OMDoc: An Open Markup Format for Mathematical Documents (Version 1.1)

Michael Kohlhase
Computer Science, Carnegie Mellon University
Pittsburgh, Pa 15213, USA
http://www.cs.cmu.edu/~kohlhase

October 5, 2005

Abstract

In this report we present a content markup scheme for (collections of) mathematical documents including articles, textbooks, interactive books, and courses. It can serve as the content language for agent communication of mathematical services on a mathematical software bus. We motivate and describe the OMDoc language and present an Xml document type definition for it. Furthermore, we discuss applications and tool support.

This document describes version 1.1 of the OMDoc format. This version is mainly a bug-fix release that has become necessary by the experiments of encoding legacy material and theorem prover interfaces in OMDoc. The changes are relatively minor, mostly adding optional fields. Version 1.1 of OMDoc freezes the development so that version 2.0 can be started off.

In contrast to the OMDoc format which has not changed much, this report is a total re-write, it closes many documentation gaps, clarifies various remaining issues. and adds a multitude of new examples.

Contents

Chapter 1

Introduction

It is plausible to expect that the way we do (i.e. conceive, develop, communicate about, and publish) mathematics will change considerably in the next nine[1] years. The Internet plays an ever-increasing role in our everyday life, and most of the mathematical activities will be supported by mathematical software systems (we will call them mathematical services) connected by a commonly accepted distribution architecture, which we will call the mathematical software bus. We will subsume all proposed architectures and implementations of this idea [FHJ+99, FK99, DCN+00, AZ00] by the term MathWeb. We believe that interoperability based on communication protocols will eventually make the constructions of bridges between the particular implementations simple, so that the combined systems appear to the user as one homogeneous web.

One of the tasks that have to be solved is to define an open markup language for the mathematical objects and knowledge exchanged between mathematical services. The OMDoc format presented in this report attempts to do this by providing an infrastructure for the communication and storage of mathematical knowledge.

In chapter 2 we will describe the status quo of mathematical markup schemes before OMDoc and show that these markup schemes – while giving a good basis – are not sufficient for content-based markup of mathematical knowledge. They do not provide markup for mathematical forms like definitions, theorems, and proofs that have long been considered paradigmatic of mathematical documents like textbooks and papers. They also leave implicit the large-scale structure of mathematical knowledge. In particular, it

[1] In the release document of OMDoc1.0 [Koh00c] we claimed that it would change in the next 10, and that is one year ago.

has traditionally been structured into mathematical theories that serve as a situating context for all forms of mathematical communication.

In chapter 3, we define the OMDoc markup primitives and motivate them from either particular structures in mathematical documents or from processing needs of computer-supported mathematics. As all mathematical communication is in the form of (or can be transcribed to) mathematical documents such as publications, overhead slides, letters, e-mails, in/output from mathematical software systems, OMDoc uses documents as a guiding intuition for mathematical knowledge with the goal of providing a framework, where all of these forms can be accommodated. In accordance with this motivation OMDoc provides a rich mix of elements of informal and formal mathematics. To model particular kinds of documents in OMDoc usually only a subset will be needed, e.g. informal ones for traditional mathematical textbooks, or formal ones for communication of software systems. However, availability of both kinds of markup primitives in OMDoc allow to develop novel kinds of mathematical documents, where formal and informal elements are intimately intermixed.

We will discuss current and intended applications of the OMDoc format in chapter 4 and discuss which applications will need which parts of the OMDoc format.

Finally, the appendix contains useful materials like the OMDoc document type definition, and a quick reference table.

OMDoc **Version 1.1**

This document describes version 1.1 of the OMDoc format. Version 1.0 has been released on November 1. 2001, after about 18 Months of development, to give developers a stable interface to base their systems on. It has been adopted by various projects in automated deduction, algebraic specification and computer-supported education. The experience from these projects has uncovered a multitude of small deficiencies and extension possibilities of the format, that have been discussed in the OMDoc community. Version 1.1 is an attempt to roll the uncontroversial and non-disruptive part of the extensions and corrections into a consistent language format. We have tried to keep the changes to version 1.0 conservative, adding optional attributes or child elements.

In some cases we had to introduce non-conservative changes, to repair design flaws and inconsistencies of version 1.0. One example is the `hpothesis` element that has received a required attribute `discharged-in` that is necessary for specifying the scope of local assumptions in proofs, and cannot be inferred from the context. To minimize disruption we have tried to keep changes like this one to a minimum for the elements that are in frequent use today. We are working on a new version (OMDoc2.0) that will incorporate re-organizations of central features of OMDoc like the `definition` element.

We have however re-organized some parts of the OMDoc format that are currently less used in the anticipation that this will make them more effective. Examples are the representations of complex theories (see sections 3.3.2 to 3.3.4) or the organization of non-XML data (section 3.4.2).

Finally, we have added new features that were missing from OMDoc1.0 and turned out to be important for the enterprise of representing mathematical knowledge. Examples of this are a new referencing scheme for OMDoc elements in section 3.6 and a new way of specifying presentation for OMDoc elements. In both cases, the method that was used in OMDoc1.0 for symbols is extended and generalized to arbitrary OMDoc elements. These extensions have found their way into OMDoc1.1, even though they are not totally fixed yet, since we anticipate to gain implementation experience for OMDoc2.0. They are non-disruptive, since they are strictly additional.

An element-by-element account of the changes is tabulated in appendix B.

Acknowledgments

Of course the OMDoc format has not been developed by one person alone, the original proposal was taken up by several research groups, most notably the ΩMEGA group at Saarland University, the InKa and ActiveMath projects at the German Research Center of Artificial Intelligence (DFKI), the RIACA group at the Technical University of Eindhoven, the In2Math project at the University of Koblenz, and the CourseCapsules project at Carnegie Mellon University. They have discussed the initial proposals, represented their materials in OMDoc and in the process refined the format with numerous suggestions and discussions (see `http://www.mathweb.org/~mailists/omdoc` for the archive of the OMDoc mailing list.)

The author specifically would like to thank Serge Autexier, Olga Caprotti, David Carlisle, Claudio Sacerdoti Coen, Arjeh Cohen, Armin Fiedler, Andreas Franke, George Goguadze, Dieter Hutter, Erica Melis, Paul Libbrecht, Martijn Oostdijk, Alberto Palomo Gonzales, Martin Pollet, Julian Richardson, Manfred Riem, and Michel Vollebregt for their input, discussions and feedback from implementations and applications.

The work presented in this report was supported by the “Deutsche Forschungsgemeinschaft” in the special research action “Resource-adaptive cognitive processes” (SFB 378), and a five-year Heisenberg Stipend to the author. Carnegie Mellon University and SRI International have supported the author while working on revisions for version 1.1.

Chapter 2

Mathematical Markup Schemes

Mathematical texts are usually very carefully designed to give them a structure that supports understanding of the complex nature of the objects discussed and the argumentations about them. Of course this holds not only for texts in pure mathematics, but for any argumentative text that contains mathematical notation, in particular for texts from the sciences and engineering disciplines. In such texts the document is often structured according to the argument made and specialized notation (mathematical formulae) is used for the particular objects discussed. In contrast to this, the structure of texts like novels or poems normally obey different (often esthetic) constraints. Therefore, we will use the adjective "mathematical" in an inclusive way to make this distinction on text form, not strictly on the scientific labeling.

The observation, that the task of recovering the semantic structure from the given representation as a written text or a recording is central to understanding, holds for any discourse. For mathematical discourses the structure is so essential that the field has developed a lot of conventions about document form, numbering, typography, formula structure, choice of glyphs for concepts, etc. These conventions have evolved over a long scientific history and carry a lot of the information needed to understand a particular text. However, these conventions were developed for the consumption by humans (mathematicians) and mainly with "ink-on-paper" representations (books, journals, letters) in mind.

In the age of Internet publication and mathematical software systems the "ink-on-paper" target turns out to be too limited in many forms. The universal accessibility of the documents on the Internet breaks the assumption implicit in the design of traditional mathematical documents, that the reader will come from the same (scientific) background as the author and will directly understand the notations and structural conventions used by the author. We can also rely less and less on the assumption that mathematical documents are primarily for human consumption as mathematical software systems are more and more embedded into the process of doing mathematics. This, together with the fact that mathematical documents are primarily produced and stored on computers, has led to the development of specialized markup schemes for mathematics.

In the next sections we will discuss some of the paradigmatic markup schemes setting the stage with general document markup schemes for web-deployed documents. In section 2.3 we will discuss representation formalisms for mathematical objects. We will use section 2.4 to show that extending general document markup approaches with mathematical formulae is not sufficient for a content-based markup of mathematical documents, as it leaves many central aspects of mathematical knowledge and structure implicit.

2.1 Document Markup for the Web

In this section we will discuss some of the paradigmatic markup schemes to get a feeling for the issues involved. Of course, we will over-stress the issues for didactic reasons; due to economic pressures, none of the markup schemes survives in a pure form anymore.

Text processors and desktop publishing systems (think for example of Microsoft Word) are software systems aiming to produce "ink-on-paper" or "pixel-on-screen" representations of documents. They are very well-suited to execute the typographic conventions mentioned above. Their internal markup scheme mainly defines presentation traits like character position, font choice, and characteristics, or page breaks. This is perfectly sufficient for producing high-quality presentations of the documents on paper or on screen, but does not support for instance document reuse (in other contexts or across the development cycle of a text). The problem is that these approaches concentrate on the form and not the function of text elements. Think e.g. of the notorious section renumbering problems in early (WYSIWYG) text processors. Here, the text form of a numbered section heading was used to express the function of identifying the position of the respective section in a sequence of sections (and maybe in a larger structure like a chapter).

This perceived weakness has lead to markup schemes that concentrate more on function than on form. We will take the TEX/LATEX [Knu84, Lam94] approach as a paradigmatic example here. A typical section heading would be specified by something like this:

```
\section[{\TeX}]{The Joy of {\indextoo{\TeX}}}\label{sec:TeX}
```

This specifies the function of the text element: The title of the section should be "The Joy of TEX", which (if needed e.g. in the table of contents) can be abbreviated as "TEX", the word "TEX" is put into the index, and the section number can be referred to using the label `sec:TeX`. To determine from this functional specification the actual form (e.g. the section number, the character placement and font information), we need a document formatting engine, such as Donald Knuth's TEX program [Knu84], and various style declarations, e.g. in the form of LATEX style files [Lam94]. This program will transform the functional specification using the style information into a markup scheme that specifies the form, like DVI [Knu84], or PostScript [Rei87] that can directly be presented on paper or on screen. Note that e.g. renumbering is not a problem in this approach, since the actual numbers are only inferred by the formatter at runtime. This, together with the ability to simply change style file for a different context, yields much more manageable and reusable documents, and has led to a wide adoption of the function-based approach. So that even word-processors like MS Word now include functional elements. Purely form-oriented approaches like DVI or PostScript are normally only used for document delivery.

To contrast the two markup approaches we will speak of presentation markup for markup schemes that concentrate on form and of content markup for those that specify the function and infer the form from that. As we have emphasized before, few markup schemes are pure in the sense of this distinction, for instance LATEX allows to specify traits such as font size information, or using

```
{\bf proof}:...\hfill\Box
```

to indicate the extent of a proof (the formatter only needs to "copy" them to the target format). The general experience in such mixed markup schemes is that presentation markup is more easily specified, but that content markup will enhance maintainability, and reusability. This has led to a culture of style file development (specifying typographical and structural conventions), which now gives us a wealth of style options to choose from in LATEX.

Another member of the content markup family that additionally takes the problem of document metadata into account, i.e. the description of the document itself and the relations to other documents (cf. section 3.1), is the "Simple Generalized Markup Language" SGML [Gol90]. It tries to give the markup scheme a more declarative semantics (as opposed to the purely procedural – and rather baroque – semantics of TeX), to make it simpler to reason about (and thus reuse) documents. It comes with its own style sheet language DSSSL [DuC97] and formatter Jade.

