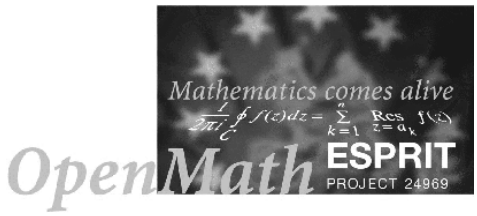
Task: **1.3**
Version: **0.1**
Date: **June 1999**



The *OpenMath* Standard

The *OpenMath* Esprit Consortium

Editors

O. Caprotti and A. M. Cohen

D1.3.3a (DRAFT)

Abstract

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

Contents

List of Figures

Chapter 1

OpenMath History

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 [5] 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 form a global *OpenMath* community which is coordinated by a Steering Committee and operates 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 [3]). By the end of 1996 a simplified specification had been agreed on and some prototype implementations have come about [11].

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 at their most recent workshop in Tallahassee (November 1998) endorsed this document as the current *OpenMath* standard.

1.2 Conclusion

OpenMath has grown from a common language for communication of mathematical objects to standard for communication between a far greater variety of software tools than computer algebra systems. The fact that the new standard enables the transport of typed lambda calculus expressions, sees to it that any formal mathematical statement, as well as its formal proof, can be communicated between automated proof checkers and other software. Thus, the possibility arises for a new Bourbakian universe whose data carrier is *OpenMath*.

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 [8]. 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 Content Dictionaries

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

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

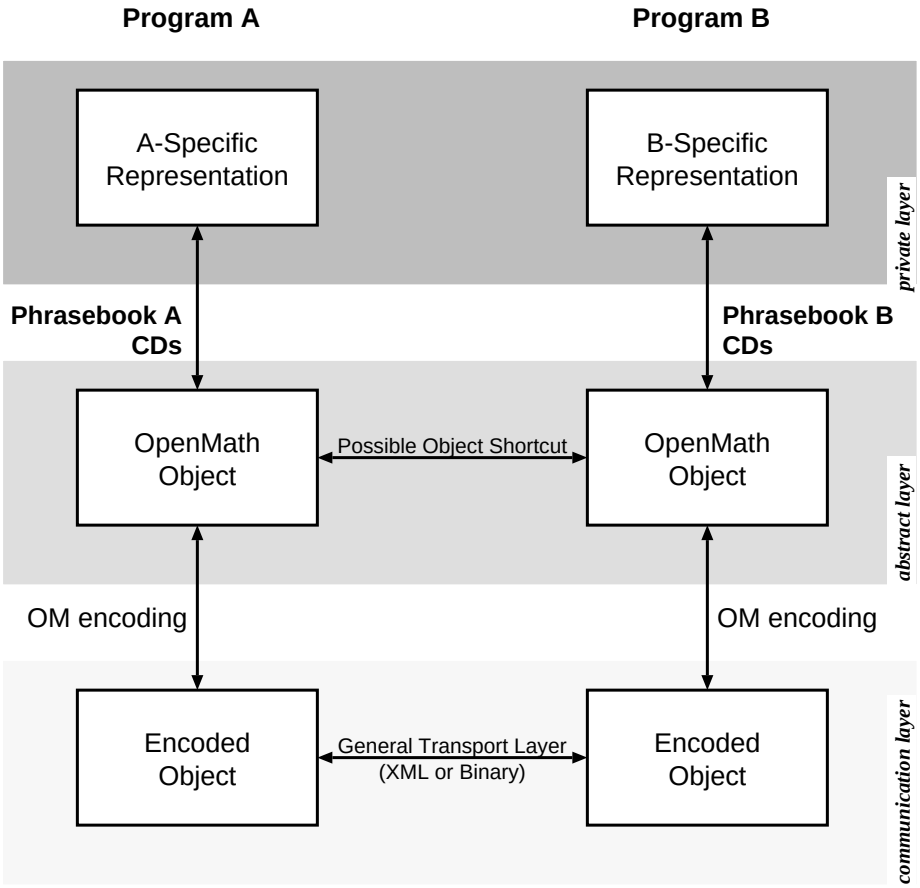


Figure 2.1: The *OpenMath* Architecture

2.3 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 carry any behavioural directive, they merely represent mathematical entities. This gives the freedom to a Phrasebook, while interpreting an *OpenMath* object, e.g., of renaming bound variables since this does not change the mathematical entity conveyed by the *OpenMath* object.

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* has no intention of being a query or programming language but a language to represent mathematical entities. 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.

2.4 *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.

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 3.1) and next by means of an abstract grammar description (Section 3.2).

3.1 Informal 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:

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

Symbols are uniquely defined by a Content Dictionary in which they occur and by a name. Each symbol has no more than one definition in a Content Dictionary. More Content Dictionaries may define differently a symbol with the same name (e.g., the symbol `plus` can be defined as associative operator in one Content Dictionary and as an associative-commutative operator in another Content Dictionary). The name of a symbol can only contain alphanumeric characters. More precisely, a symbol name verifies 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. Rendering of the symbol is explained in the presentation field of the symbol's definition in the Content Dictionary.

