The OpenMath 2.0 Standard (created from LaTeX)

The OpenMath Society
S. Buswell, O. Caprotti, D. P. Carlisle, M. C. Dewar, M. Gäetano, M. Kohlhase

June 2004

Abstract

This document describes version 2.0 of OpenMath: a standard for the representation and communication of mathematical objects.

This version clarifies and extends OpenMath 1.1 [Con02]. OpenMath allows the *meaning* of an object to be encoded rather than just a visual representation. It is designed to allow the free exchange of mathematical objects between software systems and human beings. On the worldwide web it is designed to allow mathematical expressions embedded in web pages to be manipulated and computed with in a meaningful and correct way. It is designed to be machine-generatable and machine-readable, rather than written by hand.

The OpenMath Standard is the official reference for the OpenMath language and has been approved by the OpenMath Society. It is not intended as an introductory document or a user's guide, for the latest available material of this nature please consult the OpenMath web-site at `http://www.openmath.org`.

This document includes an overview of the OpenMath architecture, an abstract description of OpenMath objects and two mechanisms for producing concrete encodings of such objects. The first, in XML, is designed primarily for use on the web, in documents, and for applications which want to mix OpenMath as a content representation with MathML as a presentation format. The second, a binary format, is designed for applications which wish to exchange very large objects, or a lot of data as efficiently as possible. This document also includes a description of Content Dictionaries - the mechanism by which the meaning of a symbol in the OpenMath language is encoded, as well as an XML encoding for them. Finally it includes guidelines for the development of OpenMath-compliant applications. Further background on OpenMath and guidelines for its use in applications may be found in the accompanying Primer [Con04a].

Contents

Chapter 1

Introduction to OpenMath

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

1.1 OpenMath Architecture

The architecture of OpenMath is described in 1.1 and summarizes the interactions among the different OpenMath components. There are three layers of representation of a mathematical object. The first is a private layer that is the internal representation used by an application. The second is an abstract layer that is the representation as an OpenMath object. Note that these two layers may, in some cases, be the same. The third is a communication layer that translates the OpenMath object representation into 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.

1.2 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 2 and 3.

The standard endorses two encodings in XML and binary formats. At the time of writing, these are the encodings supported by most existing OpenMath tools and applications, however they are not the only possible encodings of OpenMath objects. Users who wish to define their own encoding , are free to do so provided that there is a well-defined correspondence between the new encoding and the abstract model defined in 2.

1.3 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 OpenMath 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.

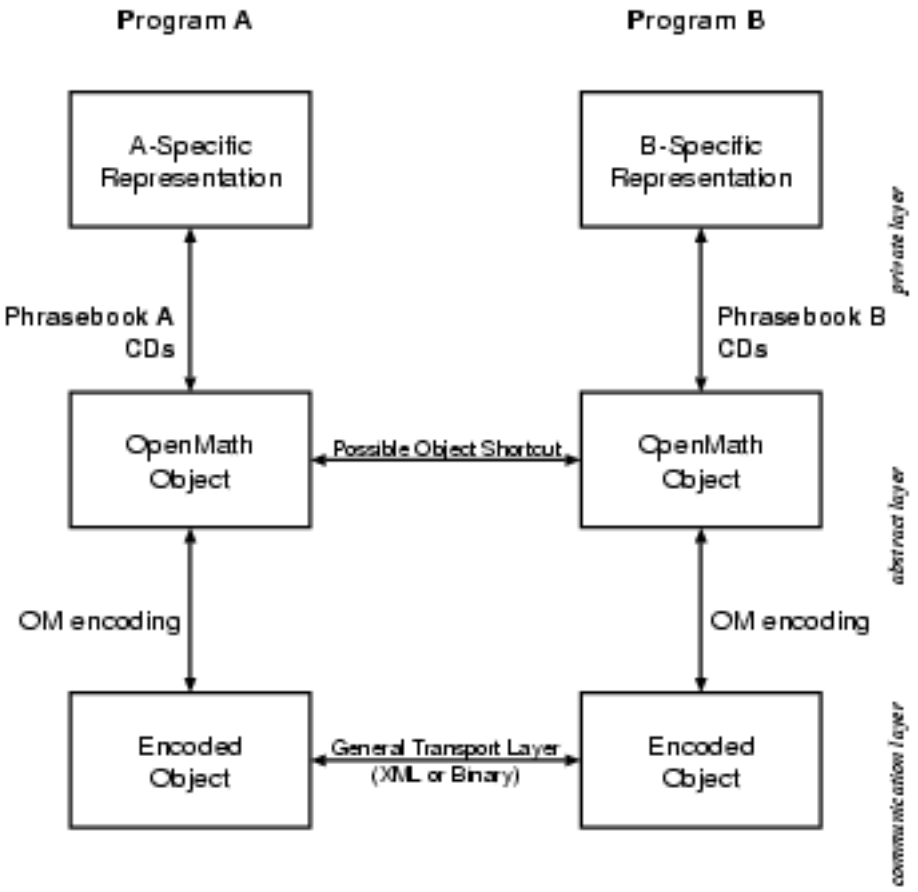

Figure 1.1: The OpenMath Architecture

1.4 Additional Files

Several additional files are related to Content Dictionaries. Signature Dictionaries 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 a specific set of Content Dictionaries as a CD-Group.

