Version: 1.1b
Date: October 24 2002

The OpenMath Standard

The OpenMath Esprit Consortium

Editors

O.Caprotti, D.P.Carlisle and A.M.Cohen

© 2002 The OpenMath Consortium (24.969)

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*.

Change-marked edition notes

This edition contains colour coded change markings relative to the OpenMath 1.0 document…

- New text is marked in green.
- Deleted text is marked in red.

Sections with new text

- 3.2 Further Description of OpenMath Objects
- 4.1.1 A Grammar for the xml Encoding

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 [3] 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 [6].

In 1996 a group of European participants in *OpenMath* decided to bid for funding under the European 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 [10]. 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.

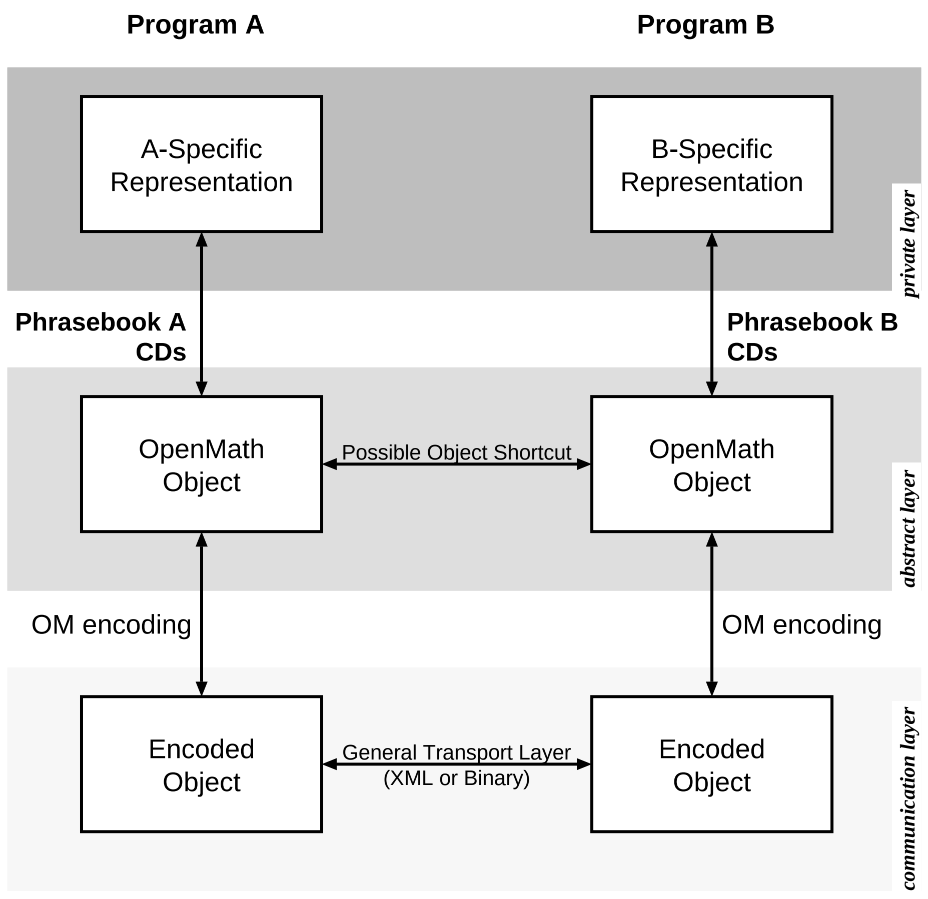

Figure 2.1: The *OpenMath* Architecture

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 *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.

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 compuational 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 at an informal level (Section 3.2) and next by means of an abstract grammar description (Section 3.1).

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 two fields of information, a *name* and a *Content Dictionary.* Each is a sequence of characters matching a regular expression, as described below.

(vi) Variable.

A Variable consists of a *name* which is a sequence of characters matching a regular expression, as described below.

3.1.2 Compound *OpenMath* Objects

OpenMath objects are built recursively as follows.

(i) Basic *OpenMath* objects are *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, $A_1, \ldots, A_n$, $(n > 0)$ are *OpenMath* objects, then

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

is an *OpenMath attribution object* and A is the object *stripped of attributions.* 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.*

3.2 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 [13].

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. In definition in Section 3.1 we have left this information implicit. However, it should be kept in mind that all symbols appearing in an *OpenMath* object are defined in a Content Dictionary. 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-commutativeset theoretic union in a Content Dictionary `set1` but another Content Dictionary, `multiset1` might define a symbol `union` as the union of multi-sets. The name of a symbol can only contain alphanumeric characters and underscores. More precisely, a symbol name matches the following regular expression:

`[A-Za-z]` `[A-Za-z0-9_]`*

Notice that these symbol names are case sensitive. *OpenMath recommends* that symbol names should be no longer than 100 characters.