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). Variable names are restricted to use a subset of the printable ASCII characters. Some *OpenMath* renderers may interpret these names as UTF-7 [12] encoded unicode characters for the purpose of displaying the variable. Formally the names must match the regular expression:

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

For example the variable named `+A7E-` *might* be displayed by an *OpenMath* renderer as α as `+A7E-` is the UTF7 encoding for unicode position hex 3B1.

Floating-point numbers are computer objects rather than bigfloats. They are double precision floating-point numbers following the IEEE 754-1985 standard [2].

Character strings are sequences of characters. These characters come from the Unicode standard [9].

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.

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 will be called "arguments". An *OpenMath* application object can be used to convey the mathematical notion of application of a function to a set of arguments like in $\sin(x)$. More generally, an *OpenMath* application object can be used to convey a mathematical object built from other objects such as a polynomial constructed from a set of monomials.

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`.

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.

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. We can think of the following kinds of error:

- *mathematical errors* directly related to the mathematical meaning of some symbols (as specified in the relevant content dictionary), e.g., as `division by zero`
- *construction errors* occurring when some structural constraints on a particular object have been violated, e.g., `malformed object`, `expecting a numeric argument`, `CD name not found` ...
- *operational errors* reporting a known limitation in a program, e.g., `operation not implemented or partially implemented`
- *implementation errors* reporting an unexpected problem in a program, e.g., `assertion failed`
- *external (asynchronous) errors* completely dynamic and out of the control of the programmer, e.g., `damaged encoding`, `not enough memory`, `time limit reached`, `remote machine down`, ...

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 head label, as defined in the Content Dictionaries, should interpret the object as functional application, as constructor, or as variable binding accordingly. Thus, for example, for some application, the *OpenMath* object corresponding to $2+5$ may be seen as a command to write 7.

3.2 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.

Definition 1 (Basic *OpenMath* objects)

Integers, symbols (defined in Content Dictionaries), variables, floating point numbers, character strings, and bytearrays are basic OpenMath objects.

Definition 2 (*OpenMath* objects)

OpenMath objects are built recursively as follows.

(i) Basic OpenMath objects are OpenMath objects.

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

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

is an OpenMath application object.

(iii) If B and C are OpenMath objects, and v_1, ..., v_n $(n \geq 0)$ are OpenMath variables, then

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

is an OpenMath binding object.

(iv) If $S_1, \ldots, S_n$ are OpenMath symbols, and A, A_1, ..., A_n, $(n > 0)$ are OpenMath objects, then

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

is an OpenMath attribution object.

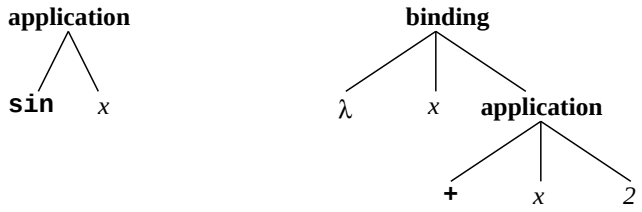


Figure 3.1: The *OpenMath* application and binding objects for $\sin(x)$ and $\lambda x.x + 2$ in tree-like notation.

(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.

A few comments are in order.

Symbols An *OpenMath* symbol always carries the name of the Content Dictionary that defines it. In the definition above 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 6.

Application *OpenMath* recognizes functional application and constructors. For instance, suppose that the *OpenMath* symbol `sin` is defined in a Content Dictionary for trigonometry, then **application**(`sin`, x) is the abstract *OpenMath* object corresponding to $\sin(x)$. Constructors build inhabitants of some symbolic type, like for instance the type of rational numbers or the type of polynomials. The rational number, usually denoted as $1/2$, is represented by the *OpenMath* application object **application**(`Rational`, 1, 2). The symbol `Rational` must be defined, by a Content Dictionary, as a constructor symbol for the rational numbers.

Binding Variable binding is expressed using *OpenMath* binding objects. The head symbol denotes the binder object, for instance `lambda`, `exists`, or `SetOf(integer)`. A sequence of *variables* and the object in which they are bound follows the head of the binding object. Thus the mathematical function $\lambda x.x + 2$ is represented by the *OpenMath* object

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

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

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

In view of this, repeated occurrences of the same variable in a binding are disambiguated consequently. For instance, the object:

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

is semantically equivalent to:

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

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

Phrasebooks are allowed use α conversion in order to avoid clashes of variable names. Suppose an object Ω contains an occurrence of the object $\mathbf{binding}(B, v, C)$. This object $\mathbf{binding}(B, v, C)$ can be replaced in Ω by $\mathbf{binding}(B, z, C')$ where z is a variable not occurring free in C and C' is obtained from C by replacing each 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.

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

Attribution Composition of attributions, as in

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

is semantically equivalent to a single attribution, that is

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

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.)

The syntactic class of an attributed variable is variable, and that of all other attributed objects is object. In particular, an attributed symbol is an object.

Objects can be decorated in a multitude of ways. In Deliverable [6], typing of *OpenMath* objects is expressed by using an attribution. $\mathbf{attribution}(A, \texttt{type}\ t)$ is a way to store that object A has type t. Note that both A and t are *OpenMath* objects.

Error Error objects might consist only of a symbol as in the object: $\mathbf{error}(S)$. Such errors are typically unrelated to any *OpenMath* object and occur for instance during communication.

