsolete Signature File kept only for archival purposes).

Signature The content of the `Signature` element has to be a valid *OpenMath* object in XML encoding as specified in Chapter 4. Additionally, the object must represent a valid type in the type system identified by the XML attribute `type` of the `CDSignature` element. See Section 5.4.1.4 for examples.

1. A *type definition*: the name of the Content Dictionary or of the CDGroup (cfg. Section 5.4.2) that represents the type system being used.
2. A *CD name*: the name of the CD for which signatures are being defined.
3. A *review date* as defined in Section 5.2.

4. A *status*: as defined in Section 5.2.

5. A series of *signatures* which are *OpenMath* objects in some encoding. The objects must represent types as defined by the type definition.

5.4.1.2 A RelaxNG Schema for a Signature File

The following is a reference encoding of a signature dictionary, designed to be used with Content Dictionaries in the XML encoding.

```
# **********************************************
#
# Relax NG Schema for OpenMath CD Signatures
#
# **********************************************

default namespace = "http://www.openmath.org/OpenMathCDS"

include "openmath2.rnc"
CDSComment = element CDSComment { text }
CDSReviewDate = element CDSReviewDate { text }
CDSStatus = element CDSStatus { text }
CDSignatures =
  element CDSignatures {
    attlist.CDSignatures,
    (CDComment | CDSComment)*,
    (CDSReviewDate? & CDSStatus?),
    (CDComment | CDSComment | Signature)*
  }
attlist.CDSignatures =
  attribute cd { text },
  attribute type { text }
Signature = element Signature { attlist.Signature, OMOBJ? }
attlist.Signature = attribute name { text }
start = CDSignatures
```

5.4.1.3 Examples

An example of a signature file for the type system STS and the `arith1` Content Dictionary is given in Appendix A.3. Each signature entry is similar to the following one for the *OpenMath* symbol `<OMS cd="arith1" name="plus"/>`:

```
<Signature name="plus">
<OMOBJ>
 <OMA>
  <OMS cdbase="http://www.openmath.org/cd" name="mapsto" cd="sts"/>
  <OMA>
   <OMS cdbase="http://www.openmath.org/cd" name="nassoc" cd="sts"/>
   <OMV name="AbelianSemiGroup"/>
  </OMA>
  <OMV name="AbelianSemiGroup"/>
 </OMA>
</OMOBJ>
</Signature>
```

5.4.2 CDGroups

The CD Group mechanism is a convenience mechanism for identifying collections of CDs. A CD Group file is an XML document used in the (static or dynamic) negotiation phase where communicating applications declare and agree on the Content Dictionaries which they process. It is a complement, or an alternative, to the individual declaration of Content Dictionaries understood by an application. Note that CD Groups do *not* affect the *OpenMath* objects themselves. Symbols in an object always refer to content dictionaries, not groups.

For an application to declare that it "understands CDGroup G" is exactly equivalent to, and interchangable with, the declaration that it "understands Content Dictionaries x_1, x_2, ... x_n", where x_1, ... x_n are the members of CDGroup G.

5.4.2.1 The Specification of CDGroups

CDGroups are XML documents, hence a valid CDGroup should

- be valid according to the schema given in Figure 5.3,
- adhere to the extra conditions on the content of the elements given in Section 5.4.2.2.

Apart from some header information such as CDGroupName and CDGroup version, a CDGroup is simply an unordered list of CDs, identified by name and optionally version number and URL.

5.4.2.2 Further Requirements of a CDGroup

The notion of being a valid CDGroup implies that the following requirements on the content of the elements described by the schema given in Section 5.4.1.3 are also met.

CDGroup The XML element CDGroup is the outermost element in a CDGroup document.

```
# Schema for OpenMath CD groups

# info on the CD group itself

CDGroupName = element CDGroupName { text }
CDGroupVersion = element CDGroupVersion { text }
attlist.CDGroupVersion &= empty
CDGroupRevision = element CDGroupRevision { text }
CDGroupURL = element CDGroupURL { text }
CDGroupDescription = element CDGroupDescription { text }
# info on the CDs in the group
CDComment = element CDComment { text }
CDGroupMember =
  element CDGroupMember {
    attlist.CDGroupMember, CDComment?, CDName, CDVersion?, CDURL?
  }
CDName = element CDName { text }
CDVersion = element CDVersion { text }
CDURL = element CDURL { text }
# structure of the group
CDGroup =
  element CDGroup {
    CDGroupName,
    CDGroupVersion,
    CDGroupRevision?,
    CDGroupURL,
    CDGroupDescription,
    (CDGroupMember | CDComment)*
  }
start = CDGroup
```

Figure 5.1: Relax NG Specification of CDGroups

CDGroupName The text occurring in the `CDGroupName` element corresponds to the name of the CDGroup. For the syntactical requirements, see `CDName` in Section 5.3.3.

CDGroupVersion

CDGroupRevision The text occurring in these elements contains the major and minor version numbers of the CDGroup.

CDGroupURL The text occurring in the `CDGroupURL` element identifies the location of the CD-Group file, not necessarily of the member Content Dictionaries. For the syntactical requirements, see `CDURL` in Section 5.3.3.

CDGroupDescription The text occurring in the `CDGroupDescription` element describes the mathematical area of the CDGroup.

CDGroupMember The XML element `CDGroupMember` encloses the data identifying each member of the CDGroup.

CDName The text occurring in the `CDName` element corresponds to the name of a Content Dictionary in the CDGroup. For the syntactical requirements, see `CDName` in Section 5.3.3.

CDVersion The text occurring in the `CDVersion` element identifies which version of the Content Dictionary is to be taken as member of the CDGroup. This element is optional. In case it is missing, the latest version is the one included in the CDGroup. For the syntactical requirements, see `CDVersion` in Section 5.3.3.

CDURL The text occurring in the `CDURL` element identifies the location of the Content Dictionary to be taken as member of the CDGroup. This element is optional. In case it is missing, the location of the CDGroup identified by the element `CDGroupURL` is assumed. For the syntactical requirements, see `CDURL` in Section 5.3.3.

CDComment See `CDComment` in Section 5.3.3.

5.5 Content Dictionaries Reviewing Process

The *OpenMath* Society is responsible for implementing a review and referee process to assess the accuracy of the mathematical content of Content Dictionaries. The status (see `CDStatus`) and/or the version number (see `CDVersion`) of a Content Dictionary may change as a result of this review process.

Chapter 6

OpenMath Compliance

Applications that meet the requirements specified in this chapter may label themselves as OpenMath *compliant*. *OpenMath* compliancy is defined so as to maximize the potential for interoperability amongst *OpenMath* applications.

6.1 Encoding

This standard defines two reference encodings for *OpenMath*, the binary encoding and XML encoding, defined in Chapter 4.

As a minimum, an *OpenMath* compliant application, which accepts or generates *OpenMath* objects, *must* be capable of doing so using the XML encoding. The ability to use other encodings is optional.

6.2 Content Dictionaries

An *OpenMath* compliant application *must* be able to support the error Content Dictionary defined in Appendix A.5.

A compliant application must declare the names and version numbers of the Content Dictionaries that it supports. Equivalently it may declare the Content Dictionary Group (or groups) and major version number (not revision number), rather than listing individual Content Dictionaries. Applications that support all Content Dictionaries (e.g. renderers) should refer to the implicit CD Group `all`.

If a compliant application supports a Content Dictionary then it must explicitly declare any symbols in the Content Dictionaries that are not supported. Phrasebooks are encouraged to support every symbol in the Content Dictionaries.

Symbols which are not listed as unsupported are *supported* by the application. The meaning of *sup-*

ported will depend on the application domain. For example an *OpenMath* renderer should provide a default display for any *OpenMath* object that only references supported symbols, whereas a Computer Algebra System will be expected to map such an object to a suitable internal representation, in this system, of this mathematical object. It is expected that the application's *phrasebooks* for supported Content Dictionaries will be constructed such that propertes of the symbol expressed in the Content Dictionary are respected as far as possible for the given application domain. However *OpenMath* compliance does *not* imply any guarantee by the *OpenMath* Society on the accuracy of these representations.

Content Dictionaries available from the official *OpenMath* repository at www.openmath.org need only be referenced by name, other Content Dictionaries *should* be referenced using the `CDBASE` and the `CDName`.

When receiving an *OpenMath* symbol, e.g. *s*, that is not supported from a supported Content Dictionary, a compliant application will act as if it had received the *OpenMath* object

$$\mathbf{error}(Unhandled\_Symbol, s)$$

where `Unhandled_Symbol` is the symbol from the error Content Dictionary.

Similarly if it receives a symbol, e.g. *s*, from an unsupported Content Dictionary, it will act as if it had received the *OpenMath* object

$$\mathbf{error}(Unsupported\_CD, s)$$

Finally if the compliant application receives a symbol from a supported Content Dictionary but with an unknown name, then this must either be an incorrect object, or possibly the object has been built using a later version of the Content Dictionary. In either case, the application will act as if it had received the *OpenMath* object

$$\mathbf{error}(Unexpected\_Symbol, s)$$

6.3   Lexical Errors

The previous section defines the behaviour of a compliant application upon receiving well formed *OpenMath* objects containing unexpected symbols. This standard does not specify any behaviour for an application upon receiving ill-formed objects.

Chapter 7

Conclusion

The goal of this document is to define the *OpenMath* 2 standard. The issues addressed by the *OpenMath* 2 standard are:

- Informal and formal definition of the *OpenMath* objects.
- Informal and formal definition of the notion of Content Dictionaries.

To do this, *OpenMath* objects are precisely defined and two encodings are described to represent these objects using XML and binary code. Furthermore, the Document Type Definition for validating Content Dictionaries and *OpenMath* objects is given.