The Internet, where screen presentation, hyperlinking, computational limitations, and bandwidth considerations are much more important than in the "ink-on-paper" world of publishing, has brought about a whole new set of markup schemes. The problems that need to be addressed are that

i) the size, resolution, and color depth of a given screen are not known at the time the document is marked up,

ii) the structure of a text is no longer limited to a linear text with (e.g. numbered) cross-references as in a book or article as Internet documents are in general hypertexts,

iii) the computational resources of the computer driving the screen are not known beforehand. Therefore the distribution of work (e.g. formatting steps) between the client and the server has to be determined at runtime. Finally, the related problem that

iv) the bandwidth of the Internet is ever-growing but limited.

The "Hypertext Markup Language" (HTML [RHJ98]) is a presentation markup scheme that shares the basic syntax with SGML and addresses the problem of variable screen size and hyperlinking by exporting the decision of character placement and page order to a browser running on the client. This ensures the high degree of reusability of documents on the Internet, while conserving bandwidth, so that HTML carries most of the markup on the Internet today. Of course HTML has been augmented with its own (limited) style sheet language CSS [Bos98] that is executed by the browser. The need for content markup schemes for maintaining documents on the server, as well as for specialized presentation of certain text parts (e.g. for mathematical or chemical formulae), has led to a profusion of markup schemes for the Internet, most of which share the basic SGML syntax with HTML. However, due to its origin in the publishing world, full SGML is much too complex for the Internet, and in particular the DSSSL formatter is too unwieldy and resource-hungry for integration into web browsers.

2.2 XML, the eXtensible Markup Language

This diversity problem has led to the development of the unifying XML (eXtensible Markup Language) framework [BPSM97] for Internet markup languages, which we will introduce in more detail in this section. As OMDoc and all mathematical markup schemes discussed here are XML applications (instances of the XML framework), we will go more into the technical details to supply the technical prerequisites for understanding the specification. We will briefly mention XML validation and transformation tools. Readers with prior knowledge of XML can safely skip this section, if the material reviewed in this section is not enough, we refer the reader to [Har01].

Conceptually speaking, XML views a document as a tree of so-called elements. For communication this tree is represented as a well-formed bracketing structure (see Figure 2.5 for an example), where the brackets of an element `el` are represented as `<el>` (opening) and `</el>` (closing); the leaves of this tree are represented as empty elements `<el></el>`, which can be abbreviated as `<el/>`. The element nodes of this tree can be annotated by further information in so-called attributes in opening brackets: `<el visible="no">` might add the information for a formatting engine to hide this element. As a document is a tree, the XML specification mandates that there must be a unique document root, which in OMDoc is the `omdoc` element. Note that all XML parsers will reject a document that is not well-formed XML, e.g. if it contains non-matching element brackets (e.g. a single `<br>`) or multiple document roots.

XML offers two main mechanisms for specifying a subset of trees (or well-bracketed XML documents) as admissible. A document type definition DTD is a context-free grammar for trees[1], that can be used by validating XML parsers to reject XML documents that do not conform to the OMDoc DTD (cf. appendix E). Note that DTDs cannot enforce all constraints that a particular XML application may want to impose on documents. Therefore DTD validation is only a necessary condition for validity with respect to that application. Recently XML has added another grammar formalism: the XML schema language, which can express a slightly stronger set of constraints. Since an XML schema allows stronger document validation, it usually takes normative precedence over the DTD in specifications.

Concretely, an OMDoc document has the general form shown in Figure 2.1. The first line identifies the document as an XML document (version

[1]Actually, a recent extension of the XML standard (XLink) also allows to express graph structures, but the admissibility of graphs is not covered by the DTD. See also section 3.6 on cross-referencing in OMDoc.

```
<?xml version="1.0"?>
<!DOCTYPE omdoc PUBLIC "-//OMDoc//DTD OMDoc V1.1//EN"
                       "http://www.mathweb.org/omdoc/omdoc.dtd" []>
<omdoc>
 ...
</omdoc>
```

Figure 2.1: The Structure of an XML document with DTD.

1.0 of the XML specification). The second line specifies the DTD and the document root OMDoc this it is intended for. In this case the `omdoc` element starting in line three is the root element and will be validated against the DTD found at the URL specified in line two. The last line contains the end tag of the `omdoc` element and ends the file. Every XML element following this line would be considered as another document root.

```
<!DOCTYPE omdoc PUBLIC "-//OMDoc//DTD OMDoc V1.1//EN"
                       "http://www.mathweb.org/omdoc/omdoc.dtd"
  [<!ENTITY % mathmldtd SYSTEM
             "http://www.w3.org/Math/DTD/mathml1/mathml.dtd">
   %mathmldtd;
   <!ELEMENT el (math)><!ATTLIST el att CDATA #REQUIRED>]>
```

Figure 2.2: A Document Type Declaration with Internal Subset

A DTD specified in the `<!DOCTYPE` declaration can be enhanced or modified by adding declarations in the internal subset of the `DOCTYPE` declaration (the empty `[]` in Figure 2.1). In Figure 2.2, we have modified the DTD by declaring that the `el` element has a required attribute `att` and must contain a single `math` child. The declarations for that are contained in the MathMl DTD, which we have included by first declaring the parameter entity `%mathmldtd;` and then referencing it. The internal subset allows to change the DTD grammar for selected elements and to extend the admissible content of elements that were given the content type `ANY` in the original DTD.

The XML schema applicable to an XML document is given by a different mechanism. XML assumes that all elements in a document belong to a given namespace. Technically, an XML namespace is simply a string that uniquely identifies the intended semantics of the elements. It is a URI (uniform resource identifier; a special string that identifies resources on the Internet, see [Har01]). Note that it need not be a valid URL (uniform resource locator; i.e. a pointer to a document provided by a web server.). Namespaces are used to differentiate XML vocabularies or languages, so they can be safely mixed in documents. In principle, every element and attribute name is prefixed by a namespace, i.e. it is a pair `ns:n`, where `ns` is a namespace and `n` is a simple name (that does not contain a colon). We call such a namespace/name pair a qualified name. In most cases, namespaces can be elided or abbreviated when writing XML. Namespaces can be declared on any XML element via the `xmlns` attribute: the element and all its descendents are in this namespace, unless they have a namespace attribute of their own or there is a namespace declaration in a closer ancestor that overwrites it. Similarly, a namespace abbreviation can be declared on any element, it is declared by an attribute declaration of the form `xmlns:nsa="nsURI"`, where `nsa` is a name space abbreviation, i.e. a simple name and `nsURI` is the URI of the namespace. In the scope of this declaration (in all descendants, where it is not overwritten) a qualified name `nsa:n` denotes the qualified name `nsURI:n`.

```
<?xml version="1.0"?>
<omdoc xmlns="http://www.mathweb.org/omdoc"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="omdoc.xsd http://www.mathweb.org/omdoc">
 ...
</omdoc>
```

Figure 2.3: An XML document with XML Schema.

Let us now consider Figure 2.3, which shows the attributes, namespaces, and namespace abbreviations necessary to associate an XML document with an XML schema. The `xmlns` attribute in the `omdoc` element declares that the URI `http://www.mathweb.org/omdoc` is the default namespace for the document, i.e. all element and attribute names without a colon are in this namespace. The attribute `xmlns:xsi` declares the namespace abbreviation `xsi` for the namespace of XML schema instances. Finally, the attribute `xsi:schemaLocation` identifies the XML schema that is relevant for this element (and thus for the document). Note that with this mechanism schemata can be associated with elements (in contrast to DTDs that can only be associated with whole documents), which makes mixing XML vocabularies much simpler.

Since XML elements only encode trees, the distribution of whitespace (including s) in non-text elements has no meaning in XML, and can therefore be added and deleted without effecting the semantics.

XML considers as comments anything between <!-- and --> in a document. They should be used with care, since they are not even read by the XML parser, and therefore do not survive processing by XML applications. Material that is relevant to the document, but not valid XML, e.g. binary data or data that contains angle brackets or elements that are unbalanced or not defined in the DTD can be embedded into CDATA sections. A CDATA section begins with <[CDATA[and suspends the XML parser until the string]]> is found. Another way to include such material is to escape the XML-specific symbols "<", ">", and "&" to <, > and &. According to the XML specification a CDATA section is equivalent to directly including the XML-escaped contents. For instance <[CDATA[a<b³]]> and a<b³ are equivalent, as a consequence an XML application is free to choose the form of its output and the particular form should not relied upon.

XML comes with the XSLT style language transformations [Dea99], that is lightweight enough to allow integration of XSLT-transformers into browsers (they are present in version 6 of Microsoft's Internet Explorer and in version 6 of the Netscape Navigator). XSLT programs or style sheets consist of a set of so-called templates (rules that match certain nodes in the XML tree) that are recursively applied to the input tree to produce the desired output.

2.3 Mathematical Objects and Formulae

The two best-known open markup formats for representing mathematics for the Web are MATHML and OPENMATH. There are various other formats that are proprietary or based on specific mathematical software packages like Wolfram Research's MATHEMATICA. We will not concern ourselves with them, since we are only interested in open formats.

MATHML [CIMP01] is an XML-based markup scheme for mathematical formulae. It has developed out of the effort to include presentation primitives for mathematical notation (in TEX quality) into HTML, and was the first XML application. Since the aim is to do most of the formatting inside the browser, where resource considerations play a large role, it restricts itself to a fixed set of mathematical concepts – the so-called K-12 fragment of mathematics (Kindergarten to 12^{th} grade). K-12 is a large set of commonly used glyphs for mathematical symbols and very general and powerful presentation primitives, as they make up the lower level of TEX. However it does not offer the programming language features of TeX[2] for the obvious computing resource considerations. MathMl is supported by the current versions of the primary commercial browsers MS Internet Explorer and Netscape Navigator by special plug-ins, and natively by MathMl-enabled versions of the open source browsers Mozilla and Amaya.

```
<semantics>
 <mrow>
  <mrow><mo>(</mo><mi>a</mi> <mo>+</mo> <mi>b</mi><mo>)</mo></mrow>
  <mo>&InvisibleTimes;</mo>
  <mrow><mo>(</mo><mi>c</mi> <mo>+</mo> <mi>d</mi><mo>)</mo></mrow>
 </mrow>
 <annotation-xml encoding="MathML-Content">
  <apply><times/>
   <apply><plus/><ci>a</ci> <ci>b</ci></apply>
   <apply><plus/><ci>c</ci> <ci>d</ci></apply>
  </apply>
 </annotation-xml>
</semantics>
```

Figure 2.4: Mixing Presentation and Content MathMl

MathMl also offers content markup for mathematical formulae, a sublanguage called content MathMl to contrast it from the presentation MathMl described above. Furthermore, it offers a specialized `semantics` element that allows to annotate MathMl formulae with content markup, e.g. so that they can be passed on to other mathematical software systems like computer algebra systems. Figure 2.4 shows an example of this for the arithmetical expression $(a+b)(c+d)$. The outermost `semantics` element is a MathMl primitive for annotating MathMl elements with other representations. Here it is used for mixing presentation and content markup. The first child of the `semantics` element is the presentation (this is used by the MathMl-aware browser) which is annotated by `annotation-xml` element, which contains the content markup. Let us first look at the presentation markup. The `mrow` elements are a general grouping device the layout engine uses for purposes of alignment and line-breaking. The `mo` elements marks its content as a mathematical operator and the `mi` element marks its content as a mathematical identifier. The entity reference `⁢` is a character that is not displayed, but stands for the multiplication operator.

[2] TeX contains a full, Turing-complete – if somewhat awkward – programming language that is mainly used to write style files. This is separated out by MathMl to the XslT language it inherits from Xml.

For content markup, the logical structure of the formula is in the center. MathMl uses the `apply` element for function application. In this case the multiplication function `times`, which is applied to the results of the addition function `plus`, applied to some identifiers. Both the elements `times` and `plus` are modeled as empty elements. Note that brackets are not explicitly represented, since they are purely presentational devices and the information is implicit in the structure of the formula and can be deduced from notational conventions. The `mi` element has content counterpart `ci` for content identifier, which conceptually corresponds to a logical variable. The concept of a domain constants is either modeled by a special element (if it is in the K-12 range as `plus` and `times`, there are about 80 others) or by the `csymbol` element.