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

1.5 Phrasebooks

The conversion of an OpenMath object to/from the internal representation in a software application is performed by an interface program called a *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 $\sin(x)$. 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 2

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 (2.1) and then give a more informal description (2.2).

2.1 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.

2.1.1 Basic OpenMath objects

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 [Iee].

(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 *symbol name*, a *Content Dictionary name*, and (optionally) a *Content Dictionary base URI*, The name of a symbol is a sequence of characters matching the regular expression described in 2.3. 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 2.3) and, optionally, a unique prefix called a *cdbase* which is used to disambiguate multiple Content Dictionaries of the same name. There are other properties of the symbol that are not explicit in these fields but whose values may be obtained by inspecting the Content Dictionary specified. These include the symbol definition, formal properties and examples and, optionally, a *Role* which is a restriction on where the symbol may appear in an OpenMath object. The possible roles are described in 2.1.4.

(vi) Variable. A Variable must have a *name* which is a sequence of characters matching a regular expression, as described in 2.3.

2.1.2 Derived OpenMath Objects

Derived OpenMath objects are currently used as a way by which non-OpenMath data is embedded inside an OpenMath object. A derived OpenMath object is built as follows:

(i) If A is *not* an OpenMath object, then **foreign**(A) is an OpenMath *foreign object*. An OpenMath foreign object may optionally have an *encoding* field which describes how its contents should be interpreted.

2.1.3 OpenMath Objects

OpenMath objects are built recursively as follows.

(i) Basic OpenMath objects are OpenMath objects. (Note that derived OpenMath objects are *not* OpenMath objects, but are used to construct OpenMath objects as described below.)

(ii) If $A_1,\ldots,\ A_n$ $(n > 0)$ are OpenMath objects, then

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

is an OpenMath *application object*.

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

$$\textbf{attribution}(A, A_1S_1, \ldots, A_nS_n)$$

is an OpenMath *attribution object*. A is the object *stripped of attributions*. If $S_1,\ldots,\ S_n$ are referred to as *keys* and $A_1,\ldots,\ A_n$ as their associated *values*. If, after recursively applying stripping to remove attributions, the resulting un-attributed object is a variable, the original attributed object is called an *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

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

is an OpenMath *binding object*.

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

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

is an OpenMath *error object*.

OpenMath objects that are contstructed via rules (ii) to (v) are jointly called *compound OpenMath objects*

2.1.4 OpenMath Symbol Roles

We say that an OpenMath symbol is used to *construct* an OpenMath object if it is the first child of an OpenMath application, binding or error object, or an even-indexed child of an OpenMath attribution object (i.e. the *key* in a *(key, value)* pair). The *role* of an OpenMath symbol is a restriction on how it may be used to construct a compound OpenMath object and, in the case of the key in an attribution object, a clarification of how that attribution should be interpreted. The possible roles are:

(*i*) *binder* The symbol may appear as the first child of an OpenMath binding object.

(*ii*) *attribution* The symbol may 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, without changing the meaning.

(*iv*) *error* The symbol may appear as the first child of an OpenMath error object.

(*v*) *application* The symbol may appear as the first child of an OpenMath application object.

(*vi*) *constant* The symbol cannot be used to construct an OpenMath compound object.

A symbol cannot have more than one role and cannot be used to construct a compound OpenMath object in a way which requires a different role (using the definition of construct given earlier in this section). This means that one cannot use a symbol which binds some variables to construct, say, an application object. However it does not prevent the use of that symbol as an *argument* in an application object (where by argument we mean a child with index greater than 1).

If no role is indicated then the symbol can be used anywhere. Note that this is not the same as saying that the symbol's role is *constant*.

2.2 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 [Con03a].

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.

Symbols are uniquely defined by the Content Dictionary in which they occur and by a name. The form of these definitions is explained in **??**. 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).

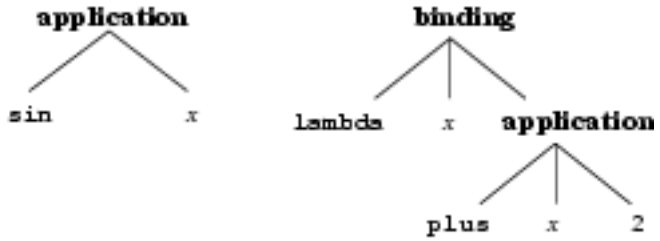

Figure 2.1: The OpenMath application and binding objects for $\lambda x.x + 2$ in tree-like notation.

Derived OpenMath objects are constructed from non-OpenMath data. They differ from bytearrays in that they can have any structure. Currently there is only 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. They may also appear in OpenMath error objects, for example to allow an application to report an error in processing such an object.

The four following constructs can be used to make compound OpenMath objects out of basic or derived OpenMath objects.

Application constructs an OpenMath object from a sequence of one or more OpenMath objects. The first child of an application is referred to as its "head" while the remaining objects are called its "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 suitable Content Dictionary, 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 $\frac{1}{2}$, is represented by the OpenMath application object **application**$(\mathit{Rational}, 1, 2)$. 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*:

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

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.