Appendix A

CD Files

A.1 The meta Content Dictionary

```
<CD>

<CDComment>

    This document is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

    The copyright holder grants you permission to redistribute this
    document freely as a verbatim copy. Furthermore, the copyright
    holder permits you to develop any derived work from this document
    provided that the following conditions are met.
      a) The derived work acknowledges the fact that it is derived from
         this document, and maintains a prominent reference in the
         work to the original source.
      b) The fact that the derived work is not the original OpenMath
         document is stated prominently in the derived work.  Moreover if
         both this document and the derived work are Content Dictionaries
         then the derived work must include a different CDName element,
         chosen so that it cannot be confused with any works adopted by
         the OpenMath Society.  In particular, if there is a Content
         Dictionary Group whose name is, for example, 'math' containing
         Content Dictionaries named 'math1', 'math2' etc., then you should
         not name a derived Content Dictionary 'mathN' where N is an integer.
         However you are free to name it 'private_mathN' or some such.  This
         is because the names 'mathN' may be used by the OpenMath Society
         for future extensions.
      c) The derived work is distributed under terms that allow the
         compilation of derived works, but keep paragraphs a) and b)
         intact.  The simplest way to do this is to distribute the derived
         work under the OpenMath license, but this is not a requirement.
    If you have questions about this license please contact the OpenMath
    society at http://www.openmath.org.
</CDComment>


<CDName> meta </CDName>
<CDReviewDate> 2003-04-01 </CDReviewDate>
<CDDate> 2001-03-12 </CDDate>
<CDVersion> 2 </CDVersion>
<CDRevision> 0 </CDRevision>
```

```
<CDStatus> official </CDStatus>
<CDURL> http://www.openmath.org/cd/meta.ocd </CDURL>

<Description>
This is a content dictionary to represent content dictionaries, so
that they may be passed between OpenMath compliant application in a
similar way to mathematical objects.

The information written here is taken from chapter 4 of the current
draft of the "OpenMath Standard".
</Description>

<CDDefinition>
<Name> CD </Name>
<Description>
The top level element for the Content Dictionary. It just acts
as a container for the elements described below.
</Description>
</CDDefinition>

<CDComment>
For those that do not have access to the DTD, the elements
allowed in a Content Dictionary are the following
(in no particular order):


&lt;CD&gt;

&lt;CDComment&gt;

     This document is distributed in the hope that it will be useful,
     but WITHOUT ANY WARRANTY; without even the implied warranty of
     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

     The copyright holder grants you permission to redistribute this
     document freely as a verbatim copy. Furthermore, the copyright
     holder permits you to develop any derived work from this document
     provided that the following conditions are met.
       a) The derived work acknowledges the fact that it is derived from
          this document, and maintains a prominent reference in the
          work to the original source.
       b) The fact that the derived work is not the original OpenMath
          document is stated prominently in the derived work.  Moreover if
          both this document and the derived work are Content Dictionaries
```

then the derived work must include a different CDName element,
chosen so that it cannot be confused with any works adopted by
the OpenMath Society. In particular, if there is a Content
Dictionary Group whose name is, for example, 'math' containing
Content Dictionaries named 'math1', 'math2' etc., then you should
not name a derived Content Dictionary 'mathN' where N is an integer.
However you are free to name it 'private_mathN' or some such. This
is because the names 'mathN' may be used by the OpenMath Society
for future extensions.

    c) The derived work is distributed under terms that allow the
compilation of derived works, but keep paragraphs a) and b)
intact. The simplest way to do this is to distribute the derived
work under the OpenMath license, but this is not a requirement.

If you have questions about this license please contact the OpenMath
society at http://www.openmath.org.
</CDComment>

<CDName> </CDName>
<Description> </Description>
<CDReviewDate> </CDReviewDate>
<CDDate> </CDDate>
<CDVersion> </CDVersion>
<CDRevision> </CDRevision>
<CDStatus> </CDStatus>
<CDURL>? </CDURL>
<CDUses>? <CDUses>
<CDDefinition>*
<Name> </Name>
<Description> </Description>
<Example>* </Example>
<FMP>* </FMP>
<CMP>* </CMP>
</CDDefinition>

where a question mark (?) denotes it can be repeated 0 or 1 times, and a star
(*) denotes 0 or more times. <CDComment> </CDComment>
 can also occur in most places – see the DTD for the precise rules.
</CDComment>

<CDDefinition>
<Name> CDDefinition </Name>
<Description>
This symbol is used to represent the element which contains the

definition of each symbol in a content dictionary. That is: it must
contain a 'Name' element and a 'Description' element, and it may contain
an arbitrary number of 'Example', 'FMP' or 'CMP' elements.
```
</Description>
</CDDefinition>

<CDDefinition>
<Name> CDName </Name>
<Description>
```
An element which contains the string corresponding to the name of the CD.
The string must match the syntax for CD names given in the OpenMath
Standard. Here and elsewhere white space occurring at the beginning or
end of the string will be ignored.
```
</Description>
</CDDefinition>

<CDDefinition>
<Name> CDURL </Name>
<Description>
```
An optional element.
If it is used it contains a string representing the URL where the
canonical reference copy of this CD is stored.
```
</Description>
</CDDefinition>

<CDDefinition>
<Name> Example </Name>
<Description>
```
An element which contains an arbitrary number of children,
each of which is either a string or an OpenMath Object.

These children give examples in natural language, or in OpenMath, of the
enclosing symbol definition.
```
</Description>
</CDDefinition>

<CDDefinition>
<Name> CDDate </Name>
<Description>
```
An element which contains a date as a string in the ISO-8601
YYYY-MM-DD format. This gives the date at which the Content Dictionary
was last edited.
```
</Description>
</CDDefinition>
```

```
<CDDefinition>
<Name> CDVersion </Name>
<Description>
An element which contains a version number for the CD.
This should be a non negative integer. Any change to the CD
that affects existing OpenMath applications that support this CD
should result in an increase in the version number.
</Description>
</CDDefinition>


<CDDefinition>
<Name> CDRevision </Name>
<Description>
An element which contains a revision number (or minor version number)
This should be a non-negative integer starting from zero for each
new version. Additional examples  would be typical changes
to a CD requiring a new revision number.
</Description>
</CDDefinition>



<CDDefinition>
<Name> CDReviewDate </Name>
<Description>
An element which contains a date as a string in the ISO-8601
YYYY-MM-DD format. This gives the date at which the Content Dictionary
is next scheduled for review. It should be expected to be stable
until at least this date.
</Description>
</CDDefinition>

<CDDefinition>
<Name> CDStatus </Name>
<Description>
An element giving information on the status of the CD.
The content of the element must be one of the following strings.

official (approved by the OpenMath Society),

experimental (currently being tested),

private (used by a private group of OpenMath users), or
```

obsolete (an obsolete CD kept only for archival purposes).
</Description>
</CDDefinition>

<CDDefinition>
<Name> CDComment </Name>
<Description>
This symbol is used to represent the element of a content dictionary which
explains some aspect of that content dictionary. It should have one string
argument which makes that explanation.
</Description>
</CDDefinition>

<CDDefinition>
<Name> CDUses </Name>
<Description>
An element which contains zero or more CDNames which correspond
to the CDs that this CD depends on, i.e. uses in examples and FMPs. If
the CD is dependent on any other CDs they must be present here.
</Description>
</CDDefinition>

<CDDefinition>
<Name> Description </Name>
<Description>
An element which contains a string corresponding to the
description of either the CD or the symbol
(depending on which is the enclosing element).
</Description>
</CDDefinition>

<CDDefinition>
<Name> Name </Name>
<Description>
An element containing the string corresponding to the name of
the symbol being defined. This must match the syntax for
symbol names given in the OpenMath Standard. Here and elsewhere white
space occurring at the begining or end of the string will be ignored.
</Description>
</CDDefinition>

<CDDefinition>

```
<Name> CMP </Name>
<Description>
An optional element (which may be repeated many times) which contains
a string corresponding to a property of the symbol being
defined.
</Description>
</CDDefinition>

<CDDefinition>
<Name> FMP </Name>
<Description>
An optional element which contains an OpenMath Object.
This corresponds to a property of the symbol being defined.
</Description>
</CDDefinition>

</CD>
```