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.

Chapter 4

OpenMath Encodings

In this chapter, two encodings are defined that map *OpenMath* objects to 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 [1] (also known as ASCII characters) and is an XML application. Although the XML markup of the encoding uses only ASCII characters, OpenMath strings may uses arbitrary Unicode/ISO 10646:1988 characters [9] 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.

This encoding is rather simple and straightforward (except, maybe, for character strings). The main problem is to cope with the lexical and syntactic constraints of XML [10]. The latter implies escaping some characters.

4.1.1 A Grammar for the XML Encoding

The syntax of the XML encoding is more restrictive than the XML syntax. Literal values for attributes must always be enclosed between quotation marks ("). Apostrophe characters (')

are not allowed as enclosure characters, that is `<OMS cd = "transc" name ="sin"/>` but not `<OMS cd = 'transc' name ='sin'/>`. The character set is restricted to ASCII. This encoding is case sensitive for element and attribute names: neither `<omv name="x"/>` nor `<OMV NAME="x"/>` could be recognized. These restrictions are given to ease the writing of *OpenMath* applications. As this encoding will be produced mainly by programs and not humans, these restrictions have no practical drawbacks.

The notation used in this section and in Figure 4.1 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.

S	⟶	(space|tab|cr|nl)+
integer	⟶	`-?[0-9]+ | -?x[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]+`
utf7	⟶	`([A-Za-z0-9+='(),-./:?!"#$%*;@[]^_‘{|}] | S)+`
varname	⟶	`([A-Za-z0-9+='(),-./:?!#$%*;@[]^_‘{|}])+`
base64	⟶	`([A-Za-z0-9+/=] | S)+`
start	⟶	`<OMOBJ>` S? object S? `</OMOBJ>`
symbol	⟶	`<OMS` S `name` S? `=` S "symbname" S? `cd` S? `=` S? "cdname" S? `/>`
	|	`<OMS` S `cd` S? `=` S? "cdname" S? `name` S? `=` S? "symbname" S? `/>`
variable	⟶	`<OMV` S `name` S? `=` S? "varname" S? `/>`
	|	`<OMATTR>` S? `<OMATP>` attrs S? `</OMATP>` S? variable S? `</OMATTR>`
object	⟶	symbol
	|	variable
	|	`<OMI>` S integer S `</OMI>`
	|	`<OMF` S `dec` S? `=` S? "fpdec" S? `/>`
	|	`<OMF` S `hex` S? `=` S? "fphex" S? `/>`
	|	`<OMSTR>` S utf7 S `</OMSTR>`
	|	`<OMB>` S base64 S `</OMB>`
	|	`<OMA>` S? object S? objects S? `</OMA>`
	|	`<OMBIND>` S? object S? `<OMBVAR>` S? variables S? `</OMBVAR>` S? object S? `</OMBIND>`
	|	`<OME>` S? symbol S? objects S? `</OME>`
	|	`<OMATTR>` S? `<OMATP>` S? attrs S? `</OMATP>` S? object S? `</OMATTR>`
attrs	⟶	symbol S? object
	|	symbol S? object S? attrs
objects	⟶	S?
	|	object S? objects
variables	⟶	S?
	|	variable S? variables

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

The document type definition corresponding to this grammar is given in Figure 4.2.

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). Integers written in base 10 satisfy the regular expression `-?[0-9]+`. Integers written in base 16 satisfy `-?x[0-9A-F]+`.

The integer 10 can be thus encoded as:

```
<OMOBJ>
   <OMI> 10 </OMI>
</OMOBJ>
```

or as:

```
<OMOBJ>
   <OMI> xA </OMI>
</OMOBJ>
```

but neither `<OMI> +10 </OMI>` nor `<OMI> +xA </OMI>` can be used.

The negative integer -120 can be encoded as:

```
<OMOBJ>
     <OMI> -120 </OMI>
</OMOBJ>
```

or as:

```
<OMOBJ>
     <OMI> -x78 </OMI>
</OMOBJ>
```

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 UTF-7 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:

```
<!-- 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 4.2: DTD for the *OpenMath* XML encoding of objects.

```
<OMOBJ>
     <OMV  name="x"/>
</OMOBJ>
```

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 utf8[**?**], although XML encoded OpenMath may be embedded in a containg 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 `&`.

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 [4]. 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 [4] for more detailed information.

The XML encoding of compound *OpenMath* objects is described here 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, ..., 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:

```
<OMOBJ><OMA>
       <OMS cd="transc" name="sin"/>
       <OMV name="x"/>
       </OMA>
</OMOBJ>
```

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

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**(`lambda`, x, **application**(`sin`, x)) is:

```
<OMOBJ><OMBIND>
         <OMS cd="fns" name="lambda"/>
         <OMBVAR><OMV name="x"/></OMBVAR>
         <OMA><OMS cd="transc" name="sin"/>
              <OMV name="x"/>
         </OMA>
         </OMBIND>
</OMOBJ>
```

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

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

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

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

or to express the type of a variable:

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

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

For instance, the encoding of the object **error**(`DivisionByZero`, **application**(`divide`, x, 0)) is:

```
<OMOBJ><OME>
       <OMS cd="arith" name="DivisionByZero"/>
       <OMA>
            <OMS cd="arith" name="divide" />
            <OMV name="x"/>
```

```
            <OMI> 0 </OMI>
        </OMA>
        </OME>
</OMOBJ>
```

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 n 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.

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

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	⟶	variable
	|	variable vars
objects	⟶	
	|	object objects

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

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 [2], 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 16 bits (Unicode) characters or not. If the string contains only 8 bits 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 bytes characters, it is encoded as the two bytes 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 (16 bits 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).

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="arith"/>
    <OMA>
      <OMS name="plus" cd="arith"/>
      <OMV name="x"/>
      <OMV name="y"/>
    </OMA>
    <OMA>
```

```
      <OMS name="plus" cd="arith"/>
      <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	70	p (symbol name begin
08	symbol tag	6c	l .
05	cd length	75	u .
05	name length	73	s .)
61	a (cd name begin	05	variable tag
72	r .	01	name length
69	i .	78	x (name)
74	t .	05	variable tag
68	h .)	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 (arith:plus)
10	begin application tag	45	variable tag (with share bit on)
08	symbol tag	00	reference to first variable seen (x)
05	cd lenght	05	variable tag
04	name length	01	name length
61	a (cd name begin	7a	z (variable name)
72	r .	11	end application tag
69	i .	11	end application tag
74	t .	19	end object tag

4.2.5 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 (CD's) 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.

Key notion: A *Content Dictionary* holds the meanings of (various) mathematical "words". These words are referred to as *symbols*.

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).

For the vast majority of cases it is thought that the Content Dictionaries that an application understands will be constant, and be intrinsic to the applications' mathematical use. However the above approach can also be used for applications which supposedly understand every Content Dictionary, or alternatively for applications which understand a changeable number of Content Dictionaries (perhaps after being sent Content Dictionaries in some way).

The last paragraph brings up interesting issues regarding the use of Content Dictionaries. The primary use is thought to be for designers of Phrasebooks. (Phrasebooks are 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, to enable the Phrasebook designer to effectively state which objects translate to which.

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.

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 Dictionary functionality

In this section we define the functionality of the 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.

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 version number of the current version of the Content Dictionary.
- The status of the Content Dictionary, which is one of the following
  - official (approved by the *OpenMath* society)
  - experimental (currently being tested)
  - private (used by a private group of *OpenMath* users)
  - obsolete (an obsolete Content Dictionary kept only for archival purposes)
- An optional URL for this Content Dictionary.
- An optional list of Content Dictionaries which this Content Dictionary depends on (producing a Content Dictionary hierarchy).

The optional arguments above are only optional in the sense that their information may not exist; if it does however, it must be present.

Information that is restricted to a particular symbol includes:

- The name of the symbol.
- A description of this symbol.
- Optional examples of the use of this symbol. The information present here can be any valid XML .
- Optional properties that this symbol should obey. Properties can one of two cases:
  - formal (i.e. completely in terms of other *OpenMath* objects), or
  - commented (i.e. just valid XML text).

  An application which says it understands a Content Dictionary symbol need not understand a property of the symbol, but if it does, then the application must adhere to this property.
- Default presentation information. This is included since a large proportion of mathematics will require rendering, and it is thought that *OpenMath* should not *completely* distance itself from this.

Since the previous description of Content Dictionaries, `FunctorClass` and `PartOf` have been deleted; such information should be conveyed elsewhere, e.g., in the descriptions. `CDExpire` has been given the more meaningful name `CDReviewDate` and the date format has been made ISO compliant.

Also Signature information that was previously was provided in the Content Dictionaries is now described in *Signature Files* described in section 5.4 below.

Content Dictionaries may be grouped into *CD Groups*. These groups allow applications to easily refer to collections of Content Dictionaries. One particular CD Group of interest is the "MathML CD Group". 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. *OpenMath* objects built from symbols that come from Content Dictionaries in this CD Group may be expected to be eaily transformed between *OpenMath* and MathML encodings. The detailed structure of a CD Group is described in section 5.5 below.

5.3 An encoding for Content Dictionaries

The encoding for Content Dictionaries has been designed to be totally valid XML, at the same time as being relatively easy to read and write. A valid Content Dictionary document should be

- parsable by the DTD given in Figure 5.1,
- adhere to the stricter definitions of the parsed character data, PCDATA for short, given in section 5.3.2.

We now explain exactly what these conditions mean for the encoding of Content Dictionaries, and give examples of sub-encodings. An example of a complete Content Dictionary is given in appendix A, which is a Content Dictionary for describing Content Dictionaries themselves.

5.3.1 The DTD specification of a Content Dictionary

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 PCDATA referred to in Figure 5.1 must actually make sense to, say, a Phrasebook designer. In this section we define exactly the format of the PCDATA 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.

`CDURL` The text occurring in the `CDURL` element should be a valid URL where the source for the Content Dictionary encoding can be found (if it exists). The filename should conform to ISO 9660 [7].

`Example` The text occurring in the `Example` element is used to give examples of the enclosing symbol, and can be any `utf7` text. Note that `Examples` must be with respect to some symbol and cannot be "loose" in the Content Dictionary. With the use of inheritance to reduce verbosity it is suggested that wherever possible, all the examples concerning a symbol be in with the symbol definition.

`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` elemet 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 the Content Dictionary.[1]

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

`CDUses` It mentions the names of the Content Dictionaries that are used in the definitions of the symbols in the current Content Dictionary.

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

`Name` The text occurring in the `Name` element corresponds to the name of the symbol, and is specified as in Chapter4.

`CMP` The text occurring in the `CMP` element corresponds to a property of the symbol.

`FMP` The content of the `FMP` element also corresponds to a property of the symbol, however the content of this element must be a valid *OpenMath* object in the XML encoding.

[1] Add discussion about minor and major changes between versions. Minor changes do not invalidate objects built with previous versions, viceversa for major changes.