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

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.

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

is semantically equivalent to:

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

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

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

$$\mathbf{attribution}(\mathbf{attribution}(A, S_1A_1, \ldots, S_hA_h), S_{h+1}A_{h+1}, \ldots, S_nA_n)$$

is semantically equivalent to a single attribution, that is

$$\mathbf{attribution}(A, S_1A_1, \ldots, S_nA_n)$$

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 role *attribution*, then replacement of the attributed object by the object itself is not harmful and preserves the semantics. When the key has role *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 role specification then attribution is acting as adornment annotation.

Objects can be decorated in a multitude of ways.

An example of the use of an adornment attribution would be to indicate the colour in which an OpenMath object should be displayed, for example $\mathbf{attribution}(A, colour, red)$. Note that both A and red are arbitary OpenMath objects whereas *color* is a symbol. An example of the use of a semantic attribution would be to indicate the type of an object. For example the object $\mathbf{attribution}(A, type, t)$ represents the judgment stating that object A has type t. Note that both A and t are arbitary OpenMath objects whereas *type* is a symbol.

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: $\mathbf{error}(S)$.

2.3 Names

The names of symbols, variables and content dictionaries must conform to the production Name specified in the following grammar (which is identical to that for XML names in XML 1.1, [Con04b]). Informally speaking, a name is a sequence of Unicode [Con03a] characters which begins with a letter and cannot contain certain punctuation and combining characters. The notation #x... represents the hexadecimal value of the encoding of a Unicode character. Some of the character values or *code points* in the following productions are currently unassigned, but this is likely to change in the future as Unicode evolves [1]

```
Name          ⟶NameStartChar (NameChar)*
NameStartChar⟶":"| [A−Z] | "_"| [a−z] | [#xC0−#xD6] | [#xD8−#xF6] |
              ⟶[#xF8−#x2FF] | [#x370−#x37D] | [#x37F−#x1FFF] |
              ⟶[#x200C−#x200D] | [#x2070−#x218F] | [#x2C00−#x2FEF] |
              ⟶[#x3001−#xD7FF] | [#xF900−#xFDCF] | [#xFDF0−#xFFFD] |
              ⟶[#x10000−#xEFFFF]
NameChar      ⟶NameStartChar | "−"| "."| [0−9] | #xB7 | [#x0300−#x036F] |
              ⟶[#x203F−#x2040]
```

CD Base A cdbase must conform to the grammar for URIs described in [IETgu]. 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.

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 [Pos] filename, namely:

```
Name       ⟶(PosixLetter | '_') (Char)*
Char       ⟶PosixLetter | Digit | '.'| '−'| '_'
PosixLetter⟶'a'| 'b'| ... | 'z'| 'A'| 'B'| ... | 'Z'
```

Canonical URIs for Symbols To facilitate the use of OpenMath within a URI-based framework (such as RDF [Con04d] or OWL [Con04c]), we provide the following scheme for constructing a canonical URI for an OpenMath Symbol:

URI = cdbase−value + '/' + cd−value + '#' + name−value

So for example the URI for the symbol with cdbase `http://www.openmath.org/cd`, cd transc1 and name sin is:

http://www.openmath.org/cd/transc1#sin

In particular, this now allows us to refer uniquely to an OpenMath symbol from a MathML document [Con03b]:

```
<mathml:csymbol xmlns:mathml="http://www.w3.org/1998/Math/MathML/"
                definitionURL="http://www.openmath.org/cd/transc1#sin">
  <mo> sin </mo>
</csymbol>
```

[1] We note that in XML 1 the name production explicitly listed the characters that were allowed, so all the characters added in versions of Unicode after 2.0 (which amounted to tens of thousands of characters) were not allowed in names.

2.4 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 3

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 (inter-process communications over a network is an example).

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

3.1 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 easily be 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.

3.1.1 A Schema for the XML Encoding

The XML encoding of an OpenMath object is defined by the Relax NG schema [Spece] given below. Relax NG has a number of advantages over the older XSD Schema format [Cony], 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 **??**, 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 **??**.

Standard tools exist for generating a DTD or an XSD schema from a Relax NG Schema. Examples of such documents are given in **??**, respectively.

```
# RELAX NG Schema for OpenMath 2


default namespace om = "http://www.openmath.org/OpenMath"
```

```
start = OMOBJ

# OpenMath object constructor
OMOBJ = element OMOBJ { compound.attributes,
                        attribute version { xsd:string }?,
                        omel }


# Elements which can appear inside an OpenMath object
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))}


cdbase = attribute cdbase { xsd:anyURI}?

# attributes common to all elements
common.attributes = (attribute id { xsd:ID })?

# attributes common to all elements that construct compount OM objects.
compound.attributes = common.attributes,cdbase

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

# variable
OMV = element OMV { common.attributes,
                    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 }

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

# apply constructor
OMA = element OMA { compound.attributes, omel+ }
```

```
# binding constructor
OMBIND = element OMBIND { compound.attributes, omel, OMBVAR, omel }

# variables used in binding constructor
OMBVAR = element OMBVAR { common.attributes, omvar+ }

# error constructor
OME = element OME { common.attributes, OMS, (omel|OMFOREIGN)* }

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

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

# foreign constructor
OMFOREIGN = element OMFOREIGN {
    compound.attributes, attribute encoding {xsd:string}?,
   (omel|notom)* }

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

# reference constructor
OMR = element OMR { common.attributes,
                    attribute href { xsd:anyURI }
                  }
```