In contrast to this very rich language that defines the meaning of extended presentation primitives, the OpenMath standard [CC98] builds on an extremely simple kernel (mathematical objects represented by content formulae), and adds an extension mechanism, the so-called **content dictionaries**. These are machine-readable specifications of the meaning of the mathematical concepts expressed by the OpenMath symbols. Just like the library mechanism of the `C` programming language, they allow to externalize the definition of extended language concepts. As a consequence, K-12 need not be part of the OpenMath language, but can be defined in a set of content dictionaries (see `http://www.openmath.org/cdfiles/html/core`). Moreover, OpenMath is purely based on content markup.

The central construct of OpenMath is that of an OpenMath object (`OMOBJ`), which has a tree-like representation made up of applications (`OMA`), binding structures (`OMBIND` using `OMBVAR` to tag the bound variables), variables (`OMV`) and symbols (`OMS`). The `OMS` element carries attributes `cd` and `name` attributes. The `name` attribute gives the name of the symbol. The `cd` attribute specifies content dictionary, a document that defines the meaning of a collection of symbols including the one referenced by the `OMS` itself. As variables do not carry a meaning independent of their local content, `OMV` only carries a `name` attribute. See Figure 2.5 for an example that uses most of the elements.

For convenience, OpenMath also provides other basic data types useful in mathematics: `OMI` for integers, `OMB` for byte arrays, `OMSTR` for strings, and `OMF` for floating point numbers, and finally `OME` for errors. Just like MathMl, OpenMath offers an element for annotating (parts of) formulae with external information (e.g. MathMl or LaTeX presentation): the

OMATTR[3] element, which pairs an OpenMath object with an attribute-value list. To attribute an OpenMath object, it is embedded as the second child in an OMATTR element. The attribute-value list is specified by children of the OMATP element, which is the first child, and has an even number of children: children at even position must be OMS (specifying the attribute), and children at odd positions are the values of the attributes given by their immediately preceding siblings.

The content dictionaries that make up the extension mechanism provided in OpenMath are tied into the object representation by the cd attribute of the OMS element that specifies the defining content dictionary.

OpenMath and MathMl are well-integrated:

- the core content dictionaries of OpenMath mirror the MathMl constructs (see http://www.openmath.org/cdfiles/html/core); there are converters between the two formats.

- MathMl supports the semantics element, that can be used to annotate MathMl presentations of mathematical objects with their OpenMath encoding. Analogously, OpenMath supports the presentation symbol in the OMATTR element, that can be used for annotating with MathMl presentation.

- OpenMath is the designated extension mechanism for MathMl beyond K-12 mathematics: content MathMl supports the csymbol element, which has an attribute definitionURL that points to a document (an OpenMath CD) that defines the meaning of the symbol. The content of the csymbol element is MathMl presentation markup for the symbol.

Figure 2.5 shows OpenMath and content MathMl representations of the law of commutativity for addition on the reals (the logical formula $\forall a, b : R. a + b = b + a$). The mathematical meaning of symbols (that of applications and bindings is known from the folklore) is specified in a set of content dictionaries, which contain formal (FMP "formal mathematical property") or informal (CMP "commented mathematical property") specifications of the mathematical properties of the symbols. For instance, the specification in Figure 2.6 is part of the standard OpenMath content dictionary

[3]Note that the meaning of this element is somewhat underdefined, it is stated in the standard, that any OpenMath compliant application is free to disregard attribuitions (so they do not have a meaning), but in practice, they are often used for specifying e.g. type information.

OpenMath

```
<OMOBJ>
 <OMBIND>
  <OMS cd="quant1" name="forall"/>
  <OMBVAR>
   <OMATTR>
    <OMATP>
     <OMS cd="sts" name="type"/>
     <OMS cd="setname1" name="R"/>
    </OMATP>
    <OMV name="a"/>
   </OMATTR>
   <OMATTR>
    <OMATP>
     <OMS cd="sts" name="type"/>
     <OMS cd="setname1" name="R"/>
    </OMATP>
    <OMV name="b"/>
   </OMATTR>
  </OMBVAR>
   <OMA>
    <OMS cd="relation" 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>
```

MathML

```
<math>
 <apply>
  <forall/>
  <bvar>
   <ci type="real">a</ci>
   <ci type="real">b</ci>
  </bvar>
  <apply>
   <eq/>
   <apply>
    <plus/>
    <ci type="real">a</ci>
    <ci type="real">b</ci>
   </apply>
   <apply>
    <plus/>
    <ci type="real">b</ci>
    <ci type="real">a</ci>
   </apply>
  </apply>
 </apply>
</math>
```

Figure 2.5: $\forall a, b : \mathbf{R}. a + b = b + a.$ in OpenMath and MathML format

`arith1.ocd` for the elementary arithmetic operations. The content of the `FMP` element is actually the OpenMath object in the representation on the left of Figure 2.5, we have abbreviated it here in the usual mathematical notation, and we will keep doing this in the remaining document: wherever an Xml element in a figure contains mathematical notation, it stands for the corresponding OpenMath element.

```
<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>∀ a,b.a + b = b + a</FMP>
</CDDefinition>
```

Figure 2.6: Part of the OpenMath CD arith1.

2.4 Meta-Mathematical Objects

The mathematical markup languages OpenMath and MathMl we have discussed in the last section have dealt with mathematical objects and formulae. This level of support is sufficient for representing very established areas of mathematics like K-12 high school math, where the meaning of concepts and symbols is totally clear, or for the communication needs of symbolic computation services like computer algebra systems, which manipulate and compute objects like equations or groups. The formats either specify the semantics of the mathematical object involved in the standards document itself (MathMl) or in a fixed set of generally agreed-upon documents (OpenMath content dictionaries). In both cases, the mathematical knowledge involved is relatively fixed. Eeven in the case of OpenMath, which has an extensible library mechanism, it is not in itself an object of communication (content dictionaries are mainly background reference for the implementation of OpenMath interfaces).

There are many areas of mathematics, where this level of support is insufficient, because the mathematical knowledge expressed in definitions, theorems (stating properties of defined objects), their proofs, and even whole mathematical theories becomes the primary "object" of mathematical communication. We will call these "objects" meta-mathematical objects, since they contain knowledge *about* mathematical objects. As a consequence it is not the structure of the mathematical objects themselves, but the structure of elements of mathematical knowledge and their interdependencies that is communicated, between mathematicians.

Traditional mathematics has developed a rich set of conventions to mark up the structure of mathematical knowledge in documents. For instance, mathematical statements like theorems, definitions, and proofs like the ones in Figure 2.7 are delimited by keywords (e.g. **Lemma** and $\Box$) or by changes in text font (claims are traditionally written in italics). We will collectively refer to meta-mathematical objects like axioms, definitions, theorems, and proofs as mathematical statements, since they state properties of mathematical objects.

> **Definition 3.2.5** (Monoid)
> A monoid is a semigroup $S = (G, \circ)$ with an element $e \in G$, such that $e \circ x = x$ for all $x \in G$. e is called a left unit of a S.
>
> **Lemma 3.2.5**
> *A monoid has at most one left unit.*
> **Proof**: We assume that there is another left unit $f \ldots$ This contradicts our assumption, so we have proven the claim. □

Figure 2.7: A fragment of a traditional mathematical Document

The large-scale structure of mathematical knowledge is mapped to informal groups of mathematical statements called theories, and often mapped into monographies (titled e.g. "Introduction to Group Theory") or chapters and sections in textbooks. The rich set of relations among such theories is described in the text, sometimes supported by mathematical statements called representation theorems. In fact, we can observe that mathematical texts can only be understood with respect to a particular mathematical context given by a theory which the reader can usually infer from the document. The context can be given explicitly, e.g. by the title of a book such as "Introduction to the Theory of Finite Groups" or implicitly (e.g. by the fact that the e-mail comes from a person that we know works on finite groups, and we can see that she is talking about math).

Mathematical theories have been studied by meta-mathematicians and logicians in the search of a rigorous foundation of mathematical practice. They have been formalized as collections of symbol declarations giving names to the mathematical objects that are particular to the theory and logical formulae, which state the laws governing the properties of the theory. A key research question was to determine conditions for the consistency of mathematical theories. In inconsistent theories (such that do not have models) all statements are vacuously valid[4], and therefore, only consistent theories make interesting statements about mathematical objects. It is one of the key

[4] A statement is valid in a theory, iff it is true for all models of the theory. If there are none, it is vacuously valid.

observations of meta-mathematics that more formulae can be added without endangering consistency, if they can be proven from the formulae already in the theory. As a consequence, consistency of a theory can be determined by classifying the formulae into theorems, i.e. those that have a proof, and axioms – those that do not – and examining consistency of the axioms only. Thus the role of proofs is twofold, they allow to push back the assumptions about the world to simpler and simpler assumptions, and they allow to test the model by deriving consequences of these basic assumptions that can be tested against the data.

A second important observation is that new symbols together with axioms defining their properties can be added to a theory without endangering consistency, if they are of a certain restricted syntactical form. These so-called definitional forms mirror the various types of mathematical definitions (e.g. equational, recursive, implicit definitions). This leads to the so-called principle of conservative extension, which states that conservative extensions to theories (by theorems and definitions) are safe for mathematical theories, and that possible sources for inconsistencies can be narrowed down to small sets of axioms.

Even though all of this has theoretically been known to (meta)-mathematicians for almost a century, it has only been an explicit object of formal study and exploited by mathematical software systems in the last decades. Much of the meta-mathematics has been formally studied in the context of proof development systems like AUTOMATH [dB80] Nuprl [CAB+86], HOL [GM93], MIZAR [Rud92] and ΩMEGA [BCF+97] which utilize strong logical systems that allow to express both mathematical statements and proofs as mathematical objects. Some systems like Isabelle [PN90] and Elf [Pfe91] even allow the specification of the logic language itself, in which the reasoning takes place. Such semi-automated theorem proving systems have been used to formalize substantial parts of mathematics and mechanically verify many theorems in the respective areas. These systems usually come with a library system that manages and structures the body of mathematical knowledge formalized in the system so far.

In software engineering, mathematical theories have been studied under the label of (algebraic) specification. Theories are used to specify the behavior of programs and software components. Under the pressure of industrial applications, the concept of a theory (specification) has been elaborated from a practical point of view to support the structured development of specifications, theory reuse, and modularization. Without this additional structure, real world specifications become unwieldy and unmanageable in practice. Just as in the case of the theorem proving systems, there is a whole zoo of specification languages, most of them tied to particular software systems. They differ in language primitives, theoretical expressivity, and the level of tool support.

Even though there have been standardization efforts, the most recent one being the CASL standard (Common Algebraic Specification Language; see [CoF98]) there have been no efforts of developing this into a general markup language for mathematics with attention to web communication and standards. The OMDOC format attempts to provide a content-oriented markup scheme that supports all the aspects and structure of mathematical knowledge we have discussed in this section. Before we define the language in the next chapter, we will briefly go over the consequences of adopting a markup language like OMDOC as a standard for web-based mathematics.

2.5 An Active Web of Mathematical Knowledge

It is a crucial – if relatively obvious – insight that true cooperation of mathematical services is only feasible if they have access to a joint corpus of mathematical knowledge. Moreover, having such a corpus would allow to develop added-value services like

1. cut and paste on the level of computation (take the output from a web search engine and paste it into a computer algebra system),

2. automatically proof checking published proofs,

3. math explanation (e.g. specializing a proof to an example that simplifies the proof in this special case),

4. semantical search for mathematical concepts (rather than keywords),

5. data mining for representation theorems (are there unnoticed groups out there),

6. classification: given a concrete mathematical structure, is there a general theory for it?

As the online mathematical knowledge is presently only machine-readable, but not machine-understandable, all of these services can currently only be performed by humans, limiting the accessibility and thus the potential value of the information. Services like this will transform the now passive and human-centered fragement of the Internet that deals with mathematical content, into an active (by the services) web of mathematical knowledge (we will speak of mathweb for this vision).

Of course, this promise of activating a web of knowledge is in no way limited to mathematics, and the task of transforming the current presentation-oriented world-wide web into a "semantic web" [Lee98] has been identified as one of the main challenges by the world wide web consortium (W3C, the fundamental standardizing body for the WWW, see `http://www.w3c.org`).

The direct applications of MathWeb (apart from the general effect towards a semantic web) are by no means limited to mathematics proper. Until now, the MathMl working group in the W3C has led the way in many web technologies (presenting mathematics on the web taxes the current web technology to its limits); the endorsement of the MathMl standard by the W3 Committee is an explicit testimony to this. We expect that the effort of creating an infrastructure for digital mathematical libraries will play a similar role, since mathematical knowledge is the most rigorous and condensed form of knowledge, and will therefore pinpoint the problems and possibilities of the semantic web.