```
<!-- 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 -->
<!--                                              -->
<!--                                              -->
<!-- ******************************************** -->
<!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 CDStatus       %inhel; >
<!ELEMENT Description    %inhel; >
<!ELEMENT Name           %inhel; >
<!ELEMENT Signature      %inhel; >
<!ELEMENT Presentation   %inhel; >
<!ELEMENT CMP            %inhel; >
<!-- include dtd for OM objects -->
<!ENTITY  % omobjectdtd SYSTEM "omobj.dtd" >
%omobjectdtd;
<!ELEMENT FMP            %inhel2; >             <!-- allow embedded OM -->
<!ELEMENT Example        %inhel2; >
<!ELEMENT CDDefinition    (CDComment | Name | Description  |
                           Signature | Example | FMP |
                          CMP | Presentation)* >
<!ELEMENT CD             (CDComment | CDName | Description |
                         CDReviewDate | CDDate |CDVersion |
                         CDStatus | CDURL | CDUses | CDDefinition |
Example)* >
<!-- end of DTD for OM CD -->
```

Figure 5.1: DTD of *OpenMath* Content Dictionaries

5.4 Content Dictionary Signature Files

Early drafts of the *OpenMath* standard specified that Content Dictionaries had a Signature element in which the *signature* of the symbol was defined. The disadvantage of this approach is that the signature would need to be reference to a specific type system. However *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.

Signature files have a header which specifies the Content Dictionary which 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.

The exact syntax for the signature file is specified in the following XML DTD.

5.4.1 XML DTD for Signature Files

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

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

<!ELEMENT CDComment        (#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 -->
```

5.5 Content Dictionary Groups

A CD Group file is an XML document that essentially has header information such as CDGroup-Name and CDGroup version, analogous to the elements in CD files described above. Then follows a list of Content Dictionaries names, and optionally URLs to canonical copies of the Content Dictionaries, and comments elements.

The exact format of the CD Group file is specified by the XML DTD given in the next section.