Note: This schema specifies names as being of the xsd:NCName type. At the time of writing, W3C Schema types are defined in terms of XML 1 [Con98]. This limits the characters allowed in a name to a subset of the characters available in Unicode 2.0, which is far more restrictive than the definition for an OpenMath name given in 2.3. It is expected that W3C Schema types will be augmented to match the new XML 1.1 recommendation [Con04b], but for portability reasons applications should avoid using the new XML 1.1 name characters unless they are absolutely required. The XML 1.1 specification has a useful appendix giving advice on good strategies to use when naming identifiers.

3.1.2 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. In previous versions of this standard this attribute did not exist, so any OpenMath object without such an attribute must conform to version 1 (or equivalently 1.1) of the OpenMath standard. Objects which conform to the description given in this document should have version="2.0".

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. If a symbol does not have an explicit cdbase attribute, then it inherits its cdbase from the first ancestor in the XML tree with one, should such an element exist. In this document we have tended to omit the cdbase for clarity.

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.

As described in 2.3, 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. For example the URI for the above symbol is `http://www.openmath.org/cd/transc1\#sin`.

Note that the role attribute described in 2.1.4 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"/>

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]+)?([eE](-?)[0-9]+)?

or one of the special values: INF, -INF or NaN.

The value of hex is a base 16 representation of the 64 bits of the ieee Double. Thus the number represents mantissa, exponent, and sign from lowest to highest bits using a least significant byte ordering. This consists of a string of 16 digits 0-9, A-F.

For example, both <OMF dec="1.0e−10"/> and <OMF hex="3DDB7CDFD9D7BDBB"/> are valid representations of the floating point number 1×10^{-10}.

The symbols INF, −INF and NaN represent positive and negative infinity, and *not a number* as defined in [Iee]. Note that while infinities have a unique representation, it is possible for NaNs to contain extra information about how they were generated and if this informations is to be preserved then the hexadecimal representation must be used. For example <OMF hex="FFF8000000000000"/> and <OMFhex="FFF8000000000001"/> are both hexadecimal representations of NaNs.

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 2045 [BF96]. 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 is used for padding at the end of the data). All line breaks and carriage return, space, form feed and horizontal tabulation characters are ignored. The reader is referred to [BF96] for more detailed information.

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 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**$(\lambda, x, \textbf{application}(\sin, x))$is:

```
<OMBIND>
  <OMS cd="fns1" name="lambda"/>
  <OMBVAR><OMV name="x"/></OMBVAR>
  <OMA>
    <OMS 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="annotations1" name="presentation-form"/>
    <OMFOREIGN encoding="MathML-Presentation">
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <mi>sin</mi><mfenced><mi>x</mi></mfenced>
      </math>
    </OMFOREIGN>
  </OMATP>
  <OMA>
   <OMS 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="annotations1" name="presentation-form"/>
    <OMFOREIGN encoding="text/x-latex">\sin(x)</OMFOREIGN>
  </OMATP>
  <OMA>
    <OMS cd="transc1" name="sin"/>
    <OMV name="x"/>
  </OMA>
</OMATTR>
```

For a discussion on the use of the encoding attribute see **??**.

Errors are encoded using the OME element. The error whose symbol is s and whose arguments are the OpenMath objects or OpenMath derived objects $e_1, \ldots, e_n$ 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 $\mathbf{error}(DivisionByZero\mathbf{application}(divide, x, 0))$ 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>
```

If a mathml Content Dictionary contained an unhandled_csymbol symbol, then an OpenMath to MathML translator might return an error such as:

```
<OME>
  <OMS cd="mathml" name="unhandled_csymbol"/>
  <OMFOREIGN encoding="MathML-Content">
    <mathml:csymbol xmlns:mathml="http://www.w3.org/1998/Math/MathML/"
                    definitionURL="http://www.nag.co.uk/Airy#A">
      <mathml:mo>Ai</mathml:mo>
    </mathml:csymbol>
  </OMFOREIGN>
 </OME>
```

Note that it is possible to embed fragments of valid OpenMath inside an OMFOREIGN element but that it cannot contain invalid OpenMath. In addition, the arguments to an OMERROR must be well-formed XML. If an application wishes to signal that the OpenMath it has received is invalid or is not well-formed then the offending data must be encoded as a string. For example:

```
<OME>
  <OMS cd="parser" name="invalid_XML"/>
  <OMSTR>
    &ltOMA> <OMS name="cos" cd="transc1">
      <OMV name="v"> </OMA>
  </OMSTR>
 </OME>
```

Note that the "¡" and "¿" characters have been escaped as is usual in an XML document.

References OpenMath integers, floating point numbers, character strings, bytearrays, applications, binding, attributions can also be encoded as an empty OMR element with an href attribute whose value is the value of a URI referencing an id attribute of an OpenMath object of that type. The OpenMath element represented by this OMR reference is a copy of the OpenMath element referenced href attribute. Note that this copy is *structurally equal*, but not identical to the element referenced. These URI refererences will often be relative, in which case they are resolved using the base URI of the document containing the OpenMath.

For instance, the OpenMath object

```
<math id="nestedap" display="block">
  <mrow>
    <mi mathvariant="bold">application</mi>
    <mrow>
      <mo fence="true">(</mo>
      <mrow>
        <mi>f</mi>
        <mo separator="true">,</mo>
        <mi mathvariant="bold">application</mi>
        <mrow>
          <mo fence="true">(</mo>
          <mrow>
            <mi>f</mi>
            <mo separator="true">,</mo>
            <mi mathvariant="bold">application</mi>
            <mrow>
              <mo fence="true">(</mo>
              <mrow><mi>f</mi><mo separator="true">,</mo><mi>a</mi><mo separator="true">,</mo
```

```
          <mo fence="true">)</mo>
        </mrow>
        <mo separator="true">,</mo>
        <mi mathvariant="bold">application</mi>
        <mrow>
          <mo fence="true">(</mo>
          <mrow><mi>f</mi><mo separator="true">,</mo><mi>a</mi><mo separator="true">,</mo
          <mo fence="true">)</mo>
        </mrow>
        <mo fence="true">)</mo>
      </mrow>
      <mo separator="true">,</mo>
      <mi mathvariant="bold">application</mi>
      <mrow>
        <mo fence="true">(</mo>
        <mrow>
          <mi>f</mi>
          <mo separator="true">,</mo>
          <mi mathvariant="bold">application</mi>
          <mrow>
            <mo fence="true">(</mo>
            <mrow><mi>f</mi><mo separator="true">,</mo><mi>a</mi><mo separator="true">,</m
            <mo fence="true">)</mo>
          </mrow>
          <mo separator="true">,</mo>
          <mi mathvariant="bold">application</mi>
          <mrow>
            <mo fence="true">(</mo>
            <mrow><mi>f</mi><mo separator="true">,</mo><mi>a</mi><mo separator="true">,</m
            <mo fence="true">)</mo>
          </mrow>
          <mo fence="true">)</mo>
        </mrow>
      </mrow>
      <mo fence="true">)</mo>
    </mrow>
  </mrow>
</mrow>
</math>
```

can be encoded in the XML encoding as either one of the XML encodings given in 3.1 (and some intermediate versions as well).

3.1.3 Some Notes on References

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 3.1, the OMA element with id="t1" and also the second OMR dominate the OMA element with id="t11".

Figure 3.1: Shared vs. unshared representations

```
<OMOBJ version="2.0"> <OMOBJ version="2.0">
  <OMA> <OMA>
    <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 href="#t11" />
        <OMV name="f" />
        <OMV name="a" />
        <OMV name="a" />
      </OMA>
    </OMA> </OMA>
    <OMA> <OMR 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 3.2: Sharing between OpenMath objects (A cycle of order 2).

```
<OMOBJ version="2.0"> <OMOBJ version="2.0">
  <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>
```

An Acyclicity Constraint

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

Consider for instance the following (illegal) XML representation

```
<OMOBJ version="2.0">
  <OMA id="foo">
    <OMS cd="arith1" name="divide"/>
    <OMI>1</OMI>
    <OMA>
      <OMS 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 ¡math display="block"¿ ¡mfrac¿ ¡mn¿1¡/mn¿ ¡mrow¿ ¡mn¿1¡/mn¿ ¡mo¿+¡/mo¿ ¡mfrac¿ ¡mn¿1¡/mn¿ ¡mrow¿ ¡mn¿1¡/mn¿ ¡mo¿+¡/mo¿ ¡mfrac¿¡mn¿1¡/mn¿¡mi¿...¡/mi¿¡/mfrac¿ ¡/mrow¿ ¡/mfrac¿ ¡/mrow¿ ¡/mfrac¿ ¡/math¿ this would correspond to an infinite tree of applications, which is not admitted by the structure of OpenMath objects described in 2.

Note that the acyclicity constraints is not restricted to such simple cases, as the example in 3.2 shows.

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.

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 cd="fns1" name="lambda"/>
  <OMBVAR><OMV name="X"/></OMBVAR>
  <OMA>
```

```
    <OMV name="f"/>
    <OMBIND id="inner">
      <OMS cd="fns1" name="lambda"/>
      <OMBVAR><OMV name="X"/></OMBVAR>
      <OMR id="copy" href="#orig"/>
    </OMBIND>
    <OMA id="orig"><OMV name="g"/><OMV name="X"/></OMA>
  </OMA>
</OMBIND>
```

it represents the OpenMath object

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

which has two sub-terms of the form $\mathbf{application}(g, X)$, 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

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

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

3.1.4 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 [Con99] or xml:lang attributes to specify the language used in strings [Con04b].

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 schema specified above.