All modern sciences have a strongly mathematicised core, and will benefit. The real market and application area for the techniques developed in this project lies with high-tech and engineering corporations like Airbus Industries, Daimler Chrysler, Phillips, ... that rely on huge formula databases. Currently, both the content markup as well as the added-value services alluded to above are very underdeveloped, limiting the usefulness of the vital knowledge. The content-markup aspect needed for mining this information treasure and obtaining a competitive edge in development is exactly what we are attempting to develop in OMDoc.

Chapter 3

OMDoc Elements

In this chapter, we define the OMDoc language features and their meaning. We motivate them from either particular structures in mathematical documents or from processing needs of computer-supported mathematics and give concrete examples based on these.

The content of this chapter is normative for the OMDoc format; an OMDoc document is valid as an OMDoc document, iff it meets all the constraints imposed in this chapter. OMDoc applications will normally presuppose valid OMDoc documents, and only claim to exhibit the intended behavior on such. Note that OMDoc validity does not yet imply that documents make sense mathematically, only that they can be safely processed by OMDoc applications.

Part of the constraints imposed by the OMDoc definition can be checked by suitable XML tools. The namespace URI for OMDoc is `http://www.mathweb.org/omdoc`, referring to this specification that gives the OMDoc elements their meaning. We have developed a document type definition (DTD) (cf. appendix E) that can be used by an XML parser to partially validate OMDoc documents.

In the rest of the chapter we will introduce the XML elements used by the OMDoc language grouped by thematic role. Before we come to the mathematical elements proper, we detail OMDoc metadata in section 3.1. This "data about data" can be used to annotate many OMDoc elements by descriptive and administrative information that facilitates navigation and organization.

In section 3.2 we define various mathematical statements, i.e. elements that allow to mark up mathematical forms like definitions, theorems, proofs, and examples; that have long been considered paradigmatic of mathematical documents like textbooks and papers.

In section 3.3 we will introduce markup for simple mathematical theories, which group mathematical statements and provide a notion of context for mathematical statements. Here we build on concepts from the field of algebraic specification, where structured representation of large corpora of formal scientific knowledge about the meaning of programs and the mathematical structures used in them has been studied extensively.

But mathematical documents contain more than this: e.g. exercises, applets, notation declarations are intermixed with explanatory text. We deal with this in see section 3.4 to 3.5.2.

In section 3.6 we will address the problem of identifying and referencing OMDoc elements in larger collections of documents. The approach will be to use special uniform resource identifiers (URI), for the examples until then we will only use local (intra-document) references, which are a special case.

The Xml root root element of the OMDoc format is the `omdoc` element, it contains all other elements described in the rest of this chapter. We call an OMDoc element a top-level element, if it can appear as a direct child the `omdoc` element. The `omdoc` element has a required attribute `id` that can be used to reference the whole document. The `version` attribute is used to specify the version of the OMDoc format the file conforms to. It is fixed to `1.1` by the OMDoc document type definition in appendixE. This will prevent validation with a different DTD. Similarly, the `xmlns` attribute fixes the namespace URI for OMDoc to `http://www.mathweb.org/omdoc`. Furthermore, the `omdoc` element has the attributes `type`, which will be presented in section 3.4.1 and `catalogue` (see section 3.6).

3.1 Metadata for Mathematical Elements

The World Wide Web was originally built for human consumption, and although everything on it is machine-readable, most of it is not machine-understandable. The accepted solution is to use metadata (data about data) to describe the data contained on the Web. In OMDoc, we use one of the best-known metadata schemas for documents – the Dublin Core (cf. `http://purl.org/dc/`), which is the basis for many metadata formats, such as the Xml resource description format (RDF). The purpose of annotating metadata in OMDoc is to facilitate the administration of documents, e.g. digital rights management, and to generate input for metadata-based tools, e.g. RDF-based navigation and indexing of document collections.

The `metadata` element contains elements for Dublin Core metadata and an optional `extradata` element for user-defined or application-specific metadata. The `extradata` element may contain arbitrary well-formed XML, as long as its elements are declared in the internal subset of the document type definition (see the discussion on page 10). Figure 3.1 shows an example of pedagogical metadata in the `extradata` element. Note that other OMDoc applications will not act on it since such data is application-specific; they will preserve it verbatim if the output format allows it, and ignore it otherwise. The OMDoc `metadata` element can be used to provide information

```
<!DOCTYPE omdoc PUBLIC "-//OMDoc//DTD OMDoc V1.1//EN"
                      "-//OMDoc/"http://www.mathweb.org/omdoc/omdoc.dtd"
  [<!ELEMENT abstraction EMPTY><!ATTLIST abstraction level CDATA #REQUIRED>
   <!ELEMENT difficulty EMPTY><!ATTLIST difficulty level CDATA #REQUIRED>]>
...
<metadata>
 <extradata>
  <abstraction level="high"/>
  <difficulty level="simple"/>
 </extradata>
</metadata>
```

Figure 3.1: Enabling application-specific `extradata`

about the document as a whole (as a child of the `omdoc` element), as well as about specific fragments of the document, and even about the top-level mathematical elements in OMDoc. We will use the fourth column labeled "DC" in quick-reference tables like Figure 3.5 to indicate whether an OMDoc element can have a `metadata` element as the first child.

3.1.1 The Dublin Core Elements

In the following we will describe individual Dublin Core metadata elements. The descriptions below are adapted from `http://www.ietf.org/rfc/rfc2413.txt`, and augmented for the application in OMDoc where necessary. One particular adaption is that `metadata` can be used to annotate many mathematical elements.

The OMDoc `metadata` element can contain any number of instances of any Dublin Core element in any order. In fact, multiple instances of the same element type (multiple `Creator` elements, for example) can be interspersed with other elements without change of meaning.

`Title` The title of the element. The `Title` element can contain mathematical formulae as `OMOBJ` elements. In fact, it may contain the same children as the `CMP` defined in section 3.2.1 (we call this content mathematical text).

The `Title` element has an `xml:lang` attribute that specifies the language of the content. Multiple `Title` elements inside a `metadata` element are assumed to be translations of each other; they have to be unique per `xml:lang` attribute.

`Creator` A primary creator or author of the publication. Additional contributors whose contributions are secondary to those listed in `Creator` elements should be named in `Contributor` elements. Document with multiple co-authors should provide multiple `Creator` elements, each containing one author. The order of `Creator` elements is presumed to define the order in which the creators' names should be presented.

As markup for names across cultures is still un-standardized, OMDoc recommends that the content of the `Creator` elements hold the text for a single name as it would be presented to the user. The OMDoc DTD supplies a parameter entity `%DCperson` that can be suitably redefined for application-specific markup schemes.

The `Creator` elements has an optional attribute `id` so that it can be cross-referenced and a `role`, which can take the values defined in the next section to further classify the concrete contribution to the element.

`Contributor` A party whose contribution to the publication is secondary to those named in `Creator` elements. Apart from the significance of contribution, the semantics of this element is identical to that of `Creator`, it has the same restriction content and carries the same attributes plus an optional `xml:lang` attribute that specifies the target language in case the contribution is translation (i.e. if the `role` is `'trl'`).

`Subject` This element includes an arbitrary phrase or keyword, the `xml:lang` is used for the language. Multiple instances of the `Subject` element are supported per `xml:lang` for multiple keywords.

`Description` A mathematical text describing the containing element's content; the `xml:lang` is used for the language. This metadata element is only recommended for `omdoc` elements that do not have a `CMP` group (see section 3.2.1), or if the description is significantly shorter than the one in the `CMP`s.

`Publisher` The entity for making the document available in its present form, such as a publishing house, a university department, or a corporate entity. This element only applies if the `metadata` is directly inside the root element (`omdoc`) of a document.

`Date` The date and time a certain action was performed on the document. The content is in the format defined by XML Schema data type `dateTime` (see `http://www.w3.org/TR/xmlschema-2/#dateTime` for a discussion), which is based on the ISO 8601 norm for dates and times. Concretely, the format is `CCYY-MM-DDThh:mm:ss` where "`CC`" represents the century, "`YY`" the year, "`MM`" the month and "`DD`" the day, preceded by an optional leading "`-`" sign to indicate a negative number. If the sign is omitted, "`+`" is assumed. The letter "`T`" is the date/time separator and "`hh`", "`mm`", "`ss`" represent hour, minute and second respectively. Additional digits can be used to increase the precision of fractional seconds if desired i.e the format "`ss.sss...`" with any number of digits after the decimal point is supported. The `Date` element has the attributes `action` and `who` to specify who did what. The value of `who` is a reference to a `Creator` or `Contributor` element and `action` is a keyword for the action undertaken. Recommended values include `'updated'`, `'new'`, `'imported'`, `'frozen'`, `'normed'`.

`Type` Dublin Core defines a vocabulary for the document types in `http://dublincore.org/docum`
The best fit for OMDoc is one of the following

- `Dataset` Dublin Core defines this as "*A dataset is information encoded in a defined structure (for example lists, tables, and databases), intended to be useful for direct machine processing*"
- `Text` Dublin Core defines this as "*A text is a resource whose content is primarily words for reading. For example – books, letters, dissertations, poems, newspapers, articles, archives of mailing lists. Note that facsimiles or images of texts are still of the genre text.*"

The more appropriate should be selected for the element described. If it is mainly as formal mathematical formulae, then `Dataset` is better, if it is mainly given as text, then `Text` should be used.

`Format` The physical or digital manifestation of the resource. Dublin core suggests using MIME types. Following [MSLK01] we fix this to the string `application/omdoc+xml` as a (non-registered) MIME type for OMDoc.

`Identifier` A string or number used to uniquely identify the element. This is a string that uniquely identifies this document or element. As this is largely superseded by the identification scheme discussed in section 3.6 it should only be used for public identifiers like ISBN or ISSN numbers. The numbering scheme can be specified in the `scheme` attribute (it has `'isbn'` as a default).

`Source` Information regarding a prior resource from which the publication was derived. We recommend using either a URI or a scientific reference including identifiers like ISBN numbers.

`Relation` Information regarding the relation of this document to others.

`Language` If there is a primary language of the document, this can be specified here. The content must be an ISO 639 two-letter language specifier.

`Rights` Information about rights held in and over the resource or a reference to a such a statement. Typically, a `Rights` element will contain a rights management statement for the resource, or reference a service providing such information. Rights information often encompasses Intellectual Property Rights (IPR), Copyright, and various Property Rights. If the `Rights` element is absent, no assumptions can be made about the status of these and other rights with respect to the resource.

Note that Dublin Core also defines a `Coverage` element that specifies the place or time which the publication's contents addresses. This does not seem appropriate for the mathematical content of OMDoc, which is largely independent of time and geography.

The `metadata` elements can be added to many of the OMDoc elements described in this chapter, including grouping elements that can contain others that contain `metadata`. To avoid duplication, OMDoc assumes a priority-union semantics of Dublin Core `metadata`. A Dublin Core element, e.g. `Creator` that is missing in in a lower `metadata` declaration (i.e. there is no element of the same name and with the same attributes) is inherited from the upper ones. So in Figure 3.2, the two boxes are equivalent, since the metadata in theory `th1` and in definition `d1` is inherited from the main declaration in the top-level `omdoc` element. If there is a metadata element of the same name present, nothing is inherited.

```
<omdoc id="o1">
 <metadata>
  <Creator>MiKo</Creator>
 </metadata>

 <theory id="th1">




  <symbol id="s1"/>
  <definition id="d1"/>





 </theory>

 <theory id="th2">
  <metadata>
   <Creator>Paul</Creator>
  </metadata>
  <symbol id="s2">
  <definition id="d1">
   <metadata>
    <Creator>MiKo</Creator>
   </metadata>
  </definition>
 </theory>
</omdoc>
```

⟷

```
<omdoc id="o1">
 <metadata>
  <Creator>MiKo</Creator>
 </metadata>

 <theory id="th1">
  <metadata>
   <Creator>MiKo</Creator>
  </metadata>

  <symbol id="s1"/>
  <definition id="d1">
   <metadata>
    <Creator>MiKo</Creator>
   </metadata>
  </definition>
 </theory>

 <theory id="th2">
  <metadata>
   <Creator>Paul</Creator>
  </metadata>
  <symbol id="s2">
  <definition id="d1">
   <metadata>
    <Creator>MiKo</Creator>
   </metadata>
  </definition>
 </theory>
</omdoc>
```

Figure 3.2: Inheritance of `metadata`

3.1.2 Roles in Dublin Core Metadata

Because the Dublin Core metadata fields for `Creator` and `Contributor` do not distinguish roles of specific contributors (such as author, editor, and illustrator), we will follow the Open eBook [Gro99] specification and use optional `role` attributes for this purpose. The attribute values `role` attribute is adapted for OMDoc from the MARC relator code list (Machine-Readable Cataloging Record, see `http://www.loc.gov/marc/relators/re0002r1.html`)

`'aut'` (Author) Use for a person or corporate body chiefly responsible for the intellectual or artistic content of an element. This term may also be used when more than one person or body bears such responsibility.