A.2 The arith1 Content Dictionary File

```
<CD>

<CDComment>

    This document is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

    The copyright holder grants you permission to redistribute this
    document freely as a verbatim copy. Furthermore, the copyright
    holder permits you to develop any derived work from this document
    provided that the following conditions are met.
      a) The derived work acknowledges the fact that it is derived from
         this document, and maintains a prominent reference in the
         work to the original source.
      b) The fact that the derived work is not the original OpenMath
         document is stated prominently in the derived work.  Moreover if
         both this document and the derived work are Content Dictionaries
         then the derived work must include a different CDName element,
         chosen so that it cannot be confused with any works adopted by
         the OpenMath Society.  In particular, if there is a Content
     e    Dictionary Group whose name is, for example, 'math' containing
         Content Dictionaries named 'math1', 'math2' etc., then you should
         not name a derived Content Dictionary 'mathN' where N is an integer.
         However you are free to name it 'private_mathN' or some such.  This
         is because the names 'mathN' may be used by the OpenMath Society
         for future extensions.
      c) The derived work is distributed under terms that allow the
         compilation of derived works, but keep paragraphs a) and b)
         intact.  The simplest way to do this is to distribute the derived
         work under the OpenMath license, but this is not a requirement.
    If you have questions about this license please contact the OpenMath
    society at http://www.openmath.org.
</CDComment>

<CDName> arith1 </CDName>
<CDURL> http://www.openmath.org/cd/arith1.ocd </CDURL>
<CDReviewDate> 2003-04-01 </CDReviewDate>
<CDStatus> official </CDStatus>
<CDDate> 2002-05-31 </CDDate>
<CDVersion> 2 </CDVersion>
```

```
<CDRevision> 1 </CDRevision>
<CDUses>
  <CDName>alg1</CDName>
  <CDName>fns1</CDName>
  <CDName>integer1</CDName>
  <CDName>interval1</CDName>
  <CDName>linalg2</CDName>
  <CDName>logic1</CDName>
  <CDName>quant1</CDName>
  <CDName>relation1</CDName>
  <CDName>set1</CDName>
  <CDName>setname1</CDName>
  <CDName>transc1</CDName>
</CDUses>


<Description>
This CD defines symbols for common arithmetic functions.
</Description>

<CDDefinition>
<Name> lcm </Name>
<Description>
The symbol to represent the n-ary function to return the least common
multiple of its arguments.
</Description>

<CMP> lcm(a,b) = a*b/gcd(a,b) </CMP>

<FMP>
<OMOBJ>
  <OMA>
    <OMS cd="relation1" name="eq"/>
    <OMA>
      <OMS cd="arith1" name="lcm"/>
      <OMV name="a"/>
      <OMV name="b"/>
    </OMA>
    <OMA>
      <OMS cd="arith1" name="divide"/>
      <OMA>
        <OMS cd="arith1" name="times"/>
<OMV name="a"/>
<OMV name="b"/>
      </OMA>
```

```
      <OMA>
        <OMS cd="arith1" name="gcd"/>
        <OMV name="a"/>
        <OMV name="b"/>
      </OMA>
    </OMA>
  </OMA>
</OMOBJ>
</FMP>
<CMP>
for all integers a,b |
There does not exist a c&gt;0 such that c/a is an Integer and c/b is an
Integer and lcm(a,b) &gt; c.
</CMP>

<FMP>
<OMOBJ>
<OMBIND>
  <OMS cd="quant1" name="forall"/>
  <OMBVAR>
    <OMV name="a"/>
    <OMV name="b"/>
  </OMBVAR>
  <OMA>
    <OMS cd="logic1" name="implies"/>
    <OMA>
      <OMS cd="logic1" name="and"/>
      <OMA>
        <OMS cd="set1" name="in"/>
<OMV name="a"/>
<OMS cd="setname1" name="Z"/>
      </OMA>
      <OMA>
        <OMS cd="set1" name="in"/>
<OMV name="b"/>
<OMS cd="setname1" name="Z"/>
      </OMA>
    </OMA>
    <OMA>
      <OMS cd="logic1" name="not"/>
      <OMBIND>
        <OMS cd="quant1" name="exists"/>
        <OMBVAR>
          <OMV name="c"/>
```

```
          </OMBVAR>
          <OMA>
            <OMS cd="logic1" name="and"/>
            <OMA>
              <OMS cd="relation1" name="gt"/>
                <OMV name="c"/>
                <OMI>0</OMI>
            </OMA>
            <OMA>
              <OMS cd="integer1" name="factorof"/>
                <OMV name="a"/>
                <OMV name="c"/>
            </OMA>
            <OMA>
              <OMS cd="integer1" name="factorof"/>
                <OMV name="b"/>
                <OMV name="c"/>
            </OMA>
            <OMA>
              <OMS cd="relation1" name="lt"/>
              <OMV name="c"/>
              <OMA>
                <OMS cd="arith1" name="lcm"/>
                <OMV name="a"/>
                <OMV name="b"/>
              </OMA>
            </OMA>
          </OMA>
        </OMBIND>
      </OMA>
    </OMA>
</OMBIND>
</OMOBJ>
</FMP>
</CDDefinition>

<CDDefinition>
<Name> gcd </Name>
<Description>
The symbol to represent the n-ary function to return the gcd (greatest
common divisor) of its arguments.
</Description>

<CMP>
```

```
for all integers a,b |
There does not exist a c such that a/c is an Integer and b/c is an
Integer and c &gt; gcd(a,b).

Note that this implies that gcd(a,b) &gt; 0
</CMP>

<FMP>
<OMOBJ>
<OMBIND>
  <OMS cd="quant1" name="forall"/>
  <OMBVAR>
    <OMV name="a"/>
    <OMV name="b"/>
  </OMBVAR>
  <OMA>
    <OMS cd="logic1" name="implies"/>
    <OMA>
      <OMS cd="logic1" name="and"/>
      <OMA>
        <OMS cd="set1" name="in"/>
<OMV name="a"/>
<OMS cd="setname1" name="Z"/>
      </OMA>
      <OMA>
        <OMS cd="set1" name="in"/>
<OMV name="b"/>
<OMS cd="setname1" name="Z"/>
      </OMA>
    </OMA>
    <OMA>
      <OMS cd="logic1" name="not"/>
      <OMBIND>
        <OMS cd="quant1" name="exists"/>
        <OMBVAR>
          <OMV name="c"/>
        </OMBVAR>
        <OMA>
          <OMS cd="logic1" name="and"/>
          <OMA>
            <OMS cd="set1" name="in"/>
            <OMA>
              <OMS cd="arith1" name="divide"/>
              <OMV name="a"/>
```

```
          <OMV name="c"/>
        </OMA>
        <OMS cd="setname1" name="Z"/>
      </OMA>
      <OMA>
        <OMS cd="set1" name="in"/>
        <OMA>
          <OMS cd="arith1" name="divide"/>
          <OMV name="b"/>
          <OMV name="c"/>
        </OMA>
        <OMS cd="setname1" name="Z"/>
      </OMA>
      <OMA>
        <OMS cd="relation1" name="gt"/>
        <OMV name="c"/>
        <OMA>
          <OMS cd="arith1" name="gcd"/>
          <OMV name="a"/>
          <OMV name="b"/>
        </OMA>
      </OMA>
    </OMA>
  </OMBIND>
 </OMA>
 </OMA>
</OMBIND>
</OMOBJ>
</FMP>

<Example>
gcd(6,9) = 3
<OMOBJ>
  <OMA>
    <OMS cd="relation1" name="eq"/>
    <OMA>
      <OMS cd="arith1" name="gcd"/>
      <OMI> 6 </OMI>
      <OMI> 9 </OMI>
    </OMA>
    <OMI> 3 </OMI>
  </OMA>
</OMOBJ>
</Example>
```

```
</CDDefinition>

<CDDefinition>
<Name> plus </Name>
<Description>
The symbol representing an n-ary commutative function plus.
</Description>
<CMP> for all a,b | a + b = b + a </CMP>
<FMP>
<OMOBJ>
  <OMBIND>
    <OMS cd="quant1" name="forall"/>
    <OMBVAR>
       <OMV name="a"/>
       <OMV name="b"/>
    </OMBVAR>
    <OMA>
      <OMS cd="relation1" name="eq"/>
      <OMA>
        <OMS cd="arith1" name="plus"/>
        <OMV name="a"/>
        <OMV name="b"/>
      </OMA>
      <OMA>
        <OMS cd="arith1" name="plus"/>
        <OMV name="b"/>
        <OMV name="a"/>
      </OMA>
    </OMA>
  </OMBIND>
</OMOBJ>
</FMP>
</CDDefinition>

<CDDefinition>
<Name> unary_minus </Name>
<Description>
This symbol denotes unary minus, i.e. the additive inverse.
</Description>
<CMP> for all a | a + (-a) = 0 </CMP>
<FMP>
<OMOBJ>
  <OMBIND>
    <OMS cd="quant1" name="forall"/>
```

```
      <OMBVAR>
        <OMV name="a"/>
      </OMBVAR>
      <OMA>
        <OMS cd="relation1" name="eq"/>
        <OMA>
          <OMS cd="arith1" name="plus"/>
          <OMV name="a"/>
          <OMA>
            <OMS cd="arith1" name="unary_minus"/>
            <OMV name="a"/>
          </OMA>
        </OMA>
        <OMS cd="alg1" name="zero"/>
      </OMA>
    </OMBIND>
</OMOBJ>
</FMP>
</CDDefinition>


<CDDefinition>
<Name> minus </Name>
<Description>
The symbol representing a binary minus function. This is equivalent to
adding the additive inverse.
</Description>
<CMP> for all a,b | a - b = a + (-b) </CMP>
<FMP>
<OMOBJ>
  <OMBIND>
    <OMS cd="quant1" name="forall"/>
    <OMBVAR>
        <OMV name="a"/>
        <OMV name="b"/>
    </OMBVAR>
    <OMA>
      <OMS cd="relation1" name="eq"/>
      <OMA>
        <OMS cd="arith1" name="minus"/>
        <OMV name="a"/>
        <OMV name="b"/>
      </OMA>
      <OMA>
        <OMS cd="arith1" name="plus"/>
```

```
        <OMV name="a"/>
        <OMA>
          <OMS cd="arith1" name="unary_minus"/>
          <OMV name="b"/>
        </OMA>
      </OMA>
    </OMA>
  </OMBIND>
</OMOBJ>
</FMP>
</CDDefinition>

<CDDefinition>
<Name> times </Name>
<Description>
The symbol representing an n-ary multiplication function.
</Description>
<Example>
<OMOBJ>
<OMA>
  <OMS cd="relation1" name="eq"/>
  <OMA>
    <OMS cd="arith1" name="times"/>
    <OMA>
      <OMS cd="linalg2" name="matrix"/>
      <OMA>
        <OMS cd="linalg2" name="matrixrow"/>
        <OMI> 1 </OMI>
        <OMI> 2 </OMI>
      </OMA>
      <OMA>
        <OMS cd="linalg2" name="matrixrow"/>
        <OMI> 3 </OMI>
        <OMI> 4 </OMI>
      </OMA>
    </OMA>
    <OMA>
      <OMS cd="linalg2" name="matrix"/>
      <OMA>
        <OMS cd="linalg2" name="matrixrow"/>
        <OMI> 5 </OMI>
        <OMI> 6 </OMI>
      </OMA>
      <OMA>
```

```
          <OMS cd="linalg2" name="matrixrow"/>
          <OMI> 7 </OMI>
          <OMI> 8 </OMI>
        </OMA>
      </OMA>
    </OMA>
    <OMA>
      <OMS cd="linalg2" name="matrix"/>
      <OMA>
        <OMS cd="linalg2" name="matrixrow"/>
        <OMI> 19 </OMI>
        <OMI> 20 </OMI>
      </OMA>
      <OMA>
        <OMS cd="linalg2" name="matrixrow"/>
        <OMI> 43 </OMI>
        <OMI> 50 </OMI>
      </OMA>
    </OMA>
  </OMA>
</OMOBJ>
</Example>
<CMP> for all a,b | a * 0 = 0 and a * b = a * (b - 1) + a </CMP>

<FMP><OMOBJ>
<OMBIND>
  <OMS cd="quant1" name="forall"/>
  <OMBVAR>
    <OMV name="a"/>
    <OMV name="b"/>
  </OMBVAR>
  <OMA>
    <OMS cd="logic1" name="and"/>
    <OMA>
      <OMS cd="relation1" name="eq"/>
      <OMA>
        <OMS cd="arith1" name="times"/>
        <OMV name="a"/>
        <OMS cd="alg1" name="zero"/>
      </OMA>
      <OMS cd="alg1" name="zero"/>
    </OMA>
    <OMA>
      <OMS cd="relation1" name="eq"/>
```

```
      <OMA>
        <OMS cd="arith1" name="times"/>
<OMV name="a"/>
<OMV name="b"/>
      </OMA>
      <OMA>
        <OMS cd="arith1" name="plus"/>
<OMA>
  <OMS cd="arith1" name="times"/>
  <OMV name="a"/>
  <OMA>
    <OMS cd="arith1" name="minus"/>
    <OMV name="b"/>
    <OMS cd="alg1" name="one"/>
  </OMA>
</OMA>
<OMV name="a"/>
      </OMA>
    </OMA>
  </OMA>
</OMBIND>
</OMOBJ></FMP>

<CMP> for all a,b,c | a*(b+c) = a*b + a*c </CMP>
<FMP><OMOBJ>
<OMBIND>
  <OMS cd="quant1" name="forall"/>
  <OMBVAR>
    <OMV name="a"/>
    <OMV name="b"/>
    <OMV name="c"/>
  </OMBVAR>
  <OMA>
    <OMS cd="relation1" name="eq"/>
    <OMA>
      <OMS cd="arith1" name="times"/>
      <OMV name="a"/>
      <OMA>
        <OMS cd="arith1" name="plus"/>
<OMV name="b"/>
<OMV name="c"/>
      </OMA>
    </OMA>
    <OMA>
```

```
      <OMS cd="arith1" name="plus"/>
      <OMA>
        <OMS cd="arith1" name="times"/>
<OMV name="a"/>
<OMV name="b"/>
      </OMA>
      <OMA>
        <OMS cd="arith1" name="times"/>
<OMV name="a"/>
<OMV name="c"/>
      </OMA>
    </OMA>
  </OMA>
</OMBIND>
</OMOBJ></FMP>
</CDDefinition>


<CDDefinition>
<Name> divide </Name>
<Description>
This symbol represents a (binary) division function denoting the first argument
right-divided by the second, i.e. divide(a,b)=a*inverse(b). It is the
inverse of the multiplication function defined by the symbol times in this CD.
</Description>
<CMP> whenever not(a=0) then a/a = 1 </CMP>
<FMP>
<OMOBJ>
  <OMBIND>
    <OMS cd="quant1" name="forall"/>
    <OMBVAR>
      <OMV name="a"/>
    </OMBVAR>
    <OMA>
      <OMS cd="logic1" name="implies"/>
      <OMA>
        <OMS cd="relation1" name="neq"/>
        <OMV name="a"/>
        <OMS cd="alg1" name="zero"/>
      </OMA>
      <OMA>
        <OMS cd="relation1" name="eq"/>
        <OMA>
          <OMS cd="arith1" name="divide"/>
          <OMV name="a"/>
```

```
            <OMV name="a"/>
          </OMA>
          <OMS cd="alg1" name="one"/>
        </OMA>
      </OMA>
    </OMBIND>
</OMOBJ>
</FMP>
</CDDefinition>

<CDDefinition>
<Name> power </Name>
<Description>
This symbol represents a power function. The first argument is raised
to the power of the second argument. When the second argument is not
an integer, powering is defined in terms of exponentials and
logarithms for the complex and real numbers.
This operator can represent general powering.
</Description>

<CMP>
x\in C implies x^a = exp(a ln x)
</CMP>

<FMP>
<OMOBJ>
  <OMA>
    <OMS cd="logic1" name="implies"/>
    <OMA>
      <OMS cd="set1" name="in"/>
      <OMV name="x"/>
      <OMS cd="setname1" name="C"/>
    </OMA>
    <OMA>
      <OMS cd="relation1" name="eq"/>
      <OMA>
        <OMS name="power" cd="arith1"/>
        <OMV name="x"/>
        <OMV name="a"/>
      </OMA>
      <OMA>
        <OMS name="exp" cd="transc1"/>
        <OMA>
          <OMS name="times" cd="arith1"/>
```

```
          <OMV name="a"/>
          <OMA>
            <OMS name="ln" cd="transc1"/>
            <OMV name="x"/>
          </OMA>
        </OMA>
      </OMA>
    </OMA>
  </OMA>
</OMOBJ>
</FMP>


<CMP>
  if n is an integer then
  x^0 = 1,
  x^n = x * x^(n-1)
</CMP>
<FMP>
<OMOBJ>
  <OMA>
    <OMS cd="logic1" name="implies"/>
    <OMA>
      <OMS cd="set1" name="in"/>
      <OMV name="n"/>
      <OMS cd="setname1" name="Z"/>
    </OMA>
    <OMA>
      <OMS cd="logic1" name="and"/>
      <OMA>
        <OMS cd="relation1" name="eq"/>
        <OMA>
          <OMS cd="arith1" name="power"/>
          <OMV name="x"/>
          <OMI>0</OMI>
        </OMA>
        <OMS cd="alg1" name="one"/>
      </OMA>
      <OMA>
        <OMS cd="relation1" name="eq"/>
        <OMA>
          <OMS cd="arith1" name="power"/>
          <OMV name="x"/>
          <OMV name="n"/>
        </OMA>
```

```
      <OMA>
        <OMS cd="arith1" name="times"/>
        <OMV name="x"/>
        <OMA>
          <OMS cd="arith1" name="power"/>
          <OMV name="x"/>
          <OMA>
            <OMS cd="arith1" name="minus"/>
            <OMV name="n"/>
            <OMI>1</OMI>
          </OMA>
        </OMA>
      </OMA>
    </OMA>
  </OMA>
</OMOBJ>
</FMP>
<Example>
<OMOBJ>
<OMA>
  <OMS cd="relation1" name="eq"/>
  <OMA>
    <OMS cd="arith1" name="power"/>
    <OMA>
      <OMS cd="linalg2" name="matrix"/>
      <OMA>
        <OMS cd="linalg2" name="matrixrow"/>
        <OMI> 1 </OMI>
        <OMI> 2 </OMI>
      </OMA>
      <OMA>
        <OMS cd="linalg2" name="matrixrow"/>
        <OMI> 3 </OMI>
        <OMI> 4 </OMI>
      </OMA>
    </OMA>
    <OMI>3</OMI>
  </OMA>
  <OMA>
    <OMS cd="linalg2" name="matrix"/>
    <OMA>
      <OMS cd="linalg2" name="matrixrow"/>
      <OMI> 37 </OMI>
```

```
      <OMI> 54 </OMI>
    </OMA>
    <OMA>
      <OMS cd="linalg2" name="matrixrow"/>
      <OMI> 81 </OMI>
      <OMI> 118 </OMI>
    </OMA>
  </OMA>
</OMA>
</OMOBJ>
</Example>
<Example>
<OMOBJ>
<OMA>
  <OMS cd="relation1" name="eq"/>
  <OMA>
    <OMS cd="arith1" name="power"/>
    <OMS cd="nums1" name="e"/>
    <OMA>
      <OMS cd="arith1" name="times"/>
      <OMS cd="nums1" name="i"/>
      <OMS cd="nums1" name="pi"/>
    </OMA>
  </OMA>
  <OMA>
    <OMS cd="arith1" name="unary_minus"/>
    <OMS cd="alg1" name="one"/>
  </OMA>
</OMA>
</OMOBJ>
</Example>
</CDDefinition>

<CDDefinition>
<Name> abs </Name>
<Description>
A unary operator which represents the absolute value of its
argument. The argument should be numerically valued.
In the complex case this is often referred to as the modulus.
</Description>
<CMP> for all x,y | abs(x) + abs(y) &gt;= abs(x+y) </CMP>
<FMP>
<OMOBJ>
  <OMBIND>
```

```
      <OMS cd="quant1" name="forall"/>
      <OMBVAR>
        <OMV name="x"/>
        <OMV name="y"/>
      </OMBVAR>
      <OMA>
        <OMS cd="relation1" name="geq"/>
        <OMA>
          <OMS cd="arith1" name="plus"/>
          <OMA>
            <OMS cd="arith1" name="abs"/>
              <OMV name="x"/>
          </OMA>
          <OMA>
            <OMS cd="arith1" name="abs"/>
              <OMV name="y"/>
          </OMA>
        </OMA>
        <OMA>
          <OMS cd="arith1" name="abs"/>
          <OMA>
            <OMS cd="arith1" name="plus"/>
            <OMV name="x"/>
            <OMV name="y"/>
          </OMA>
        </OMA>
      </OMA>
    </OMBIND>
  </OMOBJ>
</FMP>
</CDDefinition>


<CDDefinition>
<Name> root </Name>
<Description>
A binary operator which represents its first argument "lowered" to its
n'th root where n is the second argument. This is the inverse of the operation
represented by the power symbol defined in this CD.

Care should be taken as to the precise meaning of this operator, in
particular which root is represented, however it is here to represent
the general notion of taking n'th roots. As inferred by the signature
relevant to this symbol, the function represented by this symbol is
the single valued function, the specific root returned is the one
```

```
indicated by the first CMP. Note also that the converse of the second
CMP is not valid in general.
</Description>

<CMP> x\in C implies root(x,n) = exp(ln(x)/n) </CMP>
<FMP>
  <OMOBJ>
    <OMA>
      <OMS cd="logic1" name="implies"/>
      <OMA>
        <OMS cd="set1" name="in"/>
        <OMV name="x"/>
        <OMS cd="setname1" name="C"/>
      </OMA>
      <OMA>
        <OMS cd="relation1" name="eq"/>
        <OMA>
          <OMS cd="arith1" name="root"/>
          <OMV name="x"/>
          <OMV name="n"/>
        </OMA>
        <OMA>
          <OMS name="exp" cd="transc1"/>
          <OMA>
            <OMS name="divide" cd="arith1"/>
            <OMA>
              <OMS name="ln" cd="transc1"/>
              <OMV name="x"/>
            </OMA>
            <OMV name="n"/>
          </OMA>
        </OMA>
      </OMA>
    </OMA>
  </OMOBJ>
</FMP>

<CMP> for all a,n | power(root(a,n),n) = a (if the root exists!) </CMP>
<FMP>
  <OMOBJ>
    <OMBIND>
      <OMS cd="quant1" name="forall"/>
      <OMBVAR>
        <OMV name="a"/>
```

```
        <OMV name="n"/>
      </OMBVAR>
      <OMA>
        <OMS cd="relation1" name="eq"/>
        <OMA>
          <OMS cd="arith1" name="power"/>
          <OMA>
            <OMS cd="arith1" name="root"/>
            <OMV name="a"/>
            <OMV name="n"/>
          </OMA>
          <OMV name="n"/>
        </OMA>
        <OMV name="a"/>
      </OMA>
    </OMBIND>
  </OMOBJ>
</FMP>
</CDDefinition>




<CDDefinition>
<Name> sum </Name>
<Description>
An operator taking two arguments, the first being the range of summation,
e.g. an integral interval, the second being the function to be
summed. Note that the sum may be over an infinite interval.
</Description>
<Example>
  This represents the summation of the reciprocals of all the integers between
  1 and 10 inclusive.
<OMOBJ>
  <OMA>
    <OMS cd="arith1" name="sum"/>
      <OMA>
        <OMS cd="interval1" name="integer_interval"/>
        <OMI> 1 </OMI>
        <OMI> 10 </OMI>
      </OMA>
    <OMBIND>
      <OMS cd="fns1" name="lambda"/>
        <OMBVAR>
```

```
          <OMV name="x"/>
        </OMBVAR>
        <OMA>
          <OMS cd="arith1" name="divide"/>
          <OMI> 1 </OMI>
          <OMV name="x"/>
        </OMA>
      </OMBIND>
  </OMA>
</OMOBJ>
</Example>
</CDDefinition>


<CDDefinition>
<Name> product </Name>
<Description>
An operator taking two arguments, the first being the range of multiplication
e.g. an integral interval, the second being the function to
be multiplied. Note that the product may be over an infinite interval.
</Description>
<Example>
This represents the statement that the factorial of n is equal to the product
of all the integers between 1 and n inclusive.
<OMOBJ>
  <OMA>
    <OMS cd="relation1" name="eq"/>
    <OMA>
      <OMS cd="integer1" name="factorial"/>
      <OMV name="n"/>
    </OMA>
    <OMA>
      <OMS cd="arith1" name="product"/>
        <OMA>
          <OMS cd="interval1" name="integer_interval"/>
          <OMI> 1 </OMI>
          <OMV name="n"/>
        </OMA>
      <OMBIND>
        <OMS cd="fns1" name="lambda"/>
          <OMBVAR>
            <OMV name="i"/>
          </OMBVAR>
          <OMV name="i"/>
      </OMBIND>
```

```
      </OMA>
    </OMA>
</OMOBJ>
</Example>
</CDDefinition>

</CD>
```