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). Plain variable names are restricted to use a subset of the printable ASCII characters. Formally the names must match the regular expression:

``[A-Za-z0-9=+(),-./:?!#$%*;=@[]^_`{|}]``+

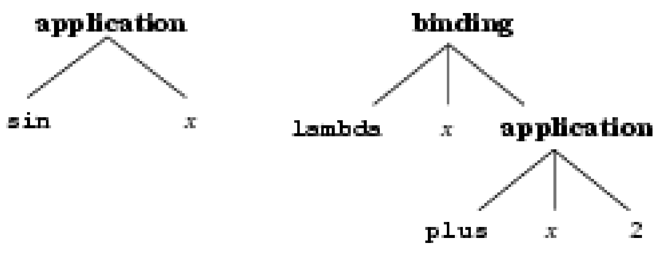

Figure 3.1: The *OpenMath* application and binding objects for $sin(x)$ and $\lambda x.x + 2$ in tree-like notation.

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 $\mathbf{application}(sin, x)$ 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 $\mathbf{application}(Rational, 1, 2)$. 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`:

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

Binding of several variables as in:

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

is semantically equivalent to composition of binding of a single variable, namely

$$\mathbf{binding}(B, v_1, (\mathbf{binding}(B, v_2, (\ldots, \mathbf{binding}(B, v_n, C) \ldots).$$

Note that it follows from this that repeated occurrences of the same variable in a binding operator are allowed. For example the object

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

is semantically equivalent to:

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

so that the outermost binding is actually a constant function (v does not occur free in the body **application**$(\texttt{times}, v, v)$)).

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.

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

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

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.

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

is semantically equivalent to:

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

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

Attribution decorates an object with a sequence of one or more pairs made up of an *OpenMath* symbol, the “attribute”, and an associated *OpenMath* object, the “value of the attribute”. The value of the attribute can be an attribution object itself. As 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.

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.)

Objects can be decorated in a multitude of ways. In [4], typing of *OpenMath* objects is expressed by using an attribution. The object $\textbf{attribution}(A, \mathit{type}\ t)$ represents the judgment stating that object A has type t. Note that both A and t are *OpenMath* objects.

Attribution can act as either annotation, in the sense of adornment, or as modifier. In the former case, replacement of the adorned object by the object itself is probably not harmful (preserves the semantics). In the latter case however, it may very well be. Therefore, attribution in general should by default be treated as a construct rather than as adornment. Only when the CD definitions of the attributes make it clear that they are adornments, can the attributed object be viewed as semantically equivalent to the stripped object.

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 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 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 uses ISO 646:1983 characters [9] (also known as ASCII characters) and is an XML application. Although the XML markup of the encoding uses only ASCII characters, OpenMath strings may use arbitrary Unicode/ISO 10646:1988 characters [13]. It can be used, for example, to send *OpenMath* objects via e-mail, news, cut-and-paste, etc. The texts produced by this encoding can be part of XML documents.

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 the most common character set (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.

4.1.1 A Grammar for the xml Encoding

The XML encoding of an OpenMath object is defined by the DTD given in Figure 4.1 below, with the following additional rules not implied by the XML DTD.

- Comments are permitted only between elements, not within element character data.
- Processing Instructions are only allowed before the OMOBJ element.
- The content of an OMB element, is a valid base64-encoded text.
- The character data forming element content and attribute values matches the regular expressions of Figure 4.2.

In addition, if the XML document encoding the *OpenMath* object is linearised into the XML concrete syntax, the following further constraints apply, which ensure thet the encoding may be read by *OpenMath* applications that may not include a full XML parser.

- The document should use UTF-8 encoding.
-

  Entity and character references should not be used.
- A `<!DOCTYPE` declaration should not be used.
-

  Character references should not be used. As `<!DOCTYPE` is not used, the only entity references that are allowed are the five predefined entity references: `&apos;` ('), `&quot;` ("), `&lt;` (<), `&gt;` (>), `&amp;` (&).
- The XML empty element form `<.../>` should always be used to encode elements such as OMF which are specified in the DTD as being EMPTY. It should never be used for elements that may sometimes be empty, such as OMSTR.

Such a linearisation of an XML encoded *OpenMath* Object would match the match the character based grammar given in Figure 4.2.