'ant' (Scientific antecedent) Use for the author responsible for a work upon which the element is based.

'clb' (Collaborator) Use for a person or corporate body that takes a limited part in the elaboration of a work of another author or that brings complements (e.g., appendices, notes) to the work of another author.

'edt' (Editor) Use for a person who prepares a document not primarily his/her own for publication, such as by elucidating text, adding introductory or other critical matter, or technically directing an editorial staff.

'ths' (Thesis advisor) Use for the person under whose supervision a degree candidate develops and presents a thesis, memoir, or text of a dissertation.

'trc' (Transcriber) Use for a person who prepares a handwritten or typewritten copy from original material, including from dictated or orally recorded material. This is also the role (on the `Creator` element) for someone who prepares the OMDoc version of some mathematical content.

'trl' (Translator) Use for a person who renders a text from one language into another, or from an older form of a language into the modern form. The target language can be specified by the `xml:lang`.

```
<metadata>
 <Title>The Joy of Jordan C* Triples</Title>
 <Creator role="aut">A</Creator>
 <Contributor role="edt">R</Contributor>
 <Contributor role="trc">S</Contributor>
</metadata>
```

Figure 3.3: A Document with editor (`edt`) and transcriber (`trc`)

Let us now consider two examples to fortify our intuition. As OMDoc documents are often used to formalize existing mathematical texts for use in mechanized reasoning and computation systems, it is sometimes subtle to specify authorship. We will discuss some typical examples to give a guiding intuition. Figure 3.3 shows metadata for a situation where editor R gives the sources (e.g. in LaTeX) of a document D written by author A to secretary S for conversion into OMDoc format. In Figure 3.4 researcher R formalizes

```
<metadata>
 <Title>Natural Numbers</Title>
 <Creator role="aut">R</Creator>
</metadata>
```

```
<metadata>
 <Title>Natural Numbers</Title>
 <Creator role="aut">R</Creator>
 <Contributor role="ant">A</Contributor>
 <Source>B</Source>
</metadata>
```

Figure 3.4: A formalization with scientific antecedent (`ant`)

the theory of natural numbers following the standard textbook B (written by author A). In this case we recommend something like the left declaration for the whole document and the right one for specific math elements, e.g. a definition inspired by or adapted from one in book B.

Element	Attributes		D	Content
	Required	Optional	C	
metadata	id	inherits, style	–	(dc-element)*, extradata
extradata			–	ANY
Creator		id, style, role	–	%DCperson
Contributor		id, style, role	–	%DCperson
Title		xml:lang	–	CMP content
Subject		xml:lang	–	CMP content
Description		xml:lang	–	CMP content
Publisher		id, style	–	ANY
Date		action, who	–	ISO 8601
Type			–	fixed: "Dataset" or "Text"
Format			–	fixed: "xml,x-omdoc"
Identifier		scheme	–	ANY
Source			–	ANY
Language			–	ISO 8601
Relation			–	ANY
Rights			–	ANY

Figure 3.5: The OMDoc metadata

3.2 Mathematical Statements

In this section we will define the OMDoc elements for mathematical statements. We call mathematical forms like axioms, definitions, theorems, and examples statements, since they are the basic units used to state properties about mathematical objects. Axioms and definitions state the meaning of symbols, that can later be used to build up other mathematical objects. Theorems state properties about objects that can be proven from the axioms and definitions, and are therefore safe to assume.

Before we go into the particular features of the different classes of statements, let us discuss their common parts.

3.2.1 Specifying Mathematical Properties

As we have said before, all mathematical statements state properties of mathematical objects. This is done either informally (given in the rigorous version of natural language interspersed with mathematical formulae sometimes called mathematical vernacular) or formally (as logical formulae), or both. For the informal representation of the content of mathematical statements, we use groups of `CMP` elements, for the formal content groups of `FMP` elements.

Element	Attributes		D	Content
	Required	Optional	C	
FMP		logic	−	(assumption*, conclusion*) | OMOBJ
assumption	id	style	+	CMP*, OMOBJ?
conclusion	id	style	+	CMP*, OMOBJ?
CMP		xml:lang	−	ANY
with	id	style	−	CMP content
omtext	id	type, for, style	+	CMP+, FMP*

Figure 3.6: The OMDoc elements for specifying mathematical properties

An `FMP` element is the general element for representing formal mathematical content as OpenMath objects[1]. `FMP`s always appear in groups, which can differ in the value of their `logic` attribute, which specifies the logical formalism. The value of this attribute specifies the logical system used in formalizing the content. All members of the multi-logic `FMP` group have to

[1] The name is taken from OpenMath content dictionaries for continuity reasons

formalize the same mathematical object or property, i.e. they have to be translations of each other.

As logical formulae often come as sequents, i.e. as sets of conclusions drawn from a set of assumptions, OMDoc also allows the content of an FMP to be a (possibly empty) set of assumption elements followed by a conclusion. The intended meaning is that the FMP asserts that one of the conclusions is entailed by the assumptions in the current context. As a consequence, <FMP><conclusion>A</conclusion></FMP> is equivalent to <FMP>A</FMP>, where A is an OpenMath representation of a mathematical formula. The assumption and conclusion elements allow to specify the content by an OpenMath object or in natural language (using CMPs).

CMP elements may contain arbitrary text interspersed with the elements OMOBJ, omlet, ref and with, no other elements are allowed.[2] The OMOBJ elements are used for mathematical objects, the omlet elements for applets (see section 3.4.3), and the with elements for supplying text fragments with attributes for referencing and presentation. In particular, presentation elements like paragraphs, emphases, itemizes, ... are forbidden, since OMDoc is concerned with content markup. Generating presentation markup from this is the duty of specialized presentation components, e.g. XslT style sheets, which can base their decisions on presentation information (see section 3.5.2). The with element is new in OMDoc1.1, it allows the same content as the CMP element, so that it can be transparently nested in there. It has the attributes id for referencing the text fragment (e.g. for creating an index) and style to associate presentation information with it. We anticipate further development on the usage of this element, so that the set of attributes is likely to be extended.

CMP elements have an xml:lang attribute that specifies the language they are written in. Thus using multilingual groups of CMP elements with different languages can promote OMDoc internationalization. Conforming with the Xml recommendation, we use the ISO 639 two-letter country codes (en $\widehat{=}$ English, de $\widehat{=}$ German, fr $\widehat{=}$ French, nl $\widehat{=}$ Dutch,...). This optional attribute has the default "'en'", so that if no xml:lang is given, then English is assumed. Of course it is forbidden to have more than one CMP per value of xml:lang per mathematical statement, moreover, CMPs that are siblings must be translations of each other, and must carry the same meaning as the logical formula in the FMP they are sibling to.

[2]The DTD provides a parameter entity!parameter %alsoinCMP that can be specialized in the local subset of the DTD to accomodate for additional elements. Note that these will not be supported by the generic tools.

```
<metadata>
 <Creator role="aut">Michael Kohlhase</Creator>
 <Contributor role="trl" xml:lang="de">Michael Kohlhase</Contributor>
 <Contributor role="trl" xml:lang="fr">Paul Libbrecht</Contributor>
</metadata>
<CMP xml:lang="en" format="omtext">
 Let <OMOBJ id="set"><OMV name="V"/></OMOBJ> be a set.
 A unary operation on <OMOBJ xref="set"/> is a function
 <OMOBJ id="func"><OMV name="F"/></OMOBJ> with
 <OMOBJ id="im">
  <OMA>
   <OMS cd="relations1" name="eq"/>
   <OMA><OMS cd="fns1" name="domain"/><OMV name="F"/></OMA>
   <OMV name="V"/>
  </OMA>
 </OMOBJ> and
 <OMOBJ id="ran">
  <OMA>
   <OMS cd="relations1" name="eq"/>
   <OMA><OMS cd="fns1" name="range"/><OMV name="F"/></OMA>
   <OMV name="V"/>
  </OMA>
 </OMOBJ>.
</CMP>
<CMP  xml:lang="de" format="omtext">
 Sei <OMOBJ xref="set"/> eine Menge.
 Eine unäre Operation ist eine Funktion <OMOBJ xref="fun"/>,
 so daß <OMOBJ xref="im"/> und <OMOBJ xref="ran"/>.
</CMP>
<CMP  xml:lang="fr" format="omtext">
 Une opération unaire sûr <OMOBJ xref="set"/> est une
 fonction <OMOBJ xref="fun"/> avec <OMOBJ xref="im"/> et
 <OMOBJ xref="ran"/>.
</CMP>
<FMP>∀V, F.binop(F, V) ⇔ Im(F) = V ∧ Dom(F) = V</FMP>
```

Figure 3.7: A multilingual group of CMP elements with FMP and metadata

Figure 3.7 shows an example of such a multilingual group. It also shows an extension that OMDoc makes to OpenMath elements. OMDoc adds (optional) attributes id and xref attributes to the OpenMath elements OMOBJ, OMA, OMBIND and OMATTR for the purpose of cross-referencing (see section E). This facility is convenient in two ways:

- it facilitizes multi-language support: Only the language-dependent parts of the text have to be re-written, the (language-independent) formulae can simply be re-used by cross-referencing.

- formulae can be represented as directed acyclic graphs (DAG) preventing exponential blowup of the encoding, since formula parts can be re-used.

Note that the extension (which MathMl provides by default) is licensed by the OpenMath standard, since pure OpenMath trees can be generated automatically from it.

Mathematical documents often contain text passages that cannot strictly be classified into the mathematical statements defined in the rest of this section. Such passages can be motivations, further explanations, historical remarks, and the like, and are modeled with a special element `omtext` in OMDoc.

`omtext` elements can appear at the top level (i.e. inside `omdoc`, `omgroup`, and `theory` elements). They have an `id` attribute, so that they can be cross-referenced. `omtext` elements basically serve to group `CMP` elements into multilingual groups and supply them with `metadata` information. Finally, `omtext` elements may contain (optional) `FMP` elements with an OpenMath object or a logical sequent that formally represents the meaning of the descriptive text in the `CMP`s (if that is feasible). In this light, the OMDoc fragment in Figure 3.7 could also be the content of an `omtext` element; the only difference to a mathematical statement is, that the purpose as a mathematical statement cannot be determined as one of the above. The purpose can be described by the optional attribute `type`, which can take the values `'abstract'`, `'introduction'`, `'conclusion'`, `'comment'`, `'thesis'`, `'antithesis'`, `'elaboration'`, `'motivation'`, `'evidence'` with the obvious meanings. In the last five cases `omtext` also has the extra attribute `for`, since these are in reference to another OMDoc element.

As Xml comments (i.e. anything between "`<!--`" and "`-->`" in a document) are not even read by the Xml parser, anything that would normally go into comments should be modeled with an `omtext` element (`type 'comment'`) or with the `ignore` element for persistent comments, i.e. comments that survive processing.

This element should be used if the author wants to comment the OMDoc representation, but the end user should never see their content, so that OMDoc text elements are not suitable.

3.2.2 Symbols, Definitions, and Axioms

Now we come to the mathematical statements that determine the meaning of mathematical objects. Axioms and definitions fix the meaning of (groups of) symbols. It is sufficient to determine the semantics of symbols, since they are the atomic units from which complex mathematical objects are built up.