5.5.1 XML DTD for CDGroup Files

```
<!-- 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 CDGroupURL          (#PCDATA) >
<!ELEMENT CDGroupDescription  (#PCDATA) >

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

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

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

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

Chapter 6

OpenMath Compliance

EDITORIAL NOTE: This chapter is cut&pasted from the OM version 1 draft of the standard available at the INRIA OpenMath site. It is here just as food for thinking.

An OpenMath application can be defined as any program that is capable of accepting and/or generating any of the encodings for OpenMath objects described in this document.

This requirement is enough to enable two OpenMath applications to exchange mathematical objects. It is enough to write a function in a computer algebra system that would read an OpenMath object from a file and translate it to its own representation and conversely, to write an object of the system to a file so that it can be read back by another OpenMath application. Nevertheless, it does not specify how an application is supposed to treat the OpenMath object, how it is supposed to react to various error conditions and how an "on-line" communication can be established between two (or more) applications. This section is dedicated to answer some of these questions. More precisely, we are going to describe additional requirements at several levels, that can be viewed as representing increasing levels of cooperation between applications:

1. Additional rules an OpenMath application must follow regarding the OpenMath objects it can read and write. This is independent of any other application or form of communication. It is just some very basic requirements on how an application should treat symbols and content dictionaries.
2. In a context where an OpenMath application A can send an OpenMath object to another application B and B can send another object to A, constraints on what B can send in response to what A send. At this level, we are mostly concerned about how common errors must be reported.
3. In a context where two OpenMath applications A and B have established a communication channel enabling them to exchange OpenMath objects, constraints on the sequence of OpenMath objects exchanged between them.
4. In the context of two applications A and B that want to communicate, methods to establish the communication.

All these requirements are layered. An application following the fourth set must follow the first, second and third sets of rules.

The first set of rules must be followed by any OpenMath application. In this sense this is the base level for OpenMath compliance. For example, a computer algebra system that can only read an OpenMath object from a file or export an OpenMath object to a file must follow them. In fact, level 1 can be applied even to applications that do not use the encodings defined in this document, for example applications using CORBA or OLE to transport OpenMath objects in their own formats.

The second set must be followed in a simple, single request/response scenario. For example, it applies to the same computer algebra system that would have a "batch mode" where it reads an OpenMath object and output some result as another OpenMath object, or for a Remote Procedure Call OpenMath server (independently of the technology used for the remote procedure call itself).

The third set is to be used in situations where a real communication is established, with a sequence of request/response. It does not say anything about the way the "physical" communication has been established.

The fourth set is describing some standard methods to establish the communication. These methods are largely operating-system dependent. They enable an OpenMath application to call (or to find) another OpenMath application.

The goal of these rules is to simplify the building of OpenMath applications and their use while being adapted to the largest number of communication methods and technologies.

6.1 Level 1: How an Application Must Behave with Respect to Content Dictionaries

In this section, we describe how an OpenMath application must behave with respect to some content dictionaries. An OpenMath application must always come with:

- A list of the content dictionaries it can deal with as input (the names of the CD or the CD themselves if they are not public, official CD)
- A list of the symbols from these content dictionaries that it will not handle (see 7.2.1),
- A list for the content dictionaries it uses in its outputs

In addition, an application should mention whether or not it is able to handle objects containing arbitrary symbols.

The content dictionaries basically define what parts of mathematics the application understands and produces. An application must follow all the requirements and constraints given in all the dictionaries it explicitly mentions when it uses (reads, manipulate and outputs) OpenMath objects containing symbols defined in these dictionaries.

There are useful OpenMath applications that can accept symbols from any dictionary. One such (trivial) example is an application converting an OpenMath object stored in a file in the binary encoding to the text encoding. We can also envisage applications that do some useful things with some CD but can potentially accept objects containing symbols from any CD. A good example is an application that typesets OpenMath objects. This application will certainly treat some symbols in some ad-hoc ways and revert to some default printing for the symbols it does not know about. For this reason, we cannot force an OpenMath application to always detect all

violations of semantic constraints (given in CD) in all the OpenMath objects it reads or writes (but it must do so for the symbols in the CD that it explicitly mentions).

As OpenMath is all about communicating mathematical objects and not requesting computations, it is usually not possible to impose a particular computational behavior to an OpenMath application receiving a particular OpenMath object. For example, the same OpenMath object can be given to both an equation editor and a symbolic computation system. The specification of how an OpenMath application should behave is thus largely outside of the scope of the information attached in an OpenMath CD, although the CD can mention some mandatory behaviours for certain kind of applications. For example, in the case of a symbol representing factorization of integers, it is very useful to specify what kind of OpenMath object an OpenMath application that can compute this factorization is supposed to return.

6.2 Level 2: Simple Communication

Here, we attempt to define how an OpenMath application should behave in the context of a very crude level of communication: in the context of a single sequence of request/response.

In this context, the application that receives the request (an OpenMath object) is required to send back an OpenMath object that represents the response (provided the application is still functional...). The request and the response must use the same encoding. If an error occurs, the response should be an OpenMath error. For certain common error conditions, some predefined symbols in the Control content dictionary should be used, as specified in the sequel.

Partially Implemented Content Dictionaries

An application that claims to understand a content dictionary is supposed to understand the objects build with the symbols occurring in the CD, following the specification given in the CD. But this understanding depends on the intended behavior of the application. The application is not required to treat all the symbols the same way. It is possible that some symbols may be "unimplemented" by the application. This is in fact necessary to insure that we can write complete CD that can be used without putting too much constraint on an application.

For example, we can envision an application that computes zeros of functions represented as algebraic expressions containing special functions. This application can claim to understand the Basic CD and the SpecialFunction CD. If the SpecialFunction CD is a (sufficiently) exhaustive CD, it may be the case that the application don't know how to compute the zeroes of some functions in this CD. If we insist that the application is denied to claim the understanding of the SpecialFunction CD in this case, we would end up with a lot of applications defining their own CD, that would be minor variations of some other or a lot of "non compliant" but very useful applications.

When an OpenMath application receives an object containing a symbol that is unimplemented, it should return an error, whose symbol is unhandledSymbol in the Control CD and whose argument is the unhandled symbol that triggered the error in the last OpenMath object received (in case there are several such symbol, only one is returned).

Unknown Symbols and Unknown Content Dictionaries

It is perfectly possible that an application reads an object containing a symbol that does not exist in the mentioned content dictionary, for example when reading an object coming from a file written by a human that made a typing error. In this case, the application should return an error, whose symbol is unknownSymbol in the Control CD and whose argument is the unknown symbol that triggered the error.

In the case where an unknown content dictionary is present, the application should return an error, whose symbol is unknownCD in the Control CD and whose argument is a string that is the name of the unknown dictionary that triggered the error.

In the case where there are several such errors in an OpenMath object, there is no requirement on which particular unknown or unhandled symbol or content dictionary should be returned with the error.

6.2.1 Encoding Errors

When an application detects a lexical or syntactic error in a piece of OpenMath encoding that it has read, it must generate an error, whose symbol is encodingError in the Control CD and whose argument is a string that should explain the error.

6.2.2 Operational and Implementation Errors

We distinguish the following kinds of common error conditions that may be generated by an OpenMath application:

1. Errors due to algorithmic restrictions of the implementation. This includes operations not implemented or partially implemented, division by zero and other domain errors.
2. Errors caused by the limitations of an implementation when dealing with OpenMath objects, such as limits on the size of objects or on the kind of objects manipulated. This can include limits on the size of a bytearray or integer, a limit on the number of arguments of an application or the inability to deal with Unicode characters outside ISO latin 1.
3. Errors caused by unexpected problems (that should never happen).

Three symbols in the Control CD are defined to report these three kinds of errors. They are

1. algorithm
2. limitation
3. unexpected

All these errors have one required argument and one optional argument. The first (mandatory) one is the symbol that is the a string describing the problem. The second (optional) one is a symbol that is relevant to the error. It is expected, for example, to be the symbol that triggered the computation that lead to an algorithm error.

This is just a predefined way to report these kinds of errors for applications that do not want to use their own conventions. An application is free to use its own way, provided it uses the "error"

construct to report errors and that the symbols used in the error object has a proper definition in its content dictionary.

6.2.3 Asynchronous Errors

A conformant OpenMath application should try to behave in certain ways in response to some asynchronous errors. What we call asynchronous errors here are errors that are directly connected with some external conditions, usually limits (not enough memory) or errors in some system calls (I/O errors,disk full, machine down).

In this case, the application should try to send an error, with the system symbol in the Control CD with an argument that is a string describing the problem.

6.3 Level 3

The communication has been physically established between two OpenMath applications when both applications have some means to send encoded OpenMath objects to the other and to receive an encoded OpenMath object sent by the other.

At this level, the sequence of OpenMath objects exchanged should be strictly request/response i.e. an application should never attempt to read more than one object without sending an object back.

An application wishing to terminate the communication should first send the OpenMath symbol terminate in the Control CD.

If an application detects an error as described at level 2 that it cannot recover from, it is required to send an error, whose symbol is terminate in the Control CD and whose argument is the error as specified at level 2. If an application detects an error for which it has defined its own symbol, it should also encapsulate it in a terminate error as explained above.

6.4 Level 4

OpenMath Clients and Servers in a UNIX Environment

Here, we describe a method that enables an OpenMath application to launch another OpenMath application and establishing a proper communication channel.

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

The Meta Content Dictionary