3.2 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).

3.2.1 A Grammar for the Binary Encoding

3.3 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. Finally, *empty* stands for the empty list of tokens.

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 2.3, for *uri* it is the grammar for URIs in [IETgu].

start	⟶	[24] object [25]	| [24+64] [m] [n] object [25]
object	⟶	basic	| compound
	|	cdbase	| foreign
	|	reference	
basic	⟶	integer	| float
	|	variable	| symbol
	|	string	| bytearray
integer	⟶	[1] [_]	| [1+64] [n] id:n [_]
	|	[1+32] [_]	
	|	[1+128] {_}	| [1+64+128] n id:n {_}
	|	[1+32+128] {_}	
	|	[2] [n] [_]digits:n	| [2+64] [n] [m] [_]digits:n id:m
	|	[2+32] [n] [_]digits:n	
	|	[2+128] n [_]digits:n	| [2+64+128] n n [_]digits:n id:n
	|	[2+32+128] n [_]digits:n	
float	⟶	[3] {_}{_}	| [3+64] [n] id:n {_}{_}
			| [3+64+128] n id:n {_}{_}
variable	⟶	[5] [n] varname:n	| [5+64] [n] [m] varname:n id:m
	|	[5+128] n varname:n	| [5+64+128] n m varname:n id:m
symbol	⟶	[8] [n] [m] cdname:n symbname:m	| [8+64] [n] [m] [k] cdname:n symbname:m id:k
	|	[8+128] n m cdname:n symbname:m	| [8+64+128] n m k cdname:n symbname:m id:k
string	⟶	[6] [n] bytes:n	| [6+64] [n] bytes:n
	|	[6+32] [n] bytes:n	
	|	[6+128] n bytes:n	| [6+64+128] n m bytes:n id:m
	|	[6+32+128] n bytes:n	
	|	[7] [n] bytes:2n	| [7+64] [n] [m]bytes:$n$id:m
	|	[7+32] [n] bytes:2n	
	|	[7+128] n bytes:2n	| [7+64+128] n m bytes:2n id:m
	|	[7+32+128] n bytes:2n	
bytearray	⟶	[4] [n] bytes:n	| [4+64] [n] [m] bytes:n id:m
	|	[4+32] [n] bytes:n	
	|	[4+128] n bytes:n	| [4+64+128] n m bytes:n id:m
	|	[4+32+128] n bytes:n	
cdbase	⟶	[9] [n] uri:n	
	|	[9+128] n uri:n	
foreign	⟶	[12] [n] [m] bytes:n bytes:m	| [12+64] [n] [m] [k] bytes:n bytes:m id:k
	|	[12+32] [n] [m] bytes:n bytes:m	
	|	[12+128] n m bytes:n bytes:m	| [12+64+128] n m k bytes:n bytes:m id:k
	|	[12+32+128] n m bytes:n bytes:m	
compound	⟶	application	| binding
	|	attribution	| error
application	⟶	[16] object objects [17]	| [16+64] [m] id:m object objects [17]
			| [16+64+128] m id:m object objects [17]
binding	⟶	[26] object bvars object [27]	| [26+64] [m] id:m object bvars object [27]
			| [26+64+128] m id:m object bvars object [27]
attribution	⟶	[18] attrpairs object [19]	| [18+64] [m] id:m attrpairs object [19]
			| [18+64+128] m id:m attrpairs object [19]
error	⟶	[22] symbol objects [23]	| [22+64] [m] id:m symbol objects [23]
			| [22+64+128] m id:m symbol objects [23]
attrpairs	⟶	[20] pairs [21]	| [20+64] [m] id:m pairs [21]
			| [20+64+128] m id:m pairs [21]
pairs	⟶	symbol object	
	|	symbol object pairs	
bvars	⟶	[28] vars [29]	| [28+64] [m] id:m vars [29]
			| [28+64+128] m id:m vars [29]
vars	⟶	*empty*	| attrvar vars
attrvar	⟶	variable	
	|	[18] attrpairs attrvar [19]	| [18+64] [m] id:m attrpairs attrvar [19]
			| [18+64+128] m id:m attrpairs attrvar [19]
objects	⟶	*empty*	| object objects
reference	⟶	internal_reference	| external_reference
internal_reference	⟶	[30] [_]	| [30+128] {_}
external_reference	⟶	[31] [n] uri:n	| [31+128] n uri:n

Figure 3.3: Grammar of the binary encoding of OpenMath objects.

3.2.2 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. Objects with start token [88] have two additional bytes m and n that characterize the version ($m.n$) of the encoding directly after the start token. This is similar to <OMOBJ **version**="m.n">.

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, two flags, and a status bit. The identifier is stored in the first five bits (1 to 5). Bit 6 is used as a *status bit* which is currently only used for managing streaming of some basic objects. Bits 7 and 8 are the *sharing flag* and the *long flag*. The sharing flag indicates that the encoded object may be shared in another (part of an) object somewhere else (see 3.2.4). Note that if the sharing flag is set (in the right column of the grammar in 3.3, then the encoding includes a representation of an identifier that serves as the target of a reference (internal with token identifier 30 or external with token identifier 31). If the long flag is set, this signifies that the names, strings, and data fields in the encoded OpenMath object are longer than 255 bytes or characters.