The `symbol` element specifies a symbol for a mathematical concept, such as 1 for the natural number "one", + for addition, = for equality, or `group` for the property of being a group. It has an `id` attribute which uniquely identifies it in a theory (see section 3.3) and an attribute `kind` that can take the values `'type'` (for objects that denote sets that are used in type systems), `'sort'` (for sets that are inductively built up from constructor symbols; see section 3.2.6), and `'object'` (the default; for all other symbols). The attribute `scope` takes the values `'global'` and `'local'`, and allows a simple specification of visibility conditions: if a `symbol` has `scope` `'local'` then it is not exported outside the theory.

The children of the `symbol` element consist of a multilingual group of `commonname` elements (parameterized by a `xml:lang` attribute) and a set of `type` elements (parameterized by the `system` attribute).

The `commonname` elements contain keyword or simple phrases, they have the same content model as the `CMP` elements. If the document containing their parent `symbol` element were stored in a data base system, it could be looked up by the content of its `commonname` children. As a consequence of the presence of the `commonname`, the symbol `id` need only be used for identification. In particular, it need not be mnemonic, though it can be, and it need not be language-dependent, since this can be done by suitable `commonname` elements. In Figure 3.8 we have a symbol declaration for the property of being monoid.

The `type` elements allow to specify type information for the symbol they are contained in. They can also appear outside of `symbol` elements on top-level, then they specify type information for the symbol referenced in its `for` attribute. The attribute `system` contains a token string that names the type system which interprets the content. The content of a `type` element is a formal representation of the type of the symbol as a mathematical object of the type system specified by the attribute `system`. It is not an error to have more than one `type` declaration per `system` attribute in a `symbol` element, this just means that the object has more than one type in the respective type system. In the example in Figure 3.8, the type of `monoid` characterizes a monoid as a three-place predicate (taking as arguments the base set, the operation and a neutral element).

The relation between the components of a monoid would typically be specified by a set of axioms (e.g. stating that the operation is commutative). For this purpose OMDoc uses the axiom element, which allows a multilingual set of CMPs and an (optional) FMP as children, which express the mathematical content of the axiom. Apart from the id attribute, axiom elements may have a generated-by attribute, which points to another OMDoc element (e.g. an adt, see section 3.2.6) which subsumes it, since it is a more succinct representation of the same mathematical content.

```
<symbol id="monoid">
 <commonname xml:lang="en">monoid</commonname>
 <commonname xml:lang="de">Monoid</commonname>
 <commonname xml:lang="it">monoide</commonname>
 <type system="simply-typed">
   set[any]→(any→any→any)→any→bool
 </type>
</symbol>
<definition id="mon.d1" for="monoid" type="implicit">
 <CMP xml:lang="en">
  A structure (M,∗,e), in which (M,∗) is a semi-group
  with unit e is called a monoid.
 </CMP>
</definition>
```

Figure 3.8: An OMDoc symbol Declaration with definition

The definition elements give meanings to (groups of) symbols (declared in a symbol element elsewhere) in terms of already defined ones. For example the number 1 can be defined as the successor of 0 (specified by the Peano axioms). Addition is usually defined recursively, etc.

Both axioms and definitions can be used to give meaning to sets of symbols. Both contain a multilingual CMP group to describe the meaning in natural language. The also contain a multi-logic FMP group that expresses this as a logical formula. In contrast to axioms which only constrain the possible interpretations of a symbol, definitions are used with the intention that they totally fix the meaning. As a consequence OMDoc definition elements are more complex, since they provide an infrastructure to ensure this. In particular, the definition element supports several kinds of definition mechanisms specified in the type attribute:

'simple' In this case, the definition contains an OpenMath object that can be substituted for the symbol specified in the for attribute of the

Element	Attributes		D	Content
	Required	Optional	C	
symbol	id	kind, scope, style	+	CMP*, (commonname | type | selector)*
commonname		xml:lang	–	CMP content
type	system	id, for, style	–	CMP*, OMOBJ
axiom	id	generated-by, style	+	symbol*,CMP*,FMP*
definition	id, for	just-by, type, generated-by, style	+	CMP*, (FMP* | requation+ | OMOBJ)?, measure?, ordering?
requation		id, style	–	pattern, value
pattern			–	OMOBJ
value			–	OMOBJ
measure			–	OMOBJ
ordering			–	OMOBJ

Figure 3.9: Symbols, Axioms, and Definitions in OMDoc

definition. Figure 3.10 gives an example of a (simple) definition of a the number one from the successor function and zero.

`'inductive'` The OpenMath object contains a formula, but in contrast to the case of `'simple'` definitions this can contain occurrences of the symbol specified in the `for` attribute of the `definition`. To guarantee termination of the recursive instantiation (this is necessary to ensure well-definedness), it is possible to specify a measure function in the form of an OpenMath object and well-founded ordering. The optional `measure` and `ordering` elements allow to do this in form of OpenMath objects. Alternatively, a termination proof can be specified in the `just-by` attribute of the definition.

`'recursive'` This is a variant of the `'inductive'` case above. It defines functions by a set of recursive equations (in `requation` elements) whose left and right hand sides are specified by the `pattern` and `value` elements. Both elements `pattern` and `value` hold an OpenMath element. The intended meaning of the defined symbol is, that the content of the `value` element (with the variables suitably substituted) can be substituted for a formula that matches the content of the `pattern` element. Figure 3.11 gives an example of a a recursive definition of

```
<symbol id="one"/>
<definition id="one.def" for="one" type="simple">
 <CMP><OMOBJ><OMS cd="nat" name="one"/></OMOBJ> is the successor of
      <OMOBJ><OMS cd="nat" name="zero"></OMOBJ>.</CMP>
 <FMP>
   <OMOBJ>
     <OMA><OMS cd="relation1" name="eq"/>
       <OMS cd="nat" name="one"/>
       <OMA xref="one.1"/>
     </OMA>
   </OMOBJ>
 </FMP>
 <OMOBJ>
   <OMA id="one.1">
     <OMS cd="int" name="suc"/>
     <OMS cd="nat" name="zero">
   </OMA>
 </OMOBJ>
</definition>
```

Figure 3.10: A simple OMDoc `definition`.

addition on the natural numbers.

Evidence of termination of the recursive replacement of values for patterns can be provided in the `measure` and `ordering` elements or the `just-by` attribute of the dominating `definition`.

`'implicit'` Here, the `FMP` elements contain a set of logical formulae that uniquely determines the value of the symbols that are specified in the `for` attribute of the definition. The necessary proof of unique existence can be specified in the `just-by` attribute. We give an example of an implicit definition in Figure 3.12.

3.2.3 Assertions and Alternatives

OMDoc uses the `assertion` element for all statements (proven or not) about mathematical objects (see Figure 3.13). Traditional mathematical documents discern various kinds of these: theorems, lemmata, corollaries, conjectures, problems, etc. These all have the same structure (formally, a closed logical formula). Their differences are largely pragmatic (theorems are normally more important in some theory than lemmata) or proof-theoretic (conjectures become theorems once there is a proof). Therefore, we represent

```
<definition id="rec-plus" for="plus" type="recursive">
 <commonname>addition</commonname>
 <CMP>Addition is defined by recursion on the second argument</CMP>
 <requation>
  <pattern>
   <OMOBJ>
    <OMA>
     <OMS cd="nat" name="plus"/>
     <OMV name="X"/>
     <OMS cd="nat" name="zero"/>
    </OMA>
   </OMOBJ>
  </pattern>
  <value><OMOBJ><OMV name="X"/></OMOBJ></value>
 </requation>
 <requation>
  <pattern>
   <OMOBJ>
    <OMA>
      <OMS cd="nat" name="plus"/>
      <OMV name="X"/>
      <OMA><OMS cd="nat" name="succ"/><OMV name="Y"/></OMA>
     </OMA>
   </OMOBJ>
  </pattern>
  <value>
   <OMOBJ>
    <OMA>
     <OMS cd="nat" name="succ"/>
     <OMA><OMS cd="nat" name="plus"/><OMV name="X"/><OMV name="Y"/></OMA>
    </OMA>
   </OMOBJ>
  </value>
 </requation>
</definition>
```

Figure 3.11: A recursive definition of addition

```
<definition id="exp-def" type="implicit" just-by="exp-well-def">
 <FMP>exp' = exp ∧  exp(0) = 1</FMP>
</definition>
<assertion id="exp-well-def">
 <CMP>
  There is at most one differentiable function that solves the
  differential equation in Definition <ref xref="exp-def"/>.
 </CMP>
</assertion>
```

Figure 3.12: An implicit definition of the exponential function

them in the general `assertion` element and leave the type distinction to a `type` attribute, which can have the following values (note that this is only a soft classification of assertions, based more on mathematical practice than on hard distinctions).

`'theorem'`, `'proposition'` (an important assertion with a proof) Note that the meaning of the `type` (in this case the existence of a proof) is not enforced by OMDoc applications. It can be appropriate to give an assertion the `type 'theorem'`, if the author knows of a proof (e.g. in the literature), but has not formalized it in OMDoc yet.

`'lemma'` (a less important assertion with a proof) The difference of importance specified in this `type` is even softer than the other ones, since e.g. reusing a mathematical paper as a chapter in a larger monograph, may make it necessary to downgrade a theorem (e.g. the main theorem of the paper) and give it the status of a lemma in the overall work.

`'corollary'` (an simple consequence) An assertion is sometimes marked as a corollary to some other statement, if the proof is considered simple. This is often the case for important theorems that are simple to get from technical lemmata.

`'postulate'`, `'conjecture'` (an assertion without proof or counter-example) Conjectures are assertions, whose semantic value is not yet decided, but which the author considers likely to be true. In particular, there is no proof or counter-example (see section 3.2.4).

`'false-conjecture'` (an assertion with a counter-example) A conjecture that has proven to be false, i.e. it has a counter-example. Such assertions are often kept for illustration and historical purposes.

`'obligation'`, `'assumption'` (an assertion on which the proof of another depends) These kinds of assertions are convenient during the exploration of a mathematical theory. They can be used and proven later (or assumed as an axiom).

`'formula'` (if everything else fails) This type is the catch-all, if none of the others applies.

Since there can be more than one definition per symbol, OMDoc has the `alternative` element. Conceptually, an alternative definition or axiom

```
<assertion id="ida.c6s1p4.l1" type="lemma">
 <CMP> A semi-group has at most one unit.</CMP>
 <FMP>∀S.sgrp(S) → ∀x,y.unit(x,S) ∧ unit(y,S) → x = y</FMP>
</assertion>
```

Figure 3.13: An assertion about semigroups

is just a group of assertions that specify the equivalence of logical formulae. Of course, alternatives can only be added in a consistent way to a body of mathematical knowledge, if it is guaranteed that it is equivalent to the existing ones. Therefore, `alternative` has the attributes `entails` and `entailed-by`, that specify `assertion`s that state the necessary entailments. It is an integrity condition of OMDoc that any `alternative` element references at least one `definition` or `alternative` element that entails it and one that it is entailed by (more can be given for convenience). The `entails-thm`, and `entailed-by-thm` attributes specify the corresponding assertions. This way we can always reconstruct equivalence of all definitions for a given symbol.

Element	Attributes		D	Content
	Required	Optional	C	
symbol	id	kind, scope, style	+	CMP*, (commonname | type | selector)*
commonname		xml:lang	−	CMP content
type	system	id, for, style	−	CMP*, OMOBJ
axiom	id	generated-by, style	+	symbol*,CMP*,FMP*
definition	id, for	just-by, type, generated-by, style	+	CMP*, (FMP* | requation+ | OMOBJ)?, measure?, ordering?
requation		id, style	−	pattern, value
pattern			−	OMOBJ
value			−	OMOBJ
measure			−	OMOBJ
ordering			−	OMOBJ

Figure 3.14: Assertions, Examples, and Alternatives in a OMDoc

3.2.4 Mathematical Examples in OMDoc

In mathematical practice, examples play an equally great role as proofs, e.g. in concept formation as witnesses for definitions or as either supporting evidence, or as counter-examples for conjectures. Therefore, examples are given status as primary objects in OMDoc. Conceptually, we model an example $\mathcal{E}$ as a pair $(\mathcal{W}, \mathbf{A})$, where $\mathcal{W} = (\mathcal{W}_1, \ldots, \mathcal{W}_n)$ is an n-tuple of mathematical objects, $\mathbf{A}$ is an assertion. If $\mathcal{E}$ is an example for a mathematical concept given as an OMDoc symbol $\mathbf{S}$, then $\mathbf{A}$ must be of the form $\mathbf{S}(\mathcal{W}_1, \ldots, \mathcal{W}_n)$.