The notation used in this section and in Figure 4.2 should be quite straightforward (+ meaning "one or more", ? meaning zero or one, and | meaning "or"). The start symbol of the grammar is "start", "space" stands for the space character, "cr" for the carriage return character, "nl" for the line feed character and "tab" for the horizontal tabulation character.

```
<!-- DTD for OM Objects - sb 29.10.98 -->
<!-- sb 3.2.99 -->

<!--
     general list of embeddable elements
      : excludes OMATP as this is only embeddable in OMATTR
      : excludes OMBVAR as this is only embeddable in OMBIND
-->

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

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

<!ENTITY % omvar "OMV | OMATTR" >

<!-- symbol -->
<!ELEMENT OMS EMPTY>
<!ATTLIST OMS name CDATA #REQUIRED  cd CDATA #REQUIRED >

<!-- variable -->
<!ELEMENT OMV EMPTY>
<!ATTLIST OMV name CDATA #REQUIRED >

<!-- integer -->
<!ELEMENT OMI (#PCDATA) >

<!-- byte array -->
<!ELEMENT OMB (#PCDATA) >

<!-- string -->
<!ELEMENT OMSTR (#PCDATA) >

<!-- floating point -->
<!ELEMENT OMF EMPTY>
<!ATTLIST OMF dec CDATA #IMPLIED  hex CDATA #IMPLIED>

<!-- apply constructor -->
<!ELEMENT OMA (%omel;)+ >

<!-- binding constructor & variable -->
<!ELEMENT OMBIND ((%omel;), OMBVAR, (%omel;)) >
<!ELEMENT OMBVAR (%omvar;)+ >

<!-- error -->
<!ELEMENT OME (OMS, (%omel;)* ) >

<!-- attribution constructor & attribute pair constructor -->
<!ELEMENT OMATTR (OMATP, (%omel;)) >
<!ELEMENT OMATP (OMS, (%omel;))+ >

<!-- OM object constructor -->
<!ELEMENT OMOBJ (%omel;) >
```

Figure 4.1: DTD for the *OpenMath* XML encoding of objects.

S	⟶	(space|tab|cr|nl)+
integer	⟶	(`-` S?)? [0-9]+ (S [0-9]+)* | (`-` S?)? `x` S? [0-9A-F]+ (S [0-9A-F]+)*
cdname	⟶	[a-z][a-z0-9_]*
symbname	⟶	[A-Za-z][A-Za-z0-9_]*
fpdec	⟶	(`-`?)([0-9]+)?(`.`[0-9]+)?(`e`([+-]?)[0-9]+)?
fphex	⟶	[0-9ABCDEF]+
varname	⟶	([A-Za-z0-9+=(),-./:?!#$%*;@[]^_‘{|}])+
base64	⟶	([A-Za-z0-9 +/=] | S)+
char	⟶	*XML Character Data*
symbnameatt	⟶	`name` S? = S? (`"` symbname `"` `|` `’` symbname `’`)
cdnameatt	⟶	`cd` S? = S? (`"` cdname `"` `|` `’` cdname `’`)
varnameatt	⟶	`name` S? = S? (`"` varname `"` `|` `’` varname `’`)
fpdecatt	⟶	`dec` S? = S? (`"` fpdec `"` `|` `’` fpdec `’`)
fphexatt	⟶	`hex` S? = S? (`"` fphex `"` `|` `’` fphex `’`)
PI	⟶	<`?` char `?>`
comment	⟶	<`!-` char `->`
SC	⟶	S+ `|` (comment S)+
start	⟶	(SC `|` PI)* `<OMOBJ` S?`>` S? object S? `</OMOBJ` S?`>`
symbol	⟶	`<OMS` S symbnameatt S cdnameatt S? `/>`
	|	`<OMS` S cdnameatt S symbnameatt S? `/>`
variable	⟶	`<OMV` S varnameatt S? `/>`
	|	`<OMATTR` S?`>` SC? omatp SC? variable SC? `</OMATTR` S?`>`
omatp	⟶	`<OMATP` S?`>` SC? attrs SC? `</OMATP` S?`>`
object	⟶	symbol
	|	variable
	|	`<OMI` S?`>` S? integer S? `</OMI` S?`>`
	|	`<OMF` S fpdecatt S? `/>`
	|	`<OMF` S fphexatt S? `/>`
	|	`<OMSTR` S?`>` char `</OMSTR` S?`>`
	|	`<OMB` S?`>` base64 `</OMB` S?`>`
	|	`<OMA` S?`>` SC? object SC? objects SC? `</OMA` S?`>`
	|	`<OMBIND` S?`>` SC? object SC?
		`<OMBVAR` S?`>` SC? variables SC? `</OMBVAR` S?`>`
		SC? object SC? `</OMBIND` S?`>`
	|	`<OME` S?`>` SC? symbol SC? objects SC? `</OME` S?`>`
	|	`<OMATTR` S?`>` SC? `<OMATP` S?`>` SC? attrs SC? `</OMATP` S?`>`
		SC? object SC? `</OMATTR` S?`>`
attrs	⟶	symbol S? object
	|	symbol S? object S? attrs
objects	⟶	SC?
	|	object SC? objects
variables	⟶	SC?
	|	variable SC? variables

Figure 4.2: Grammar for the XML encoding of *OpenMath* objects.

4.1.2 Description of the Grammar

An encoded *OpenMath* object is placed inside an `OMOBJ` element. This element can contain the elements (and integers) as described above.