A.3   The arith1 STS Signature File

```
<CDSignatures type="sts" cd="arith1">

  <CDSComment>
    Date:  1999-11-26
    Author: David Carlisle
  </CDSComment>

  <Signature name="lcm" >
    <OMOBJ>
      <OMA>
        <OMS name="mapsto" cd="sts" />
        <OMA>
          <OMS name="nassoc" cd="sts"/>
          <OMV name="SemiGroup"/>
        </OMA>
        <OMV name="SemiGroup" />
      </OMA>
    </OMOBJ>
  </Signature>

  <Signature name="gcd" >
    <OMOBJ>
      <OMA>
        <OMS name="mapsto" cd="sts" />
        <OMA>
          <OMS name="nassoc" cd="sts"/>
          <OMV name="SemiGroup"/>
        </OMA>
        <OMV name="SemiGroup" />
      </OMA>
    </OMOBJ>
  </Signature>

  <Signature name="plus">
    <OMOBJ>
      <OMA>
        <OMS name="mapsto" cd="sts"/>
        <OMA>
          <OMS name="nassoc" cd="sts"/>
          <OMV name="AbelianSemiGroup"/>
        </OMA>
```

```
          <OMV name="AbelianSemiGroup"/>
        </OMA>
      </OMOBJ>
    </Signature>

    <Signature name="unary_minus">
      <OMOBJ>
        <OMA>
          <OMS name="mapsto" cd="sts"/>
          <OMV name="AbelianGroup"/>
          <OMV name="AbelianGroup"/>
        </OMA>
      </OMOBJ>
    </Signature>

    <Signature name="minus">
      <OMOBJ>
        <OMA>
          <OMS name="mapsto" cd="sts"/>
          <OMV name="AbelianGroup"/>
          <OMV name="AbelianGroup"/>
          <OMV name="AbelianGroup"/>
        </OMA>
      </OMOBJ>
    </Signature>

    <Signature name="times">
      <OMOBJ>
        <OMA>
          <OMS name="mapsto" cd="sts"/>
          <OMA>
            <OMS name="nassoc" cd="sts"/>
            <OMV name="SemiGroup"/>
          </OMA>
          <OMV name="SemiGroup"/>
        </OMA>
      </OMOBJ>
    </Signature>

    <Signature name="divide">
      <OMOBJ>
        <OMA>
          <OMS name="mapsto" cd="sts"/>
          <OMV name="AbelianGroup"/>
```

```
        <OMV name="AbelianGroup"/>
        <OMV name="AbelianGroup"/>
      </OMA>
    </OMOBJ>
  </Signature>


  <Signature name="power">
    <OMOBJ>
      <OMA>
        <OMS name="mapsto" cd="sts"/>
        <OMS name="NumericalValue" cd="sts"/>
        <OMS name="NumericalValue" cd="sts"/>
        <OMS name="NumericalValue" cd="sts"/>
      </OMA>
    </OMOBJ>
  </Signature>


  <Signature name="abs">
    <OMOBJ>
      <OMA>
        <OMS name="mapsto" cd="sts"/>
        <OMS name="C" cd="setname1"/>
        <OMS name="R" cd="setname1"/>
      </OMA>
    </OMOBJ>
  </Signature>


  <Signature name="root">
    <OMOBJ>
      <OMA>
        <OMS name="mapsto" cd="sts"/>
        <OMS name="NumericalValue" cd="sts"/>
        <OMS name="NumericalValue" cd="sts"/>
        <OMS name="NumericalValue" cd="sts"/>
      </OMA>
    </OMOBJ>
  </Signature>



  <Signature name="sum" >
    <OMOBJ>
      <OMA>
        <OMS name="mapsto" cd="sts" />
        <OMV name="IntegerRange" />
```

```
      <OMA>
        <OMS name="mapsto" cd="sts" />
        <OMS name="Z" cd="setname1" />
        <OMV name="AbelianMonoid" />
      </OMA>
      <OMV name="AbelianMonoid" />
    </OMA>
  </OMOBJ>
</Signature>

<Signature name="product" >
  <OMOBJ>
    <OMA>
      <OMS name="mapsto" cd="sts" />
      <OMV name="IntegerRange" />
      <OMA>
        <OMS name="mapsto" cd="sts" />
        <OMS name="Z" cd="setname1" />
        <OMV name="AbelianMonoid" />
      </OMA>
      <OMV name="AbelianMonoid" />
    </OMA>
  </OMOBJ>
</Signature>

</CDSignatures>
```