The concept of structure sharing in OpenMath encodings and in particular the sharing bit in the binary encoding has been introduced in OpenMath 2 (see section 3.2.4 for details). The binary encoding in OpenMath 2 leaves the tokens with sharing flag 0 unchanged to ensure OpenMath 1 compatibility. To make use of functionality like the version attribute on the OpenMath object introduced in OpenMath 2, the tokens with sharing flag 1 should be used.

To facilitate the streaming of 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). In 3.4 we have represented an integer that is split up into three packets.

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 in 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 or-ed with the base mask bits that can be 0 for base 10 or 0x40 for base 16 or 0x80 for "base 256". It is followed by the sequence of digits (as characters for bases 10 and 16 as in the XML encoing, and as bytes for base 256) in their natural order. For example, the decimal number 8589934592 (2^{33}) is encoded as

0x02 0x0A 0x2B 0x38 0x35 0x38 0x39 0x39 0x33 0x34 0x35 0x39 0x32

and the hexadecimal number xffffff1 is encoded as

0x02 0x08 0x6b 0x66 0x66 0x66 0x66 0x66 0x66 0x66 0x31

in the base 16 character encoding and as

0x02 0x04 0xFF 0xFF 0xFF 0xFI

Byte	Hex	Meaning	Byte	Hex	Meaning
1	22	begin streamed big integer tag	7	2B	sign + (disregarded)
2	FF	255 digits in packet	8	...	the 255 digits as characters
3	2B	sign +	9	2	begin final big integer tag
4	...	the 255 digits as characters	10	42	68 digits in packet
5	22	begin streamed big integer tag	11	2B	sign + (disregarded)
6	FF	255 digits in packet	12	...	the 68 digits as characters

Figure 3.4: Streaming a large Integer in the Binary Encoding.

in the byte encoding (base 256).

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 or the symbol name is greater than or equal to 256. 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 id (if the shared bit was set) as a byte (if the long flag was not set) or a four byte integer (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 id.

Variables aare 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. 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.

Floating-point number are encoded using the floating-point number tags (token identifier 3) followed by eight bytes that are the IEEE 754 representation [Iee], 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-8859-1 (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 utf-16 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 assumed to have the same encoding for every packet. They 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 and the streaming bit set for dividing it up into packets. Then, there is the number n of bytes used to encode the encoding, and the number m of bytes used to encode the foreign object. n and m are represented as a byte (if the long flag was not set) or a four byte integer (in network byte order). These numbers are followed by an n-byte representation of the encoding attribute and an m byte sequence of bytes encoding the foreign object in their normal order (we call these the payload bytes). The encoding attribute is encoded in utf-8.

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

Note that the foreign object is encoded as a stream of bytes, not a stream of characters. Character based formats (including XML based formats) should be encoded in utf-8 to produce a stream of bytes to use as the payload of the foreign object.

cdbase scopes are encoded using the token identifier 9. The purpose of these scoping devices is to associate a cdbase with an object. The start token [9] (or [137] if the long flag is set) is followed by a single-byte (or 4-byte- if the long flag is set) number n and then by a seqence of n bytes that represent the value of the cdbase attribute (a URI) in utf-8 encoding. This is then followed by the binary encoding of a single object: the object over which this cdbase attribute has scope.

Applications are encoded using the application tags (token identifiers 16 and 17). More precisely, the application of E_0 to E_1, ..., 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, ... V_n in C is encoded as the binding tag (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, ..., 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).

Attributions are encoded using the attribution tags (token identifiers 18 and 19). More precisely, attribution of the object E with (E_1S_1, ..., E_nS_n) pairs (where S_i are the attributes) is encoded as the attributed object tag (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 tag (token identifier 21), followed by the encoding of E, followed by the end attributed object tag (token identifier 19).

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

Byte	Hex	Meaning	Byte	Hex	Meaning	Byte	Hex	Meaning
1	58	begin object tag	19	10	begin application tag	40	10	begin application tag
2	2	version 2.0 (major)	20	08	symbol tag	41	48	symbol tag (with share bit on)
3	0	version 2.0 (minor)	21	06	cd length	42	01	reference to second symbol seen (arith1:plus
4	10	begin application tag	22	04	name length	43	45	variable tag (with share bit on)
5	08	symbol tag	23	61	a (cd name begin	44	00	reference to first variable seen (x)
6	06	cd length	24	72	r .	45	05	variable tag
7	05	name length	25	69	i .	46	01	name length
8	61	a (cd name begin	26	74	t .	47	7a	z (variable name)
9	72	r .	27	68	h .	48	11	end application tag
10	69	i .	28	31	1 .)	49	11	end application tag
11	74	t .	29	70	p (symbol name begin	50	11	end application tag
12	68	h .	30	6c	l .			
13	31	1 .)	31	75	u .			
14	74	t (symbol name begin	32	73	s .)			
15	69	i .	33	05	variable tag			
16	6d	m .	34	01	name length			
17	65	e .	35	78	x (name)			
18	73	s .)	36	05	variable tag			
			37	01	name length			
			38	79	y (variable name)			
			39	11	end application tag			

Figure 3.5: A Simple example of the OpenMath binary encoding.