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 two XML-attributes `cd` and `name`. 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 name of the Content Dictionary is compulsory, but a future revision of the *OpenMath* standard might introduce a defaulting mechanism. For example, `<OMS cd="transc" name="sin"/>` is the encoding of the symbol named `sin` in the Content Dictionary named `transc`.

Variables are encoded using the `OMV` element, with only one XML-attribute, `name`, whose value is the variable name. The variable name is a subset of the printable ASCII set of characters. In particular, neither spaces nor double-quote `"` are allowed in variable names. 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]+)?(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 (The default encoding is UTF-8[17], although XML encoded OpenMath may be embedded in a containing XML document that specifies alternative encoding in the XML declaration. 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 root 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, ..., 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**$(\texttt{lambda}, x, \textbf{application}(sin, x))$ 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), …, (s_n, e_n) pairs (where s_i are the attributes), it is encoded as `<OMATTR> <OMATP>` S_1 C_1 … 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>
```

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… 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>
```

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 [14] or xml:lang attributes to specify the language used in strings [15]. Also, the encoding used in the larger document may not be UTF-8.

In particular, if *OpenMath* is used with applications that use the XML Namespace Recommendation [14] then they should ensure that *OpenMath* elements are in the namespace `http://www.openmath.org/OpenMath` This is most conveniently achieved by adding the namespace declaration

```
xmlns="http://www.openmath.org/OpenMath"
```

as an attribute to each `OMOBJ` element in the document.

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 4.3 gives a grammar for the binary encoding. 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. *name*:n denotes a sequence of n bytes named *name*. *name*:$2n$ denotes a sequence of $2n$ bytes. “start” is the start symbol of the grammar.

start	⟶	[24] object [25]
object	⟶	integer
	|	float
	|	variable
	|	symbol
	|	string
	|	bytearray
	|	construct
integer	⟶	[1] [_]
	|	[1 + 128] {_}
	|	[2] [n] [_] digits:n
	|	[2 + 128] {n} [_] digits:n
float	⟶	[3] {_} {_}
variable	⟶	[5] [n] varname:n
	|	[5 + 128] {n} varname:n
	|	[5 + 64] [n]
symbol	⟶	[8] [n] [m] cdname:n symbname:m
	|	[8 + 128] {n} {m} cdname:n symbname:m
	|	[8 + 64] [n]
string	⟶	[6] [n] chars:n
	|	[6 + 128] {n} chars:n
	|	[7] [n] chars:2n
	|	[7 + 128] {n} chars:2n
	|	[7 + 64] [n]
bytearray	⟶	[4] [n] bytes:n
	|	[4 + 128] {n} bytes:n
construct	⟶	[16] object objects [17]
	|	[22] symbol objects [23]
	|	[18] attrpairs object [19]
	|	[26] object bvars object [27]
attrpairs	⟶	[20] pairs [21]
pairs	⟶	symbol object
	|	symbol object pairs
bvars	⟶	[28] vars [29]
vars	⟶	attrvar
	|	attrvar vars
attrvar	⟶	variable
	|	[18] attrpairs attrvar [19]
objects	⟶	
	|	object objects

Figure 4.3: Grammar of the binary encoding of *OpenMath* objects.

4.2.2 Description of the Grammar

An *OpenMath* object is encoded as a sequence of bytes starting with the begin object tag (value 24) 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* 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.

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 tag (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 0x38353839393334353932` and xfffffff1 is encoded as `0x02 0x08 0x6b 0x6666666666666631`. Note that it is permitted to encode a "small" integer in any "bigger" format.

Symbols are encoded as the symbol tag (8) with the long flag set if the maximum of the length of the Content Dictionary name and the symbol name is greater than or equal to 256 (note that this should never be the case if the rules on symbols and Content Dictionary names are applied), then followed by the length of the Content Dictionary name as a byte (if the long flag was not set) or a four byte integer (in network byte order) followed by the length of the symbol name 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 Content Dictionary name, followed by the characters of the symbol name.

Variables are encoded using the variable tag (5) with the long flag set if the number of bytes (characters) in 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`.

Floating-point number are encoded using the floating-point number tag (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 contains UTF-16 characters or not. If the string contains only 8 bit characters, it is encoded as the one byte character string tag (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 contains two byte characters, it is encoded as the two byte character string tag (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 characters 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).

Bytearrays are encoded using the bytearray tag (4) with the long flag set if the number of bytes in the number of 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.

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

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

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

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

4.2.2.1 Sharing

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 Implementation Note

A typical implementation of the binary encoding 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).

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.