A.4 The MathML CDGroup

```
<CDGroup>
  <CDGroupName>mathml</CDGroupName>
    <CDGroupVersion> 2 </CDGroupVersion>
    <CDGroupRevision> 0 </CDGroupRevision>
    <CDGroupURL>
      http://www.openmath.org/cdfiles/cdgroups/mathml.ocd</CDGroupURL>
    <CDGroupDescription> MathML compatibility CD Group </CDGroupDescription>
    <CDComment>This is the first version of the Core CD group.
      It was created by D Carlisle based on MathML CD Group.</CDComment>
    <CDComment>Algebra</CDComment>
    <CDGroupMember>
      <CDName>alg1</CDName>
      <CDURL>http://www.openmath.org/cd/alg1.ocd</CDURL></CDGroupMember>
    <CDComment>Arithmetic</CDComment>
    <CDGroupMember>
      <CDName>arith1</CDName>
      <CDURL>http://www.openmath.org/cd/arith1.ocd</CDURL></CDGroupMember>
    <CDComment>Constructor for Floating Point Numbers</CDComment>
    <CDGroupMember>
      <CDName>bigfloat1</CDName>
      <CDURL>http://www.openmath.org/cd/bigfloat1.ocd</CDURL></CDGroupMember>
    <CDComment>Calculus</CDComment>
    <CDGroupMember>
      <CDName>calculus1</CDName>
      <CDURL>http://www.openmath.org/cd/calculus1.ocd</CDURL></CDGroupMember>
    <CDComment>Operations on and constructors for complex numbers</CDComment>
    <CDGroupMember>
      <CDName>complex1</CDName>
      <CDURL>http://www.openmath.org/cd/complex1.ocd</CDURL></CDGroupMember>
    <CDComment>Functions on functions</CDComment>
    <CDGroupMember>
      <CDName>fns1</CDName>
      <CDURL>http://www.openmath.org/cd/fns1.ocd</CDURL></CDGroupMember>
    <CDComment>Integer arithmetic</CDComment>
    <CDGroupMember>
      <CDName>integer1</CDName>
      <CDURL>http://www.openmath.org/cd/integer1.ocd</CDURL></CDGroupMember>
    <CDComment>Intervals</CDComment>
    <CDGroupMember>
      <CDName>interval1</CDName>
      <CDURL>http://www.openmath.org/cd/interval1.ocd</CDURL></CDGroupMember>
```

```
<CDComment>Linear Algebra - vector &amp; matrix constructors, those symbols which ar
<CDGroupMember>
  <CDName>linalg1</CDName>
  <CDURL>http://www.openmath.org/cd/linalg1.ocd</CDURL></CDGroupMember>
<CDComment>Linear Algebra - vector &amp; matrix constructors, those symbols which ar
<CDGroupMember>
  <CDName>linalg2</CDName>
  <CDURL>http://www.openmath.org/cd/linalg2.ocd</CDURL></CDGroupMember>
<CDComment>Limits of unary functions</CDComment>
<CDGroupMember>
  <CDName>limit1</CDName>
  <CDURL>http://www.openmath.org/cd/limit1.ocd</CDURL></CDGroupMember>
<CDComment>List constructors</CDComment>
<CDGroupMember>
  <CDName>list1</CDName>
  <CDURL>http://www.openmath.org/cd/list1.ocd</CDURL></CDGroupMember>
<CDComment>Basic logical operators</CDComment>
<CDGroupMember>
  <CDName>logic1</CDName>
  <CDURL>http://www.openmath.org/cd/logic1.ocd</CDURL></CDGroupMember>
<CDComment>
  MathML Numerical Types
</CDComment>
<CDGroupMember>
  <CDName>mathmltypes</CDName>
  <CDURL>http://www.openmath.org/cd/mathmltypes.ocd</CDURL>
</CDGroupMember>
<CDComment>Minima and maxima</CDComment>
<CDGroupMember>
  <CDName>minmax1</CDName>
  <CDURL>http://www.openmath.org/cd/minmax1.ocd</CDURL></CDGroupMember>
<CDComment>Multset-theoretic operators and constructors</CDComment>
<CDGroupMember>
  <CDName>multiset1</CDName>
  <CDURL>http://www.openmath.org/cd/multiset1.ocd</CDURL></CDGroupMember>
<CDComment>Symbols for creating numbers, including some defined constants
  (which can be seen as nullary constructors)</CDComment>
<CDGroupMember>
  <CDName>nums1</CDName>
  <CDURL>http://www.openmath.org/cd/nums1.ocd</CDURL></CDGroupMember>
<CDComment>Symbols for creating piecewise definitions</CDComment>
<CDGroupMember>
  <CDName>piece1</CDName>
  <CDURL>http://www.openmath.org/cd/piece1.ocd</CDURL></CDGroupMember>
```

```
<CDComment>The basic quantifiers forall and exists.</CDComment>
<CDGroupMember>
  <CDName>quant1</CDName>
  <CDURL>http://www.openmath.org/cd/quant1.ocd</CDURL></CDGroupMember>
<CDComment>Common arithmetic relations</CDComment>
<CDGroupMember>
  <CDName>relation1</CDName>
  <CDURL>http://www.openmath.org/cd/relation1.ocd</CDURL></CDGroupMember>
<CDComment>Number sets</CDComment>
<CDGroupMember>
  <CDName>setname1</CDName>
  <CDURL>http://www.openmath.org/cd/setname1.ocd</CDURL></CDGroupMember>
<CDComment>Rounding</CDComment>
<CDGroupMember>
  <CDName>rounding1</CDName>
  <CDURL>http://www.openmath.org/cd/rounding1.ocd</CDURL></CDGroupMember>
<CDComment>Set-theoretic operators and constructors</CDComment>
<CDGroupMember>
  <CDName>set1</CDName>
  <CDURL>http://www.openmath.org/cd/set1.ocd</CDURL></CDGroupMember>
<CDComment>Basic data orientated statistical operators</CDComment>
<CDGroupMember>
  <CDName>s_data1</CDName>
  <CDURL>http://www.openmath.org/cd/s_data1.ocd</CDURL></CDGroupMember>
<CDComment>Basic random variable orientated statistical operators</CDComment>
<CDGroupMember>
  <CDName>s_dist1</CDName>
  <CDURL>http://www.openmath.org/cd/s_dist1.ocd</CDURL></CDGroupMember>
<CDComment>Basic transcendental functions</CDComment>
<CDGroupMember>
  <CDName>transc1</CDName>
  <CDURL>http://www.openmath.org/cd/transc1.ocd</CDURL></CDGroupMember>
<CDComment>Vector calculus functions</CDComment>
<CDGroupMember>
  <CDName>veccalc1</CDName>
  <CDURL>http://www.openmath.org/cd/veccalc1.ocd</CDURL></CDGroupMember>
<CDComment>Alternative encoding symbols for compatibility with the MathML
  Semantic mapping constructs.</CDComment>
<CDGroupMember>
  <CDName>altenc</CDName>
  <CDURL>http://www.openmath.org/cd/altenc.ocd</CDURL></CDGroupMember>
</CDGroup>
```