If $\mathcal{E}$ is an example for a conjecture $\mathbf{C}$, then we have to consider the situation more carefully. We assume that $\mathbf{C}$ is of the form $\mathcal{Q}\mathbf{D}$ for some formula $\mathbf{D}$, where $\mathcal{Q}$ is a sequence $\mathcal{Q}_1 W_1, \ldots, \mathcal{Q}_m W_m$ of $m \geq n = \#\mathcal{W}$ quantifications of using quantifiers $\mathcal{Q}_i$ like $\forall$ or $\exists$. Now let $\mathcal{Q}'$ be a subsequence of $m - n$ quantifiers of $\mathcal{Q}$ and $\mathbf{D}'$ be $\mathbf{D}$ only that all the W_{i_j} such that the $\mathcal{Q}_{i_j}$ are absent from $\mathcal{Q}'$ have been replaced by $\mathcal{W}_j$ for $1 \leq j \leq n$. If $\mathcal{E} = (\mathcal{W}, \mathbf{A})$ supports $\mathbf{C}$, then $\mathbf{A} = \mathcal{Q}'\mathbf{D}'$ and if $\mathcal{E}$ is a counter-example for $\mathbf{C}$, then $\mathbf{A} = \neg\mathcal{Q}'\mathbf{D}'$.

OMDoc specifies this intuition in an `example` element that contains a set of OpenMath objects (the witnesses), and has the attributes

`for` for what concept or assertion it is an example. This is a reference to a `definition` or `assertion` element.

`type` specifying the aspect, the value is one of the keywords `'for'` or `'against'`

`assertion` a reference to the assertion $\mathbf{A}$ mentioned above that formally states that the witnesses really form an example for the concept of assertion. In many cases even the statement of this is non-trivial and may require a proof.

In Figure 3.15 we show an example of the usage of an `example` element in OMDoc: We declare a symbol `string-struct` for the structure $\mathcal{W} := (A^*, \circ)$, where A^* is the set of words over an alphabet A and $\circ$ is word concatenation. Then we state that $\mathcal{W}$ is a monoid with the empty word as the neutral element in an `assertion` with `id string-struct-monoid`. Then `example` element with `id="mon.ex1"` in Figure 3.15 is an example for the concept of a monoid, since it encodes the pair $(\mathcal{W}, \mathbf{A})$ where $\mathcal{W}$ is encoded as the corresponding OMDoc symbol and $\mathbf{A}$ by reference to the assertion `string-struct-monoid` in the `assertion` attribute. Example `mon.ex2` uses the pair $(\mathcal{W}, \mathbf{A}')$, as a counter-example to the false conjecture

```
<symbol id="monoid"/>
<definition id="monoid-def" for="monoid">...</definition>
...
<symbol id="string-struct"/>
<definition id="sst-def" for="string-struct">...</definition>
...
<assertion id="string-struct-monoid" type="lemma">
 <CMP>(A^*,∘) is a monoid.</CMP>
 <FMP>mon(A^*,∘)</FMP>
</assertion>
...
<example id="mon.ex1" for="monoid" type="for"
        assertion="string-struct-monoid">
 <CMP>The set of strings with concatenation is a monoid.</CMP>
 <OMOBJ><OMS cd="strings" name="strings-struct"/></OMOBJ>
</example>

<example id="mon.ex2" for="monoids-are-groups" type="against"
        assertion="strings-isnt-group">
 <CMP>The set of strings with concatenation is not a group.</CMP>
 <OMOBJ><OMS cd="strings" name="strings-struct"/></OMOBJ>
</example>

<assertion id="monoid-are-groups" type="false-conjecture">
 <CMP>Monoids are groups</CMP>
 <FMP>∀ V.mon(V) → group(V)</FMP>
</assertion>
<assertion id="strings-isnt-group">...</assertion>
```

Figure 3.15: An OMDoc representation of a mathematical example

that all monoids are groups using the assertion `strings-isnt-group` for $\mathbf{A}'$.

3.2.5 Representing Proofs in OMDoc

Proofs form an essential part of mathematics and modern sciences. Conceptually they are a representation of uncontroversial evidence for the truth of an assertion.

The question of what exactly constitutes a proof has been controversially discussed. The clearest (and most radical) definition is given by theoretical logic, where a proof is a sequence, or tree, or directed acyclic graph (DAG) of applications of inference rules from a formally defined logical calculus, that meets a certain set of well-formedness conditions. There is a whole zoo of logical calculi that are optimized for various applications. They have in common that they are extremely explicit and verbose, and that the proofs even for simple theorems can become very large. The advantage of having formal and fully explicit proofs is that they can be very easily verified, even by simple computer programs.

In OMDoc, the notion of fully formal proofs is accommodated by the `proofobject` element. It contains an optional multilingual group of `CMP` elements which describe the formal proof as well as a proof object. This will normally be a complex λ-term encoded as an OpenMath object via the Curry/Howard/DeBruijn Isomorphism (see e.g. [Tho91] for an introduction). λ-terms are the most succinct representations of calculus-level proofs, since they only document the inference rules (which are encoded as OpenMath symbols in OMDoc). Since they are fully formal, they are very difficult to read and need specialized proof presentation systems for human consumption. In mathematical practice the notion of a proof is more flexi-

$$\dfrac{\dfrac{\dfrac{[A\wedge B]}{B}\wedge ER \qquad \dfrac{[A\wedge B]}{A}\wedge EL}{B\wedge A}\wedge I}{A\wedge B\Rightarrow B\wedge A}\Rightarrow I$$

```
<proofobject id="ac.p" for="and-comm">
 <OMOBJ>
  <OMBIND>
   <OMS cd="ND(FOL)" name="impliesI"/>
   <OMBVAR>
    <OMATTR>
     <OMATP>
      <OMS cd="openproof" name="type"/>
      A ∧ B
     </OMATP>
     <OMV name="X"/>
    </OMATTR>
   </OMBVAR>
   <OMA>
    <OMS cd="ND(FOL)" name="andI" >
    <OMA>
     <OMA>
      <OMS cd="ND(FOL)" name="andEr">
      <OMV name="X"/>
     </OMA>
     <OMA>
      <OMS cd="ND(FOL)" name="andEl">
      <OMV name="X"/>
     </OMA>
    </OMA>
   </OMA>
  </OMBIND>
 </OMOBJ>
</proofobject>
```

Figure 3.16: A Proof Object for the commutativity of conjunction

ble, and more geared for consumption by humans: any line of argumentation is considered as a proof, if it convinces its readers that it can be expanded to a formal proof in the sense given above. As the expansion process is extremely tedious, this option is very seldom really carried out explicitly in practice. Moreover, as proofs are geared towards communication among humans, they are given at vastly differing levels of abstraction. From a very informal proof idea to the initiated specialist of the field, who can fill in the details himself, down to a very detailed account for skeptics or novices. Note that such a proof will normally be still well above the formal level. Furthermore, proofs will normally be tailored to the specific characteristics of the audience, who may be specialists in one part of a proof while unfamiliar to the material in others. Typically such proofs have a sequence/tree/DAG-like structure, where the leaves are natural language sentences interspersed with mathematical formulae (often called "mathematical vernacular").

To reconcile these notions of "proof" and to provide a common markup system for them, OMDoc concentrates on the tree/DAG-like structure of proofs. It supports a proof format whose structural and formal elements are derived from hierarchical data structures developed for semi-automated theorem proving (satisfying the logical side), but which also allows natural language representations at every level (allowing for natural representation of mathematical vernacular at multiple levels of abstraction.) This proof representation (see [BCF+97] for a discussion and pointers) is a DAG of nodes which represent the proof steps. The proof steps contain a representation of the local claim and a justification by either a logical inference rule or higher-level evidence for the truth of the claim. This evidence can consist either of a proof method that can be used to prove the assertion, or by a separate subproof, that could be presented if the consumer was unconvinced. Conceptually, both possibilities are equivalent, since the method can be used to compute the subproof (called its expansion).

Expansions of nodes justified by method applications are computed, but the information about the method itself is not discarded in the process as in tactical theorem provers like Isabelle or NuPrL. Thus proof nodes may have justifications at multiple levels of abstraction in a hierarchical proof data structure. Note that the assertions in the nodes can be given as mathematical vernacular (in `CMP`s) or as logical formulae (in `FMP`s). This mixed representation enhances multi-modal proof presentation [Fie97], and the accumulation of proof information in one structure. Informal proofs can be formalized [Bau99]; formal proofs can be transformed to natural language [HF96]. The first is important, since it will be initially infeasible to totally formalize all mathematical proofs needed for the correctness management of the knowledge base. Moreover, the hierarchical format allows to integrate various other proof representations like proof scripts (ΩMEGA replay files, ISABELLE proof scripts,...), references to published proofs, resolution proofs, etc, to enhance the coverage.

Element	Attributes		D	Content
	Required	Optional	C	
proof	id, for, theory	style	+	symbol*, CMP*, (metacomment | derive | hypothesis)*, conclude
proofobject	id, for, theory	style	+	CMP*, OMOBJ
metacomment		id	–	CMP*
hypothesis	id, discharged-in	style	–	symbol*,CMP*,FMP*
derive	id	style	–	CMP*, FMP*, method?, premise*, (proof | proofobject)?
conclude	id	style	–	CMP*, method?, premise*, (proof | proofobject)?
method	xref		–	OMOBJ*
premise	xref		–	EMPTY

Figure 3.17: The OMDOC Proof Elements

Let us now come to the concrete markup scheme for proofs provided by OMDOC (see Figure 3.17 for an overview). Due to the complex hierarchical structure of proofs, we cannot directly utilize the tree-like structure provided by XML, but use cross-referencing (see the discussion in section 3.6). Proofs are specified by `proof` elements in OMDOC that have the attributes `id`, `for`, and `theory`. The `for` attribute points to the assertion that is justified by this proof (this can be an `assertion` element or a `derive` proof step, thereby making it possible to specify expansions of justifications and thus hierarchical proofs). Note that there can be more than one proof for a given assertion.

The content of a proof consists of a sequence of proof steps, whose DAG structure is given by cross-referencing. These proof steps are specified in four kinds of OMDOC elements:

`derive` elements specify normal proof steps that derive a new claim from already known ones, from assertions or axioms in the current theory, or from the assumptions of the assertion that is under consideration in the proof. We will explain it in detail below.

```
<derive id="2.1.2.proof.a.proof.D2.1">
  <CMP>By <OMOBJ><OMS cd="reals" name="A2"/></OMOBJ>
       we have z + (a + (−a)) = a + (−a)
  </CMP>
  <FMP>(z + a) + (−a) = z + (a + (−a))</FMP>
  <method xref="x-mbase:omega-base-calc#foralli*"/>
    <OMOBJ><OMV name="z"/></OMOBJ>
    <OMOBJ><OMV name="a"/></OMOBJ>
    <OMOBJ>−a</OMOBJ>
  </method>
  <premise xref="A2"/>
</derive>
```

Figure 3.18: A `derive` proof step

`hypothesis` elements allow to specify local assumptions, well-known from calculi like Gentzen's Natural Deduction calculus [Gen35]. They allow the hypothetical reasoning discipline needed for instance to specify proof by contradiction, by case analysis, or simply to show that A implies B, by assuming A and then deriving B from this local hypothesis. To specify the locality of the assumption, it has the required attribute `discharged-in` that points to a proof step which discharges this hypothesis. The hypothesis is inaccessible for inference outside the subproof delimited by the `hypothesis` and the step where it is discharged. The `hypothesis` element can contain a multilingual `CMP` group and a `FMP` for the formalization of the local assumption.

`conclude` This element is a variant of `derive` that does not contain a local claim, it is reserved for the last step in a proof, which states the conclusion of the assertion. This is advantageous, since it is error-prone to repeat the claim and in mathematical vernacular, the last step is often explicitly verbalized to mark the end of the proof.