4.2.4 Example of Binary Encoding

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

```
<OMOBJ>
  <OMA>
    <OMS name="times" cd="arith1"/>
    <OMA>
      <OMS name="plus" cd="arith1"/>
      <OMV name="x"/>
      <OMV name="y"/>
    </OMA>
    <OMA>
      <OMS 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

4.3 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 passed, 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 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 Content Dictionaries

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

Other than Content Dictionary comments (which have no real semantics), Content Dictionaries have been designed to hold two types of information: that which is pertinent to the whole Content Dictionary, and that which is restricted to a particular symbol definition. Specific information pertaining to the symbols like the signature and the defining mathematical properties is conveyed in additional files associated to Content Dictionaries.

Information that is pertinent to the whole Content Dictionary includes:

- The name of the Content Dictionary.
- A description of the Content Dictionary.

- A date when the Content Dictionary is next planned to be reviewed.
- A date on which the Content Dictionary was last edited.
- The current version and revision numbers of the Content Dictionary.
- The status of the Content Dictionary.
- An optional URL for this Content Dictionary.
- An optional list of Content Dictionaries on which this Content Dictionary depends. That is, those named in Examples and FMP in this Content Dictionary.
- An optional comment, possibly containing the author's name.

Information that is restricted to a particular symbol includes:

- The name of the symbol.
- A description of this symbol.
- An optional comment.
- Optional properties that this symbol should obey.
- Optional examples of the use of this symbol.

As mentioned earlier, certain kinds of data pertaining to symbols may be conveyed in files other than a Content Dictionary. In particular, information on signatures according to a type system may be described in *Signature Files* whose format is given in Section 5.4.1. Other information such as presentation forms, extra defining mathematical properties may be associated with Content Dictionaries using files whose format is not specified by this standard. It is expected that a common method of defining the presentation for *OpenMath* symbols is via XSL [16] stylesheets giving transformations to MathML.

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 [11]. *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 Section 5.4.2 below.

5.3 The XML Encoding for Content Dictionaries

Content Dictionaries are XML documents. A valid Content Dictionary document should

- be valid according to the DTD given in Figure 5.1,
- adhere to the extra conditions on the content of the elements given in Section 5.3.2.

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 DTD Specification of Content Dictionaries

The XML DTD for Content Dictionaries is given in Figure 5.1. The allowed elements are further described in the following section.

5.3.2 Further Requirements of an *OpenMath* Content Dictionary

The notion of being a valid Content Dictionary is stronger than merely being successfully parsed by the DTD. This is because the content of the elements, referred to in Figure 5.1 as PCDATA and CDATA, must actually make sense to, say, a Phrasebook designer. In this section we define exactly the format of the elements used in Content Dictionaries.

`CDName` The text occurring in the `CDName` element corresponds to the name of Content Dictionary, and is of the form specified in Chapter 4.

`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 text occurring in the `CDReviewDate` element corresponds to the earliest possible revision date of the Content Dictionary. The date formats should be ISO-compliant in the form YYYY-MM-DD, e.g. 1953-09-26.

`CDDate` The text occurring in the `CDDate` element corresponds to the date of this version of the Content Dictionary. The date formats should be ISO-compliant in the form YYYY-MM-DD, e.g. 1953-09-26.

`CDVersion` The text occurring in the `CDVersion` element corresponds to the version number of the current version of a Content Dictionary. It should be a non negative integer.

In CDs that do not have status *experimental*, CD version numbering should adhere to the following. The version number should be a positive integer.

No changes can 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 this 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 increas should be made to the revision number, as described below. A change such as removing a symbol should not be made, instead a new CD, with a different name should be produced, so as not to invalidate existing objects.