A.5   The error Content Dictionary

```
<CD>

<CDComment>

    This document is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

    The copyright holder grants you permission to redistribute this
    document freely as a verbatim copy. Furthermore, the copyright
    holder permits you to develop any derived work from this document
    provided that the following conditions are met.
      a) The derived work acknowledges the fact that it is derived from
         this document, and maintains a prominent reference in the
         work to the original source.
      b) The fact that the derived work is not the original OpenMath
         document is stated prominently in the derived work.  Moreover if
         both this document and the derived work are Content Dictionaries
         then the derived work must include a different CDName element,
         chosen so that it cannot be confused with any works adopted by
         the OpenMath Society.  In particular, if there is a Content
         Dictionary Group whose name is, for example, 'math' containing
         Content Dictionaries named 'math1', 'math2' etc., then you should
         not name a derived Content Dictionary 'mathN' where N is an integer.
         However you are free to name it 'private_mathN' or some such.  This
         is because the names 'mathN' may be used by the OpenMath Society
         for future extensions.
      c) The derived work is distributed under terms that allow the
         compilation of derived works, but keep paragraphs a) and b)
         intact.  The simplest way to do this is to distribute the derived
         work under the OpenMath license, but this is not a requirement.
    If you have questions about this license please contact the OpenMath
    society at http://www.openmath.org.
</CDComment>

<CDName> error </CDName>
<CDURL> http://www.openmath.org/cd/error.ocd </CDURL>
<CDReviewDate> 2003-04-01 </CDReviewDate>
<CDStatus> official </CDStatus>
<CDDate> 2001-03-12 </CDDate>
<CDVersion> 2 </CDVersion>
```

```
<CDRevision> 0 </CDRevision>
<CDUses>
<CDName> arith1 </CDName>
<CDName> specfun1 </CDName>
</CDUses>

<CDDefinition>
<Name> unhandled_symbol </Name>
<Description>
This symbol represents the error which is raised when an application
reads a symbol which is present in the mentioned content
dictionary, but which it has not implemented.

When receiving such a symbol, the application should act as if it had
received the OpenMath error object constructed from unhandled_symbol
and the unhandled symbol as in the example below.
</Description>

<Example>
The application does not implement the Complex numbers:
<OMOBJ>
  <OME>
    <OMS cd="error" name="unhandled_symbol"/>
    <OMS cd="setname1" name="C"/>
  </OME>
</OMOBJ>
</Example>
</CDDefinition>

<CDDefinition>
<Name> unexpected_symbol </Name>
<Description>
This symbol represents the error which is raised when an application
reads a symbol which is not present in the mentioned content dictionary.

When receiving such a symbol, the application should act as if it had
received the OpenMath error object constructed from unexpected_symbol
and the unexpected symbol as in the example below.
</Description>
<Example>
The application received a mistyped symbol
<OMOBJ>
  <OME>
    <OMS cd="error" name="unexpected_symbol"/>
```

```
      <OMS cd="arith1" name="plurse"/>
    </OME>
</OMOBJ>
</Example>
</CDDefinition>

<CDDefinition>
<Name> unsupported_CD </Name>
<Description>
This symbol represents the error which is raised when an application
reads a symbol where the mentioned content dictionary is not
present.

When receiving such a symbol, the application should act as if it had
received the OpenMath error object constructed from unsupported_CD and
the symbol from the unsupported Content Dictionary as in the example
below.
</Description>
<Example>
The application does not know about the CD specfun1
<OMOBJ>
  <OME>
    <OMS cd="error" name="unsupported_CD"/>
    <OMS cd="specfun1" name="BesselJ"/>
  </OME>
</OMOBJ>
</Example>
</CDDefinition>

</CD>
```

Appendix B

OpenMath Schema in Relax NG XML Syntax (Normative)

This is the Relax NG Schema described in Section 4.1 expressed according to the Relax NG XML Syntax.

```
<grammar ns="http://www.openmath.org/OpenMath" datatypeLibrary="http://www.w3.org/2001/XMLSchema-
  <!-- OM2: allow OMR -->
  <define name="omel">
    <choice>
      <ref name="OMS"/>
      <ref name="OMV"/>
      <ref name="OMI"/>
      <ref name="OMB"/>
      <ref name="OMSTR"/>
      <ref name="OMF"/>
      <ref name="OMA"/>
      <ref name="OMBIND"/>
      <ref name="OME"/>
      <ref name="OMATTR"/>
      <ref name="OMR"/>
    </choice>
  </define>
  <!-- things which can be variables -->
  <define name="omvar">
    <choice>
      <ref name="OMV"/>
      <ref name="attvar"/>
```

```
      </choice>
    </define>
    <define name="attvar">
      <element name="OMATTR">
        <ref name="common.attributes"/>
        <group>
          <ref name="OMATP"/>
          <choice>
            <ref name="OMV"/>
            <ref name="attvar"/>
          </choice>
        </group>
      </element>
    </define>
    <!-- OM2: common attributes -->
    <define name="cdbase">
      <optional>
        <attribute name="cdbase">
          <data type="anyURI"/>
        </attribute>
      </optional>
    </define>
    <define name="common.attributes">
      <optional>
        <attribute name="id">
          <data type="ID"/>
        </attribute>
      </optional>
    </define>
    <define name="compound.attributes">
      <ref name="common.attributes"/>
      <ref name="cdbase"/>
    </define>
    <!-- symbol -->
    <define name="OMS">
      <element name="OMS">
        <ref name="common.attributes"/>
        <ref name="attlist.OMS"/>
      </element>
    </define>
    <define name="attlist.OMS">
      <attribute name="name">
        <data type="NCName"/>
      </attribute>
```

```
    <attribute name="cd">
      <data type="NCName"/>
    </attribute>
    <ref name="cdbase"/>
</define>
<!-- variable -->
<define name="OMV">
  <element name="OMV">
    <ref name="common.attributes"/>
    <ref name="attlist.OMV"/>
    <optional>
      <ref name="omel"/>
    </optional>
  </element>
</define>
<define name="attlist.OMV">
  <attribute name="name">
    <data type="NCName"/>
  </attribute>
</define>
<!-- integer -->
<define name="OMI">
  <element name="OMI">
    <ref name="common.attributes"/>
    <data type="string">
      <param name="pattern">\s*(-\s?)?[0-9]+(\s[0-9]+)*\s*</param>
    </data>
  </element>
</define>
<!-- byte array -->
<define name="OMB">
  <element name="OMB">
    <ref name="common.attributes"/>
    <data type="base64Binary"/>
  </element>
</define>
<!-- string -->
<define name="OMSTR">
  <element name="OMSTR">
    <ref name="common.attributes"/>
    <text/>
  </element>
</define>
<!-- floating point -->
```

```
<define name="OMF">
  <element name="OMF">
    <ref name="common.attributes"/>
    <ref name="attlist.OMF"/>
  </element>
</define>
<define name="attlist.OMF">
  <choice>
    <attribute name="dec">
      <data type="string">
        <param name="pattern">(-?)([0-9]+)?(\.[0-9]+)?(e([+\-]?)[0-9]+)?</param>
      </data>
    </attribute>
    <attribute name="hex">
      <data type="string">
        <param name="pattern">[0-9A-F]+</param>
      </data>
    </attribute>
  </choice>
</define>
<!-- apply constructor -->
<define name="OMA">
  <element name="OMA">
    <ref name="compound.attributes"/>
    <oneOrMore>
      <ref name="omel"/>
    </oneOrMore>
  </element>
</define>
<!-- binding constructor and variable -->
<define name="OMBIND">
  <element name="OMBIND">
    <ref name="compound.attributes"/>
    <ref name="omel"/>
    <ref name="OMBVAR"/>
    <ref name="omel"/>
  </element>
</define>
<define name="OMBVAR">
  <element name="OMBVAR">
    <ref name="common.attributes"/>
    <oneOrMore>
      <ref name="omvar"/>
    </oneOrMore>
```

```
      </element>
  </define>
  <!-- error -->
  <define name="OME">
    <element name="OME">
      <ref name="common.attributes"/>
      <ref name="OMS"/>
      <zeroOrMore>
        <ref name="omel"/>
      </zeroOrMore>
    </element>
  </define>
  <!-- attribution constructor and attribute pair constructor -->
  <define name="OMATTR">
    <element name="OMATTR">
      <ref name="compound.attributes"/>
      <ref name="OMATP"/>
      <ref name="omel"/>
    </element>
  </define>
  <!-- OM2: allow OMFOREIGN -->
  <define name="OMATP">
    <element name="OMATP">
      <ref name="compound.attributes"/>
      <oneOrMore>
        <ref name="OMS"/>
        <choice>
          <ref name="omel"/>
          <ref name="OMFOREIGN"/>
        </choice>
      </oneOrMore>
    </element>
  </define>
  <!-- OM2: OMFOREIGN -->
  <define name="OMFOREIGN">
    <element name="OMFOREIGN">
      <ref name="compound.attributes"/>
      <zeroOrMore>
        <choice>
          <ref name="omel"/>
          <ref name="notom"/>
        </choice>
      </zeroOrMore>
    </element>
```

```
    </define>
    <!-- Any elements not in the om namespace (valid om is allowed as a descendant) -->
    <define name="notom">
      <choice>
        <element>
          <anyName>
            <except>
              <nsName/>
            </except>
          </anyName>
          <zeroOrMore>
            <attribute>
              <anyName/>
            </attribute>
          </zeroOrMore>
          <zeroOrMore>
            <choice>
              <ref name="omel"/>
              <ref name="notom"/>
            </choice>
          </zeroOrMore>
        </element>
        <text/>
      </choice>
    </define>
    <!-- OM object constructor -->
    <define name="OMOBJ">
      <element name="OMOBJ">
        <ref name="compound.attributes"/>
        <ref name="omel"/>
      </element>
    </define>
    <define name="attlist.OMOBJ">
      <attribute name="version">
        <choice>
          <value>1.0</value>
          <value>1.2</value>
          <value>2.0</value>
        </choice>
      </attribute>
    </define>
    <!-- OM2: OMR -->
    <define name="OMR">
      <element name="OMR">
```

```
      <ref name="common.attributes"/>
      <ref name="attlist.OMR"/>
    </element>
  </define>
  <define name="attlist.OMR">
    <attribute name="xlink:href"/>
    <attribute name="xlink:type">
      <value>simple</value>
    </attribute>
    <attribute name="xlink:show">
      <value>embed</value>
    </attribute>
  </define>
  <start>
    <ref name="OMOBJ"/>
  </start>
</grammar>
```