```
<CD>

<CDName> meta </CDName>
<CDReviewDate> 1999-09-01 </CDReviewDate>
<CDDate> 1999-05-11 </CDDate>
<CDVersion> 1.1a </CDVersion>
<CDStatus> experimental </CDStatus>
<CDURL> http://openmath.nag.co.uk/Projects/openmath/corecd/cd/meta.ocd </CDURL>

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

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

<CDComment>
First Draft 1998  N. Howgrave-Graham.
Modified 1999-02-13 R Timoney to fix errors and omissions.
Modified 1999-03-28 D Carlisle to change description of Signature.
Rewritten 1999-05-07 D Carlisle.
Modified 1999-05-11 D Carlisle. Added CDDate and CDVersion.
</CDComment>




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

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

<![CDATA[
<CD>
<CDName> </CDName>
<Description> </Description>
<CDReviewDate> </CDReviewDate>
<CDDate> </CDDate>
<CDVersion> </CDVersion>
<CDStatus> </CDStatus>
<CDURL>? </CDURL>
<CDUses>? <CDUses>
<CDDefinition>*
<Name> </Name>
<Description>* </Description>
<Signature>? </Signature>
<Example>* </Example>
<FMP>* </FMP>
<CMP>* </CMP>
<Presentation>? </Presentation>
</CDDefinition>
]]>

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

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

<CDDefinition>
<Name> Example </Name>
<Description>
```

```
An element which contains an arbitrary number of children,
each of which is either a string or an XML encoding of an OpenMath Object.

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

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

<CDDefinition>
<Name> CDVersion </Name>
<Description>
An element which contains a version string for the CD.
This should be of the form 1.2a with the letter just being changed
for "cosmetic" edits to the file, and the major or minor version numbers
being changed for structural changes that affect the OpenMath Objects
that may use this CD.
</Description>
</CDDefinition>


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

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

official (approved by the OpenMath Society),

experimental (currently being tested),

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

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

```
<CDDefinition>
<Name> CDUses </Name>
<Description>
An element which contains zero or more CDNames which correspond
to the CDs that this CD depends on. This makes an inheritance
structure for CDs. If the CD is dependent on any other CDs they must
be present here.
</Description>
</CDDefinition>

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

<CDDefinition>
<Name> Name </Name>
<Description>
An element containing the string  corresponding to the name of
the symbol being defined. This must match the syntax for
symbol names given in the OpenMath Standard.
</Description>
</CDDefinition>

%<CDDefinition>
%<Name> Signature </Name>
%<Description>
%An optional element which contains the XML encoding
%of an OpenMath object corresponding to
%the type of the symbol being defined.

%This is not used in the current CD as the signatures are specified
%separately in signature files, to allow different type systems to
%be used.
%</Description>
%</CDDefinition>

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

<CDDefinition>
<Name> CMP </Name>
<Description>
An optional element (which may be repeated many times) which contains
```

```
a string corresponding to a property of the symbol being
defined.
</Description>
</CDDefinition>

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

Each child corresponds to to a property of the symbol being defined.
</Description>
</CDDefinition>

</CD>
```