```
<!-- omcd.dtd -->
<!-- ******************************************** -->
<!--                                              -->
<!-- DTD for OpenMath CD                          -->
<!-- (c) EP24969 the ESPRIT OpenMath Consortium    -->
<!-- date = 28.aug.1998                           -->
<!-- author = s.buswell sb@stilo.demon.co.uk      -->
<!--                                              -->
<!-- edited by n.howgrave-graham 30.aug.98        -->
<!-- edited by sb 4.sep.98                        -->
<!-- edited by nh-g 4.sep.98                      -->
<!-- edited by sb 1.nov.98                        -->
<!-- edited by sb 1.nov.98                        -->
<!-- edited by dpc 1999-04-13                     -->
<!-- edited by dpc 1999-05-11  CDDate & CDVersion  -->
<!-- edited by dpc 1999-06-21  Delete Signature & Presentation  -->
<!--                           Force Name to be first child of -->
<!--                           CDDefinition -->
<!--                                              -->
<!--                                              -->
<!-- ******************************************** -->
<!ELEMENT CDComment      (#PCDATA) >
<!ELEMENT CDName         (#PCDATA) >
<!ELEMENT CDURL          (#PCDATA) >
<!ELEMENT CDUses         (CDName)*  >
<!ENTITY % inhel        "(#PCDATA)" >
<!ENTITY % inhel2       "(#PCDATA | OMOBJ)*" >
<!ELEMENT CDReviewDate   %inhel; >
<!ELEMENT CDDate         %inhel; >
<!ELEMENT CDVersion      %inhel; >
<!ELEMENT CDRevision     %inhel; >
<!ELEMENT CDStatus       %inhel; >
<!ELEMENT Description    %inhel; >
<!ELEMENT Name           %inhel; >
<!ELEMENT CMP            %inhel; >
<!-- include dtd for OM objects -->
<!ENTITY  % omobjectdtd SYSTEM "omobj.dtd" >
%omobjectdtd;
<!ELEMENT FMP            (OMOBJ) >             <!-- embedded OM -->
<!ELEMENT Example        %inhel2; >
<!ELEMENT CDDefinition    (Name, Description,
                          (CDComment | Example | FMP | CMP )*) >
<!ELEMENT CD             (CDComment | CDName | Description |
                         CDReviewDate | CDDate |CDVersion |CDRevision |
                         CDStatus | CDURL | CDUses | CDDefinition |
Example)* >
<!-- end of DTD for OM CD -->
```

Figure 5.1: DTD Specification of Content Dictionaries

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

Experimental CDs may expect to have changes such as adding or removing symbols as they are developed, without requiring the name of the CD to be changed.

`CDRevision` The text occurring in the `CDRevision` element corresponds to the revision, or 'minor version number' of the current version of a Content Dictionary. It should be a non negative integer.

Minor changes to a CD that do not warrant the release of a CD with an increased version number should be marked by increasing the revision number specified in this field. When the Cd Version number is increased, the Revision number is normally reset to zero.

`CDStatus` The text occurring in the `CDStatus` element corresponds to the status of Content Dictionary, 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 Content Dictionary kept only for archival purposes).

`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 [12].

`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.

`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.

`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.

`Name` The text occurring in the `Name` element corresponds to the name of the symbol, and is specified as in Chapter 4.

`CMP` The text occurring in the `CMP` element corresponds to a property of the symbol. An application which says it understands a Content Dictionary symbol need not understand a commented property of the symbol.

`FMP` The content of the `FMP` element also corresponds to a property[1] of the symbol, however the content of this element must be a valid *OpenMath* object in the XML encoding. An application which says it understands a Content Dictionary symbol need not understand a formal property of the symbol.

[1]It corresponds to a theorem of a theory in some formal system.

5.4 Additional Information

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

5.4.1 Signature Files

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 [7]. The signature file given in Appendix A.3 is based on this formalism. Using the same mechanism, [5] shows how pure type systems can also be employed to assign types to *OpenMath* symbols.

5.4.1.1 The DTD Specification of Signature Files

Signature Files are XML documents, hence a valid Signature File should

- be valid according to the DTD given in Figure 5.2,
- adhere to the extra conditions on the content of the elements given in Section 5.4.1.2.

Signature files have a header which specifies the Content Dictionary and determines 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 XML encoded *OpenMath* object.

5.4.1.2 Further Requirements of a Signature File

The notion of being a valid Signature File is stronger than merely being successfully parsed by the DTD in Figure 5.2. In this section we define exactly the format of the elements used in Signature Files. Several of the requirements are the same as those on elements of Contents Dictionaries.

`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

```
<!-- omcdsig.dtd -->
<!-- ******************************************** -->
<!--                                              -->
<!-- DTD for OpenMath CD Signatures               -->
<!-- (c) EP24969 the ESPRIT OpenMath Consortium   -->
<!-- David Carlisle 1999-04-13                    -->
<!-- David Carlisle 1999-05-21                    -->
<!-- David Carlisle 1999-06-22                    -->
<!--                                              -->
<!--                                              -->
<!-- ******************************************** -->

<!-- include dtd for OM objects -->
<!ENTITY  % omobjectdtd SYSTEM "omobj.dtd" >
%omobjectdtd;

<!ELEMENT CDSComment      (#PCDATA) >
<!ELEMENT CDSReviewDate    (#PCDATA) >
<!ELEMENT CDSStatus    (#PCDATA) >

<!ELEMENT CDSignatures   (CDComment |CDSComment | CDSReviewDate |
                         CDSStatus | Signature )* >

<!ATTLIST CDSignatures cd CDATA #REQUIRED
                       type CDATA #REQUIRED >

<!ELEMENT Signature      (OMOBJ?) >

<!ATTLIST  Signature  name CDATA #REQUIRED >

<!-- end of DTD for OM CD Signatures -->
```

Figure 5.2: DTD Specification of Signature Files

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 5.3.2.

`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 5.4.1.3 for examples.

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 name="mapsto" cd="sts"/>
  <OMA>
   <OMS 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, $\ldots x_n$", where $x_1, \ldots x_n$ are the members of CDGroup G.

5.4.2.1 The DTD Specification of CDGroups

CDGroups are XML documents, hence a valid CDGroup should

- be valid according to the DTD 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 DTD in Figure 5.2 are also met.

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

`CDGroupName` The text occurring in the `CDGroupName` element corresponds to the name of the CDGroup. For the syntactical requirements, see `CDName` in Section 5.3.2.

`CDGroupURL` The text occurring in the `CDGroupURL` element identifies the location of the CDGroup file, not necessarily of the member Content Dictionaries. For the syntactical requirements, see `CDURL` in Section 5.3.2.

`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 5.3.2.