Appendix C

Restricting the *OpenMath* Schema (Non-Normative)

Relax NG allows one to state constraints such as *if the cd attribute of OMS is arith1 then the name attribute must be one of lcm, gcd, plus etc.* Thus it is easy to use a stylesheet to generate for any given CD, a Relax NG schema that expresses the constraint that an `OMS` naming that CD must only use symbols defined in the specified dictionary. Similarly it is possible to use the *role* information contained in the CD to restrict which symbols can be the first child of an `OMBIND` or the odd-numbered children of an `OMATP`.

The modularisation mechanisms of Relax NG then allow one to include these schema for all the CDs that you want to allow and, for example, to replace the regexp-based validation of the `OMS` attributes by explicit lists of allowed CD names, and for each CD Name, a list of allowed symbol names.

For example, a CD-specific Relax NG Schema for the arith1 CD shown in Appendix A.2 would look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0" datatypeLibrary="">
  <define name="cd.attlist.OMS" combine="choice">
    <attribute name="cd">
      <value type="string">arith1</value>
    </attribute>
    <attribute name="name">
      <choice>
        <value type="string">lcm</value>
        <value type="string">gcd</value>
        <value type="string">plus</value>
        <value type="string">unary_minus</value>
        <value type="string">minus</value>
```

```
        <value type="string">times</value>
        <value type="string">divide</value>
        <value type="string">power</value>
        <value type="string">abs</value>
        <value type="string">root</value>
        <value type="string">sum</value>
        <value type="string">product</value>
      </choice>
    </attribute>
  </define>
</grammar>
```

or, using the Relax NG compact syntax:

```
cd.attlist.OMS |=
attribute cd {string "arith1" },
attribute name {
string "lcm" |
string "gcd" |
string "plus" |
string "unary_minus" |
string "minus" |
string "times" |
string "divide" |
string "power" |
string "abs" |
string "root" |
string "sum" |
string "product" }
```

To build a schema that allows only symbols from arith1 we just need to include the *OpenMath* schema described in Section 4.1.1, override the attribute declarations for OMS, and then include the schema for arith1. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
  <include href="openmath.rng">
    <define name="attlist.OMS">
      <ref name="cd.attlist.OMS"/>
    </define>
  </include>
  <include href="arith1.rng"/>
</grammar>
```

or, in the compact syntax:

```
include "openmath.rnc" {
attlist.OMS = cd.attlist.OMS}

include "arith1.rnc"
```

Using this approach it is possible to include as many files as required.

Appendix D

OpenMath Schema in XSD Syntax (Non-Normative)

This is an XSD Schema generated from the Relax NG Schema described in Section 4.1.

```
<schema elementFormDefault="qualified" targetNamespace="http://www.openmath.org/OpenMath">
  <import namespace="http://www.w3.org/1999/xlink" schemaLocation="xlink.xsd"/>
  <!-- OM2: allow OMR -->
  <group name="omel">
    <choice>
      <element ref="om:OMS"/>
      <element ref="om:OMV"/>
      <element ref="om:OMI"/>
      <element ref="om:OMB"/>
      <element ref="om:OMSTR"/>
      <element ref="om:OMF"/>
      <element ref="om:OMA"/>
      <element ref="om:OMBIND"/>
      <element ref="om:OME"/>
      <group ref="om:OMATTR"/>
      <element ref="om:OMR"/>
    </choice>
  </group>
  <!-- things which can be variables -->
  <group name="omvar">
    <choice>
      <element ref="om:OMV"/>
      <group ref="om:attvar"/>
```

```
        </choice>
      </group>
      <group name="attvar">
        <sequence>
          <element name="OMATTR">
            <complexType>
              <sequence>
                <element ref="om:OMATP"/>
                <choice>
                  <element ref="om:OMV"/>
                  <group ref="om:attvar"/>
                </choice>
              </sequence>
              <attributeGroup ref="om:common.attributes"/>
            </complexType>
          </element>
        </sequence>
      </group>
      <!-- OM2: common attributes -->
      <attributeGroup name="cdbase">
        <attribute name="cdbase" type="xs:anyURI"/>
      </attributeGroup>
      <attributeGroup name="common.attributes">
        <attribute name="id" type="xs:ID"/>
      </attributeGroup>
      <attributeGroup name="compound.attributes">
        <attributeGroup ref="om:common.attributes"/>
        <attributeGroup ref="om:cdbase"/>
      </attributeGroup>
      <!-- symbol -->
      <element name="OMS">
        <complexType>
          <attributeGroup ref="om:common.attributes"/>
          <attributeGroup ref="om:attlist.OMS"/>
        </complexType>
      </element>
      <attributeGroup name="attlist.OMS">
        <attribute name="name" use="required" type="xs:NCName"/>
        <attribute name="cd" use="required" type="xs:NCName"/>
        <attributeGroup ref="om:cdbase"/>
      </attributeGroup>
      <!-- variable -->
      <element name="OMV">
        <complexType>
```

```
      <group minOccurs="0" ref="om:omel"/>
      <attributeGroup ref="om:common.attributes"/>
      <attributeGroup ref="om:attlist.OMV"/>
    </complexType>
</element>
<attributeGroup name="attlist.OMV">
  <attribute name="name" use="required" type="xs:NCName"/>
</attributeGroup>
<!-- integer -->
<element name="OMI">
  <complexType>
    <simpleContent>
      <restriction base="xs:anyType">
        <simpleType>
          <restriction base="xs:string">
            <pattern value="\s*(-\s?)?[0-9]+(\s[0-9]+)*\s*"/>
          </restriction>
        </simpleType>
        <attributeGroup ref="om:common.attributes"/>
      </restriction>
    </simpleContent>
  </complexType>
</element>
<!-- byte array -->
<element name="OMB">
  <complexType>
    <simpleContent>
      <extension base="xs:base64Binary">
        <attributeGroup ref="om:common.attributes"/>
      </extension>
    </simpleContent>
  </complexType>
</element>
<!-- string -->
<element name="OMSTR">
  <complexType mixed="true">
    <attributeGroup ref="om:common.attributes"/>
  </complexType>
</element>
<!-- floating point -->
<element name="OMF">
  <complexType>
    <attributeGroup ref="om:common.attributes"/>
    <attributeGroup ref="om:attlist.OMF"/>
```

```
        </complexType>
      </element>
      <attributeGroup name="attlist.OMF">
        <attribute name="dec">
          <simpleType>
            <restriction base="xs:string">
              <pattern value="(-?)([0-9]+)?(\.[0-9]+)?(e([+\-]?)[0-9]+)?"/>
            </restriction>
          </simpleType>
        </attribute>
        <attribute name="hex">
          <simpleType>
            <restriction base="xs:string">
              <pattern value="[0-9A-F]+"/>
            </restriction>
          </simpleType>
        </attribute>
      </attributeGroup>
      <!-- apply constructor -->
      <element name="OMA">
        <complexType>
          <group maxOccurs="unbounded" ref="om:omel"/>
          <attributeGroup ref="om:compound.attributes"/>
        </complexType>
      </element>
      <!-- binding constructor and variable -->
      <element name="OMBIND">
        <complexType>
          <sequence>
            <group ref="om:omel"/>
            <element ref="om:OMBVAR"/>
            <group ref="om:omel"/>
          </sequence>
          <attributeGroup ref="om:compound.attributes"/>
        </complexType>
      </element>
      <element name="OMBVAR">
        <complexType>
          <group maxOccurs="unbounded" ref="om:omvar"/>
          <attributeGroup ref="om:common.attributes"/>
        </complexType>
      </element>
      <!-- error -->
      <element name="OME">
```

```
    <complexType>
      <sequence>
        <element ref="om:OMS"/>
        <group minOccurs="0" maxOccurs="unbounded" ref="om:omel"/>
      </sequence>
      <attributeGroup ref="om:common.attributes"/>
    </complexType>
  </element>
<!-- attribution constructor and attribute pair constructor -->
<group name="OMATTR">
  <sequence>
    <element name="OMATTR">
      <complexType>
        <sequence>
          <element ref="om:OMATP"/>
          <group ref="om:omel"/>
        </sequence>
        <attributeGroup ref="om:compound.attributes"/>
      </complexType>
    </element>
  </sequence>
</group>
<!-- OM2: allow OMFOREIGN -->
<element name="OMATP">
  <complexType>
    <sequence maxOccurs="unbounded">
      <element ref="om:OMS"/>
      <choice>
        <group ref="om:omel"/>
        <element ref="om:OMFOREIGN"/>
      </choice>
    </sequence>
    <attributeGroup ref="om:compound.attributes"/>
  </complexType>
</element>
<!-- OM2: OMFOREIGN -->
<element name="OMFOREIGN">
  <complexType mixed="true">
    <choice minOccurs="0" maxOccurs="unbounded">
      <group ref="om:omel"/>
      <group ref="om:notom"/>
    </choice>
    <attributeGroup ref="om:compound.attributes"/>
  </complexType>
```

```
    </element>
    <!-- Any elements not in the om namespace (valid om is allowed as a descendant) -->
    <group name="notom">
      <sequence>
        <choice minOccurs="0">
          <any namespace="##other" processContents="skip"/>
          <any namespace="##local" processContents="skip"/>
        </choice>
      </sequence>
    </group>
    <!-- OM object constructor -->
    <element name="OMOBJ">
      <complexType>
        <group ref="om:omel"/>
        <attributeGroup ref="om:compound.attributes"/>
      </complexType>
    </element>
    <attributeGroup name="attlist.OMOBJ">
      <attribute name="version" use="required">
        <simpleType>
          <restriction base="xs:token">
            <enumeration value="1.0"/>
            <enumeration value="1.2"/>
            <enumeration value="2.0"/>
          </restriction>
        </simpleType>
      </attribute>
    </attributeGroup>
    <!-- OM2: OMR -->
    <element name="OMR">
      <complexType>
        <attributeGroup ref="om:common.attributes"/>
        <attributeGroup ref="om:attlist.OMR"/>
      </complexType>
    </element>
    <attributeGroup name="attlist.OMR">
      <attribute ref="xlink:href" use="required"/>
      <attribute ref="xlink:type" use="required"/>
      <attribute ref="xlink:show" use="required"/>
    </attributeGroup>
</schema>
```