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.

3.2.3 Example of Binary Encoding

As a simple example of the binary encoding, we can consider the OpenMath object

$$\mathbf{application}(times, \mathbf{application}(plus, x, y), \mathbf{application}(plus, x, z))$$

It is binary encoded as the sequence of bytes given in 3.5.

3.2.4 Sharing

OpenMath 2 introduced a new sharing mechanism, described below. First however we describe the original OpenMath 1 mechanism.

Sharing in Objects beginning with the identifier [24]

This form of sharing is deprecated but included for backwards compatibility with OpenMath 1. It 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 encoded as 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).

Byte	Hex	Meaning	Byte	Hex	Meaning	Byte	Hex	Meaning
1	58	begin object tag	12	50	begin application tag (shared)	23	1E	short reference
2	2	version 2.0 (major)	13	05	variable tag	24	00	to the first shared object
3	0	version 2.0 (minor)	14	01	variable length	25	11	end application tag
4	10	begin application tag	15	66	f (variable name)	26	1E	short reference
5	05	variable tag	16	05	variable tag	27	00	to the second shared object
6	01	variable length	17	01	variable length	28	11	end application tag
7	66	f (variable name)	18	61	a (variable name)			
8	50	begin application tag (shared)	19	05	variable tag			
9	05	variable tag	20	01	variable length			
10	01	variable length	21	61	a (variable name)			
11	66	f (variable name)	22	11	end application tag			

Figure 3.6: A binary encoding of the OpenMath object from 3.1.

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 tag [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 3.6 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. 3.6 shows the binary encoding of the object in 3.1 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 the conversion from the XML to the binary encoding the identifiers on the objects are not preserved. Moreover, even though the XML encoding allows references across objects, as in 3.2, the binary encoding does not (the binary encoding has no notion of a multi-object collection, while in the XML encoding this would naturally correspond to e.g. the embedding of multiple OpenMath objects into a single XML document).

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

3.2.5 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 3.1.3), 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.

Relation to the OpenMath 1 binary encoding

The OpenMath 2 binary encoding significantly extends the OpenMath 1 binary encoding to accommodate 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 constructed so as to maximise internal sharing (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 allows it to be represented more compactly.

3.3 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.

Bibliography

[BF96] N. Borenstein and N. Freed. *MIME (Multipurpose Internet Mail Extensions) Part One: Format of Internet Message Bodies*. RFC: 2045. 1996. URL: http://www.ietf.org/rfc/rfc2045.txt.

[Con02] OpenMath Consortium. *OpenMath Version 1.1*. 2002. URL: http://www.openmath.org/standard/om11/.

[Con03a] Unicode Consortium. *The Unicode Standard: Version 4.0.0*. Addison-Wesley, 2003. ISBN: ISBN 0-321-18578-1. URL: http://www.unicode.org/versions/Unicode4.0.0.

[Con03b] World Wide Web Consortium. *Mathematical Markup Language (MathML) 2.0 Specification (Second Edition)*. 2003. URL: http://www.w3.org/TR/MathML2/.

[Con04a] OpenMath Consortium. *OpenMath Primer*. 2004. URL: http://www.openmath.org/standard/primer/.

[Con04b] World Wide Web Consortium. *Extensible Markup Language (XML) 1.1., W3C Recommendation REC-xml11-20040204*. 2004. URL: http://www.w3.org/TR/2004/REC-xml11-20040204/.

[Con04c] World Wide Web Consortium. *OWL Web Ontology Language Overview*. 2004. URL: http://www.w3.org/TR/2004/REC-owl-features-20040210/.

[Con04d] World Wide Web Consortium. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. 2004. URL: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[Con98] World Wide Web Consortium. *Extensible Markup Language (XML) 1.0*. W3C Recommendation REC-xml-19980210. 1998. URL: http://www.w3.org/TR/1998/REC-xml-19980210.

[Con99] World Wide Web Consortium. *Namespaces in XML*. 1999. URL: http://www.w3.org/TR/REC-xml-names/.

[Cony] World Wide Web Consortium. *XML Schema Part 1: Structures & Part 2: Datatypes*. May 2001. URL: http://www.w3.org/TR/xmlschema-1/.

[Iee] *IEEE Standard for binary Floating-Point Arithmetic, ANSI/IEEE Standard 754*. 1985. URL: http://standards.ieee.org/reading/ieee/std_public/description/busarch/754-1985_desc.html.

[IETgu] IETF. *RFC 2396 - Uniform Resource Identifiers (URI): Generic Syntax*. August 1998. URL: http://www.ietf.org/rfc/rfc2396.txt.

[Pos] *Std 1003.1, 2003 Edition, The Open Group Base Specifications Issue 6*. 2003. URL: http://www.unix.org/version3/ieee_std.html.

[Spece] OASIS Committee Specification. *RELAX NG Specification*. December 2001. URL: http://www.oasis-open.org/committees/relax-ng/spec-20011203.html.