```
<!-- CDgroup.dtd -->
<!-- ****************************************** -->
<!--                                            -->
<!-- DTD for OpenMath CD group                  -->
<!-- (c) EP24969 the ESPRIT OpenMath Consortium    -->
<!-- date = 18.Feb.1999                         -->
<!-- author = s.buswell sb@stilo.demon.co.uk       -->
<!--                                            -->
<!--                                            -->
<!-- available at                               -->
<!-- http://www.nag.co.uk/~something here David~  -->
<!--                                            -->
<!-- ****************************************** -->

<!-- info on the CD group itself -->

<!ELEMENT CDGroupName      (#PCDATA) >
<!ELEMENT CDGroupVersion     (#PCDATA) >
<!ELEMENT CDGroupRevision     (#PCDATA) >
<!ELEMENT CDGroupURL          (#PCDATA) >
<!ELEMENT CDGroupDescription  (#PCDATA) >

<!-- info on the CDs in the group  -->

<!ELEMENT CDComment     (#PCDATA) >
<!ELEMENT CDGroupMember (CDComment?,CDName, CDVersion?, CDURL?) >
<!ELEMENT CDName     (#PCDATA) >
<!ELEMENT CDVersion     (#PCDATA) >
<!ELEMENT CDURL         (#PCDATA) >

<!-- structure of the group -->
<!ELEMENT CDGroup
   (CDGroupName, CDGroupVersion, CDGroupRevision?,
    CDGroupURL, CDGroupDescription,
     (CDGroupMember  | CDComment )* ) >

<!-- end of DTD for OM CDGroup -->
```

Figure 5.3: DTD Specification of CDGroups

`CDVersion` The text occurring in the `CDVersion` element identifies which version of the Content Dictionary isto 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 5.3.2.

`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 5.3.2.

`CDComment` See `CDComment` in Section 5.3.2.

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 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 *supported* 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 by the URL declared in the `CDURL` field of the Dictionary. This URL may be used to retrieve the Content Dictionary.

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* standard. The things are addressed by the *OpenMath* 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>

<CDName> meta </CDName>

<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.  It is acknowledged that this is
not the only way to do this, but it seems a natural way.

This can be viewed as updating the previous Meta-CD.

The information written here is taken from "The OpenMath Standard".
This document is a slightly stronger statement than "the following
symbols are defined as in the DTD at
http://www.nag.co.uk/projects/OpenMath/omstd/dtds/cd.dtd", since the
DTD often only says that the information inside the elements is PCDATA
without saying what this actually corresponds to. However this is the
only way this document is better than the DTD, and thus this is the
only extra information we give here.

Author: N. Howgrave-Graham
</Description>

<CDReviewDate> 1998-10-01 </CDReviewDate>
<CDStatus> experimental </CDStatus>
<CDURL> http://www.nag.co.uk/projects/OpenMath/omstd/cds/meta.ocd </CDURL>

<CDComment>
This is how one uses comments.

This whole document must be valid XML which means that we cannot have
sub-elements inside elements where we are expecting PCDATA. For this
reason it is suggested that one use the unicode characters when
clashes occur. For example &lt; and &gt; for less than and more than.
</CDComment>

<CDDefinition>
<Name> CD </Name>
<Description>
```

```
The DTD fully specifies the use of the CD tag
</Description>
</CDDefinition>

<CDDefinition>
<Name> CDDescription </Name>
<Description>
The DTD fully specifies the use of the CDDescription tag
</Description>
</CDDefinition>

<CDComment>
For those that do not have access to the DTD, the tagged entries
are the following (in no particular order):

&lt;CD&gt;
&lt;CDName&gt; &lt;/CDName&gt;
&lt;Description&gt; &lt;/Description&gt;
&lt;CDReviewDate&gt; &lt;/CDReviewDate&gt;
&lt;CDStatus&gt; &lt;/CDStatus&gt;
&lt;CDURL&gt;? &lt;/CDURL&gt;
&lt;CDUses&gt;? &lt;CDUses&gt;
&lt;CDDefinition&gt;*
&lt;Name&gt; &lt;/Name&gt;
&lt;Description&gt; &lt;/Description&gt;
&lt;Signature&gt;? &lt;/Signature&gt;
&lt;Example&gt;* &lt;/Example&gt;
&lt;FMP&gt;* &lt;/FMP&gt;
&lt;CMP&gt;* &lt;/CMP&gt;
&lt;Presentation&gt;? &lt;/Presentation&gt;
&lt;/CDDefinition&gt;

where an asterisk (?) denotes it can repeated 0 or 1 times, and a star
(*) denotes 0 or more times.
</CDComment>

<CDDefinition>
<Name> CDName </Name>
<Description>
A tag which contains PCDATA corresponding to the name of the CD.
</Description>
</CDDefinition>

<CDDefinition>
```

```
<Name> CDURL </Name>
<Description>
An optional tag which contains PCDATA corresponding to the URL where
the CD is stored.
</Description>
</CDDefinition>

<CDDefinition>
<Name> Example </Name>
<Description>
A tag which contains PCDATA to give an example of the
enclosing symbol definition.
</Description>
</CDDefinition>

<CDDefinition>
<Name> CDReviewDate </Name>
<Description>
A tag which contains PCDATA to give an expiry date of
the CD.  It should be in the form of ISO-8601, i.e. YYYY-MM-DD
</Description>
</CDDefinition>

<CDDefinition>
<Name> CDStatus </Name>
<Description>
A tag which contains PCDATA to give information on the
status of the CD. This can be either official (approved by the
OpenMath steering committee), 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> CDUses </Name>
<Description>
A tag which contains zero or more CDNames which correspond
to the CD’s that this CD depends on. This makes an inheritance
structure for CD’s. If the CD is dependent on any other CD’s they must
be present here.
</Description>
</CDDefinition>
```

```
<CDDefinition>
<Name> CDTypeUses </Name>
<Description>
A tag which contains zero or more CDNames which correspond
to the CD's that hold type information that this CD depends on. This
makes an inheritance type structure for CD's. If the CD types are
dependent on any other CD's they must be present here. For application
that do not respect types, this symbol can be ignored.
</Description>
</CDDefinition>

<CDDefinition>
<Name> Description </Name>
<Description>
A tag which contains PCDATA 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>
A tag which contains PCDATA corresponding to the name of
the symbol being defined.
</Description>
</CDDefinition>

<CDDefinition>
<Name> Signature </Name>
<Description>
An optional tag which contains PCDATA corresponding to
the type of the symbol being defined. This type information will be
an OpenMath type object as defined in chapter 5 of "First Draft of the
OpenMath Standard".
</Description>
</CDDefinition>

<CDDefinition>
<Name> Presentation </Name>
<Description>
An optional tag (which may be repeated many times) which contains
PCDATA corresponding to a way of presenting the symbol being defined.
</Description>
```

```
</CDDefinition>

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

<CDDefinition>
<Name> FMP </Name>
<Description>
An optional tag (which may be repeated many times) which contains
PCDATA corresponding to a property of the symbol being
defined. This property must be a valid OpenMath object.
</Description>
</CDDefinition>

</CD>
```