Appendix E

OpenMath DTD (Non-Normative)

This is a DTD generated from the Relax NG Schema described in Section 4.1. Note that we cannot express the fact that the OMFOREIGN element can contain any well-formed XML, so we have simply restricted it to contain any XML defined in the DTD.

```
<?xml encoding="UTF-8"?>

<!-- RELAX NG Schema for OpenMath 2 -->

<!-- default namespace om = "http://www.openmath.org/OpenMath" -->

<!-- OM2: allow OMR -->

<!ENTITY % omel "OMS|OMV|OMI|OMB|OMSTR|OMF|OMA|OMBIND|OME|OMATTR|OMR">

<!ENTITY % attvar "OMATTR">

<!-- things which can be variables -->

<!ENTITY % omvar "OMV|%attvar;">

<!-- OM2: common attributes -->

<!ENTITY % cdbase "
  cdbase CDATA #IMPLIED">

<!ENTITY % common.attributes "
  id ID #IMPLIED">

<!ENTITY % compound.attributes "
```

```
  %common.attributes;
  %cdbase;">

<!ENTITY % attlist.OMS "
  name NMTOKEN #REQUIRED
  cd NMTOKEN #REQUIRED
  %cdbase;">

<!-- symbol -->

<!ELEMENT OMS EMPTY>
<!ATTLIST OMS
  %common.attributes;
  %attlist.OMS;>

<!ENTITY % attlist.OMV "
  name NMTOKEN #REQUIRED">

<!-- variable -->

<!ELEMENT OMV (%omel;)?>
<!ATTLIST OMV
  %common.attributes;
  %attlist.OMV;>

<!-- integer -->

<!ELEMENT OMI (#PCDATA)>
<!ATTLIST OMI
  %common.attributes;>

<!-- byte array -->

<!ELEMENT OMB (#PCDATA)>
<!ATTLIST OMB
  %common.attributes;>

<!-- string -->

<!ELEMENT OMSTR (#PCDATA)>
<!ATTLIST OMSTR
  %common.attributes;>

<!ENTITY % attlist.OMF "
```

```
  dec CDATA #IMPLIED
  hex CDATA #IMPLIED">

<!-- floating point -->

<!ELEMENT OMF EMPTY>
<!ATTLIST OMF
  %common.attributes;
  %attlist.OMF;>

<!-- apply constructor -->

<!ELEMENT OMA (%omel;)+>
<!ATTLIST OMA
  %compound.attributes;>

<!-- binding constructor and variable -->

<!ELEMENT OMBIND ((%omel;),OMBVAR,(%omel;))>
<!ATTLIST OMBIND
  %compound.attributes;>

<!ELEMENT OMBVAR (%omvar;)+>
<!ATTLIST OMBVAR
  %common.attributes;>

<!-- error -->

<!ELEMENT OME (OMS,(%omel;)*)>
<!ATTLIST OME
  %common.attributes;>

<!-- attribution constructor and attribute pair constructor -->

<!ELEMENT OMATTR (OMATP,(%omel;))>
<!ATTLIST OMATTR
  %compound.attributes;>

<!-- OM2: allow OMFOREIGN  -->

<!ELEMENT OMATP (OMS,(%omel;|OMFOREIGN))+>
<!ATTLIST OMATP
  %compound.attributes;>
```

```
<!-- OM2: OMFOREIGN  -->

<!ELEMENT OMFOREIGN ANY>
<!ATTLIST OMFOREIGN
  %compound.attributes;>

<!-- Any elements not in the om namespace (valid om is allowed as a descendant) -->

<!-- OM object constructor -->

<!ELEMENT OMOBJ (%omel;)>
<!ATTLIST OMOBJ
  %compound.attributes;>

<!ENTITY % attlist.OMOBJ "
  version (1.0|1.2|2.0) #REQUIRED">

<!ENTITY % attlist.OMR "
  xlink:href CDATA #REQUIRED
  xlink:type (simple) #REQUIRED
  xlink:show (embed) #REQUIRED">

<!-- OM2: OMR -->

<!ELEMENT OMR EMPTY>
<!ATTLIST OMR
  xmlns:xlink CDATA #FIXED 'http://www.w3.org/1999/xlink'
  %common.attributes;
  %attlist.OMR;>
```

Appendix F

Changes between *OpenMath* 1.1 and *OpenMath* 2 (Non-Normative)

In this appendix we describe the major changes that occurred between version 1.1 and version 2 of the *OpenMath* standard. All changes to the encodings and content dictionaries have been designed to be backward compatibile, in other words all existing *OpenMath* objects and Content Dictionaries are still valid in *OpenMath* 2. On the other hand an existing *OpenMath* 1.1 application may not be able to process *OpenMath* 2 objects.

F.1   Changes to the Formal Definition of Objects

Additional features of abstract objects have been introduced:

- *OpenMath* symbols have an optional rôle qualifier which restricts the place where they may occur within compound objects. Although part of the abstract description of a symbol this information is intended to be stored in the CD. In the XML encoding it may be used to provide a more restricted schema leading to tighter validation.

- In addition to their *name* and *cd* properties, symbols now have an optional *cdbase* property. This can be used to disambiguate between two CDs which are produced independently but have the same name, and can also be used to produce a canonical URI for any *OpenMath* symbol for use in frameworks such as RDFS or MathML which need one.

- *OpenMath* variables can be indexed, in which case they carry information about their enumerator. Since *OpenMath* only allows variables and not objects to be bound, it was not possible to produce indexed variables by defining an appropriate symbol in a CD. Also, although it would be possible to produce an indexed variable using the new *semantic-attribute* role, it was felt that attributes describe or modify an object, whereas in this case we were constructing an atomic object.

- An *OpenMath* object may be attributed with a non-*OpenMath* object using the new *foreign* constructor. This allows an XML-encoded *OpenMath* object to be attributed with appropriate Presentation MathML, for example, or a base-64 encoded MPEG file of its aural rendering.

- the new role property can be used to indicate that a symbol is an *attribution*, in which case an application may ignore or remove it, or a *semantic attribution* in which case removing it is no longer guarenteed to produce an equivalent object.

- restrictions on the names of symbols, variables and content dictionaries have been relaxed to be compatible with XML and to be less Anglo-Saxon.

F.2 Changes to the encodings

The *OpenMath* version 2 standard still mandantes two encodings: XML and binary. The XML encoding in particular has been updated to reflect the latest development of XML and is now a full XML application. Version 2 encodings are backward compatible with version 1.1 encodings.

- both encodings have been updated to support the changes to the model of abstract objects described above.
- encodings support internal and external sharing of objects
- the XML encoding in version 2 is defined by a Relax NG schema and the mandated character-based grammar of version 1 has been removed, while the DTD has been relegated to an Appendix.
- an optional attribute defining the version of the encoding can be specified for the encoded object

F.3 Changes to Content Dictionaries

- In *OpenMath* version 2 Content Dictionaries are defined in terms of the abstract information content that needs to be specified for defining *OpenMath* symbols. The current implementation is thus just one possible encoding of this abstract model.
- The *CDUses* element is not part of this information model and has been made optional and deprecated in the reference encoding since it is trivial to extract its content automatically from the CD.
- A CD may now, optionally, define its cdbase.
- A CD symbol definition may now, optionally, define its role.

Bibliography

[1] John A. Abbott, André van Leeuwen and Andreas Strotmann *OpenMath*: Communicating Mathematical Information between Co-operating Agents in a Knowledge Network

[2] N. Borenstein and N Freed MIME (Multipurpose Internet Mail Extensions) Part One: Mechanism for Specifying and Describing the Format of Internet Message Bodies

[3] O. Caprotti and A. M. Cohen A Type System for *OpenMath*

[4] Olga Caprotti and Arjeh M. Cohen A Type System for *OpenMath*

[5] S. Dalmas, M. Gaëtano and S. Watt An *OpenMath* 1.0 Implementation

[6] J. Davenport A Small *OpenMath* Type System

[7] IEEE Std 1003.1-2001 (Open Group Technical Standard, Issue 6), Standard for Information Technology-Portable Operating System Interface (POSIX)

[8] IEEE Standard for binary Floating-Point Arithmetic

[9] IETF RFC 2396 - Uniform Resource Identifiers (URI): Generic Syntax

[10] ISO 7-bit coded character set for information interchange

[11] OASIS Committee Specification RELAX NG Specification

[12] OpenMath Consortium *OpenMath* Version 1 - Draft

[13] Technical committee / subcommittee: JTC 1 ISO 9660:1988 Information processing –Volume and File Structure of CDROM for Information Interchange

[14] Unicode Consortium The Unicode Standard: Version 4.0

[15] World Wide Web Consortium XSD Schema Parts 1 & 2

[16] World Wide Web Consortium Namespaces in XML

[17] World Wide Web Consortium Extensible Markup Language XML 1.0

[18] World Wide Web Consortium Mathematical Markup Language (MathML) 2.0 Specification

[19] World Wide Web Consortium Extensible Stylesheet Language (XSL) Specification

[20] F. Yergeau UTF-8, a transformation format of ISO 10646