Appendix B

The arith1 Content Dictionary

```
<CD>
<CDName> arith1 </CDName>
<CDURL> http://openmath.nag.co.uk/Projects/openmath/corecd/cd/arith1.ocd </CDURL>
<CDReviewDate> 1999-09-01 </CDReviewDate>
<CDStatus> experimental </CDStatus>
<CDDate> 1999-06-03 </CDDate>
<CDVersion> 1.0b </CDVersion>
<CDUses> </CDUses>
<Description>
This CD holds very vague notions of common arithmetic functions.
</Description>
<CDComment>
This CD is intended to be ‘compatible’ with MathML.

Written by N. Howgrave-Graham on 1998-07-05.

Modified 1999-02-13 to fix errors and omissions.
Modified 1999-03-26 to add sum and product
Minor Fixes 1999-04-13 DPC
Added unary_minus 1999-06-03 DPC
</CDComment>

<CDDefinition>
<Name> plus </Name>
<Description>
An nary commutative function plus.
</Description>
<CMP> 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 denoting  unary minus. Ie
the additive inverse.
</Description>
<CMP> 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 binary minus symbol. This is equivalent to adding the
additive inverse.
```

```
</Description>
<CMP> 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>
This is an n-ary multiplication function.
</Description>
</CDDefinition>

<CDDefinition>
<Name> divide </Name>
<Description>
This is the (binary) division function that denotes the first argument
right-divided by the second, i.e. divide(a,b)=a*inverse(b). It is the
inverse of multiplication function as commented below.
</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>
A binary powering function. The first argument is raised to the power
of the second argument. When the second argument is not an integer
care should be taken to the meaning of this function; however it is
here to represent general powering.
</Description>
</CDDefinition>

<CDDefinition>
<Name> conjugate </Name>
<Description>
A unary function to give the complex conjugate of its argument
</Description>
</CDDefinition>

<CDDefinition>
<Name> abs </Name>
<Description>
A unary function to give the absolute value of its argument. This is
used for the absolute size of complex numbers as well (commonly
referred to as mod).
</Description>
</CDDefinition>

<CDDefinition>
<Name> root </Name>
<Description>
A binary function to give roots. The first argument is "lowered" to
the root of the second argument. This can be viewed as the inverse of
powering as commented below.
```

```
Care should be taken to the meaning of this function (i.e. which root
is being taken); however it is here to represent the general notion of
taking n'th roots.
</Description>
<CMP> power(root(a,n),n) = a </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>
Form taking two arguments, first being an integer interval giving the
range of summation, second being the function to be summed. Compare
defint in calculus CD.
</Description>
<Example>
<OMOBJ>
  <OMA>
    <OMS cd="arith1" name="sum"/>
      <OMA>
        <OMS cd="interval" 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>
Form taking two arguments, first being an integer interval giving the
range of summation, second being the function to be multiped.
Compare defint in calculus CD.
</Description>
<Example>
<OMOBJ>
  <OMA>
    <OMS cd="relation1" name="eq"/>
    <OMA>
      <OMS cd="integer" name="factorial"/>
      <OMV name="n" />
    </OMA>
    <OMA>
      <OMS cd="arith1" name="product"/>
        <OMA>
          <OMS cd="interval" 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>
```

Bibliography

[1] Iso 7-bit coded character set for information interchange. ISO 646:1983, 1983.

[2] Ieee standard for binary floating-point arithmetic. ANSI/IEEE Standard 754, 1985.

[3] John A. Abbott, André van Leeuwen, and A. Strotmann. *OpenMath*: Communicating Mathematical Information between Co-operating Agents in a Knowledge Network. *Journal of Intelligent Systems*, 1998. Special Issue: "Improving the Design of Intelligent Systems: Outstanding Problems and Some Methods for their Solution.".

[4] N. Borenstein and N Freed. MIME (Multipurpose Internet Mail Extensions) Part One: Mechanism for Specifying and Describing the Format of Internet Message Bodies. RFC: 1521, September 1993. Available at `http://www.math-inf.uni-greifswald.de/ teumer/mime/1521/rfc1521ToC.html`.

[5] Stephen Buswell, Stan Devitt, Angel Diaz, Nico Poppelier, Bruce Smith, Neil Soiffer, Robert Sutor, and Stephen Watt. Mathematical Markup Language (MathML) 1.0 Specification. W3C Recommendation 19980407, April 1998. Available at `http://www.w3.org/TR/REC-MathML/`.

[6] O. Caprotti and A. M. Cohen. A Type System for OpenMath. OpenMath Deliverable 1.3.1b, September 1998. Available at `http://www.nag.co.uk/projects/OpenMath.html`.

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

[8] OpenMath Consortium. OpenMath Version 1 - Draft, June 1998. Available at `ftp://ftp-sop.inria.fr/safir/OM/v1.ps` .

[9] Unicode Consortium. *The Unicode Standard: Version 2.0.* Addison-Wesley Developers Press, 1996.

[10] World Wide Web Consortium. Extensible Markup Language (XML) 1.0. W3C Recommendation REC-xml-19980210, February 1998. Available at `http://www.w3.org/TR/1998/REC-xml-19980210`.

[11] S. Dalmas, M. Gaëtano, and S. Watt. An OpenMath 1.0 Implementation. pages 241–248. ACM Press, 1997.

[12] D. Goldsmith and M. Davis. UTF-7: A Mail Safe Transformation Format of Unicode. RFC: 2152, May 1997. Available at `http://www.cis.ohio-state.edu/htbin/rfc/rfc2152.html`.