A.2 The arith1 Content Dictionary File

```
<CD>
<CDName> arith1 </CDName>
<CDURL> http://www.openmath.org/cd/arith1.ocd </CDURL>
<CDReviewDate> 2003-04-01 </CDReviewDate>
<CDStatus> official </CDStatus>
<CDDate> 2001-03-12 </CDDate>
<CDVersion> 2 </CDVersion>
<CDRevision> 0 </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>0 such that c/a is an Integer and c/b is an
Integer and lcm(a,b) > 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 > gcd(a,b).

Note that this implies that gcd(a,b) > 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>
  </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) >= 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"/>
            <OMV name="y"/>
        </OMA>
        <OMA>
          <OMS cd="arith1" name="abs"/>
            <OMV name="x"/>
            <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 are
dependant of orientation, but in MathML</CDComment>
<CDGroupMember>
<CDName>linalg1</CDName>
<CDURL>http://www.openmath.org/cd/linalg1.ocd</CDURL></CDGroupMember>
<CDComment>Linear Algebra - vector &amp; matrix constructors, those symbols which are
pendant of orientation, and in MathML</CDComment>
<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>
<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>
```

Bibliography

[1] John A. Abbott, André van Leeuwen and A. 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] Stephen Buswell, Stan Devitt, Angel Diaz, Nico Poppelier, Bruce Smith, Neil Soiffer, Robert Sutor and Stephen Watt Mathematical Markup Language (MathML) 1.0 Specification

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

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

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

[7] J. Davenport A Small OpenMath Type System

[8] IEEE Standard for binary Floating-Point Arithmetic

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

[10] OpenMath Consortium OpenMath Version 1 - Draft

[11] Nico Poppelier, Robert Miner, Patrick Ion, David Carlisle, Ron Ausbrooks, Stephen Buswell, Stéphane Dalmas, Stan Devitt, Angel Diaz, Roger Hunter, Bruce Smith, Neil Soiffer, Robert Sutor and Stephen Watt Mathematical Markup Language (MathML) 2.0 Specification

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

[13] Unicode Consortium The Unicode Standard: Version 2.0

[14] W3C Namespaces in XML

[15] W3C Extensible Markup Language XML 1.0

[16] W3C Extensible Stylesheet Language (XSL) Specification

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