`metacomment` OMDoc supplies this element to allow for intermediate text that does not have a logical correspondence to a proof step, but e.g. guides the reader of the proof. Examples for this are remarks by the proof author, e.g. an explanation why some other proof method will not work. This element has an optional `id` for cross-reference and a multilingual `CMP` group for the text.

Since we have covered the `hypothesis` and `metacomment` elements for proof steps and `conclusion` is only a trimmed-down version, we now need

```
<proof id="t1_p1" for="t1" theory="sets">
 <conclude id="t1_p1_c">
  <CMP> We prove the assertion by a case analysis.</CMP>
  <proof id="t1_p1_c_p" for="t1_p1_c" theory="sets">
   <derive id="l1">
    <CMP>If $a \in U$, then $a \in U \cup V$.</CMP>
    <FMP>
     <assumption id="l1_A"><CMP>$a \in U$.</CMP></assumption>
     <conclusion id="l1_C"><CMP>$a \in U \cup V$.</CMP></conclusion>
    </FMP>
    <method xref="x-mbase://sets#Method-1"/>
    <proof id="l1_p" for="l1" theory="sets">
     <conclude id="l1_p_d1">
      <CMP>$a \in U \cup V$ by definition of $\cup$.</CMP>
     </conclude>
    </proof>
   </derive>
   <derive id="l2">
    <CMP>If $a \in V$, then $a \in U \cup V$.</CMP>
    <FMP>
     <assumption id="l2_A"><CMP>$a \in V$.</CMP></assumption>
     <conclusion id="l2_C"><CMP>$a \in U \cup V$.</CMP></conclusion>
    </FMP>
    <method xref="x-mbase:sets#Method-2"/>
    <proof id="l2_p" for="l2" theory="sets">
     <conclude id="l2_p_d1">
      <CMP>$a \in U \cup V$ by definition of $\cup$.</CMP>
     </conclude>
    </proof>
   </derive>
   <conclude id="t1_p_c_c1">
    <CMP> We have considered both cases, so we have $a \in U \cup V$.
    </CMP>
   </conclude>
  </proof>
 </conclude>
</proof>
```

Figure 3.19: A OMDoc representation of a proof by cases.

to define the `derive` elements. They contain an informal (natural language) representation of the proof step in a multilingual `CMP` group and a specification of the step in the formal proof data structure. This is given by the following elements:

`FMP` This gives a formal representation of the claim made by this proof step (as we have seen above, this is essentially a Gentzen-style sequent). Local assumptions from the `FMP` should not be referenced to outside the derive step they were made in. Thus, the derive step serves as a grouping device for local assumptions. In Figure 3.19, the first derive step is used to show $a \in U \cup V$ from the local assumption $a \in U$, while the second one introduces the implication.

`method` is an element that specifies a proof method or inference rule that justifies the assertion made in the `FMP` element. It has an `xref` attribute that points to the OMDoc definition `id` of the inference rule or proof method.[3] A method may have children, which are `OMOBJ` elements, these act as parameters to the method, e.g. the repeated universal instantiation method in Figure 3.18, where the parameters are the terms to instantiate for the bound variables.

`premise` These are empty elements whose `xref` attribute is used to refer to the proof- or local assumption nodes that the `method` was applied to to yield this result. These attributes specify the DAG structure of the proof.

`proof` If a derive step is a logically (or even mathematically) complex step that can be expanded into sub-steps, then the embedded `proof` element can be used to specify the sub-derivation.

This embedded `proof` allows us to specify generic markup for the hierarchic structure of proofs. Note that the same effect as embedding the `proof` element into a `derive` or `conclude` step can be obtained by specifying the `proof` at top-level and using the `for` attribute to refer to the identity of the enclosing proof step (given by its `id` attribute).

3.2.6 Abstract Data Types

Most specification languages for mathematical theories support definition mechanisms for sets that are inductively generated by a set of constructors. OMDoc supports abstract data types as a convenient shorthand for sets of inductively defined objects and recursive functions on these.

[3] At the moment OMDoc does not provide markup for such objects, so that they should best be represented by `symbol`s with `definition` where the inference rule is explained in the `CMP`, and (if appropriate) the `FMP` holds the corresponding sequent. A good alternative is to encapsulate system-specific encodings of the inference rules in `private` or `code` elements and have the `xref` attribute point to these.

The adt element for abstract data types is a piece of special syntax for the concise statement of such sets that follows the model used in the emerging CASL (Common Abstract Specification Language [CoF98]) standard. There, abstract data types declare a set of sorts (inductively defined sets), constructors (the sorts contain exactly the objects constructed only by constructors), and selectors (partial inverses of the constructors) together with type/sort information for the latter two.

An abstract data type is called free, iff there are no identities between constructor terms, i.e. two objects represented by different constructor terms can never be equal. An example of a free abstract data type is the theory of natural numbers. It has a single sort Nat, two constructors zero and suc for the successor function, and the selector pred for the predecessor function. An example of an abstract data type that is not free is the theory of finite sets given by the constructors emptyset and insert, since the set $\{a\}$ can be obtained by inserting a into the empty set an arbitrary (positive) number of times. This kind of abstract data type is called generated, since it only contains elements that are expressible in the constructors. An abstract data type is called loose, if it contains elements besides the ones generated by the constructors.

Element	Attributes		D	Content
	Required	Optional	C	
adt	id	type, style	+	CMP*, commonname*, sortdef+
sortdef	id	kind, scope, style	−	commonname*, (constructor | insort)*, recognizer?
constructor	id	type, scope, style	+	commonname*, argument*
argument	sort		+	selector?
insort	for		−	
selector	id	type, scope, kind, total, style	−	commonname*
recognizer	id	type, scope, kind, style	−	commonname*

Figure 3.20: Abstract Data Types in OMDoc

In OMDoc, we use the adt element to specify abstract data types. It has a type attribute that can have the values 'free', 'generated', and 'loose' and contains one or more sortdef elements. For instance, we can express the theory of natural numbers by the adt element in Figure 3.21.

A sortdef element is a highly condensed piece of syntax that declares a sort (an inductively defined set, i.e. one that contains all objects that can be written by a set of constructors). A sortdef element contains a set of constructor and insort elements. The latter are empty elements which refer to a sort declared elsewhere in a sortdef with their for attribute. They specify that all the constructors that sort are also constructors for the one defined in the parent sortdef.

The constructor elements specify the symbols that can be used to construct the elements of its sort. Since a constructor is in general an n-ary function, a constructor element contains n argument children that give the argument sorts of this function. Note that n may be 0 and thus the constructor element may be empty. Sometimes it is convenient to specify the inverses of a constructor. For this OMDoc offers the possibility to add an empty selector element to an argument, its attribute total specifies whether this symbol is a total (value 'yes') or a partial (value 'no') function. Finally, a sortdef element can contain a recognizer child that specifies a symbol for a predicate that is true, iff its argument is of the respective sort.

Note that the sortdef, constructor, selector, and recognizer elements define symbols of the name specified by their id element in the containing theory. To govern the visibility, they carry the attribute scope (with values 'global' and 'local') and the attribute kind (with values 'type', 'sort', 'object'). Furthermore, they can have commonname children that specify their common names.

To fortify our intuition, let us come back to our example of an abstract data type for the natural numbers in Figure 3.21. The abstract data type nat-adt is free and defines two sorts pos-nats and nats for the (positive) natural numbers. The positive numbers (pos) are generated by the successor function (which is a constructor) on the natural numbers (all positive naturals are successors). On pos, the inverse pred of succ is total. The set nats of all natural numbers is defined to be the union of pos and the constructor zero. Note that this definition implies the five well-known Peano Axioms: the first two specify the constructors, the third and fourth exclude identities between constructor terms, while the induction axiom states that nats is generated by zero and succ. The document that contains the nat-adt could also contain the symbols and axioms defined implicitly in the adt element explicitly as symbol and axiom elements for reference. These would

```
<adt id="nat-adt" type="free">
 <metadata>
  <Title>Natural Number Theory</Title>
  <Description>The Peano Axiomatization of Natural Numbers</Description>
 </metadata>
 <sortdef id="pos-nats">
  <commonname>The set of positive natural numbers</commonname>
  <constructor id="succ">
   <commonname>The successor function</commonname>
   <argument sort="nats">
    <selector total="yes" id="pred">
     <commonname>The predecessor function</commonname>
    </selector>
   </argument>
   <recognizer id="positive" scope="global">
    <commonname>
     The recognizer predicate for positive natural numbers.
    </commonname>
   </recognizer>
  </constructor>
 </sortdef>
 <sortdef id="nats">
  <commonname>The set of natural numbers</commonname>
  <constructor id="zero">
   <commonname>The number zero</commonname>
  </constructor>
  <insort for="pos-nats"/>
 </sortdef>
</adt>
```

Figure 3.21: The Natural numbers using `adt` in OMDoc

then carry the `generated-by` attribute with value `nat-adt`.

3.3 Theories as Mathematical Contexts

It is a key observation that mathematical texts can only be understood with respect to a particular mathematical context. In current (non-OMDoc) documents the reader has to infer the context from the document. Since OMDoc is concerned with semantic markup, it makes the notion of mathematical context explicit in the familiar notion of a mathematical theory: OMDoc requires that all mathematical statements explicitly reference a mathematical theory which situates them into an explicit context.

Theories are specified by the `theory` element in OMDoc, which has a required id attribute and contains a commonname element that specifies short keywords or key phrases. After these any top-level OMDoc element can occur, except theory elements themselves: OMDoc1.1 does not allow theories to nest; moreover, theory elements can contain imports elements to specify inheritance (see section 3.3.1).

Element	Attributes		D	Content
	Required	Optional	C	
theory	id	style	+	commonname*, CMP*, (statement | inclusion, imports)*
imports	id, from	type, hiding, style	−	CMP*, morphism?
morphism		id, base, style	−	requation*
inclusion	for		−	
theory-inclusion	id, from, to, by	style	+	(morphism, decomposition?)
axiom-inclusion	id, from, to	style	+	morphism?, (path-just | obligation*)
obligation	induced-by, assertion		−	EMPTY
decompo-sition	links		−	EMPTY
path-just	local, globals		−	EMPTY

Figure 3.22: Complex Theories in OMDoc

We say that symbol, axiom, definition, and type elements are constitutive for a given theory, since changing this information will yield a different theory (with different mathematical properties, see the discussion in section 2.4). Therefore these theory-constitutive elements are not allowed as top-level statements, but must be contained as children in a theory element. In particular, a symbol element must occur in a theory environment, since both the theory id and the symbol id are needed to allow referring back to this symbol in an OMS element. For instance the symbol declaration in Figure 3.8 can be referenced as <OMS cd="elal" name="monoid"/>, if it occurs in a theory with id with value elal for elementary algebra.

The other mathematical statements defined in section 3.2 we call non-constitutive statements, since they can be derived from the material specified in a theory, which we will call their home theory. They are allowed to occur outside their home theory in OMDoc documents (e.g. as top-level elements), however, if they do, they must reference their home theory in a special `theory` attribute. The division of statements into constitutive and non-constitutive ones and the encapsulation of constitutive elements in `theory` elements add a certain measure of safety to the knowledge management aspect of OMDoc. Since Xml elements cannot straddle document borders, all constitutive parts of a theory must be contained in a single document; nothing can be added later (by other authors), since this would change the meaning of the theory on which other documents may depend on.

Theories also serve another purpose in OMDoc: In the examples we have already seen that OMDoc documents contain definitions of mathematical concepts, which need to be referred to using OpenMath symbols. In particular, documents describing theories even reference OpenMath symbols they define themselves. Therefore we adapt the OpenMath standard [CC98] that requires, that OpenMath symbols are defined by `CDDefinition` elements in OpenMath content dictionaries by allowing them to be declared by `symbol` elements in `theory` elements in OMDoc documents. Here the theory name specified in the `id` attribute of the `theory` element takes the place of the `CDname` defined in the content dictionary. Thus OMDoc serves as a drop-in replacement for OpenMath content dictionaries (see [Koh00b] for further discussion).

An alternative to this would have been to generate OpenMath content dictionaries from OMDoc theories. This can be done (both by the MBase system [FK00, KF00], and by specialized XslT style sheets), but seems an unnecessary complication.

3.3.1 Simple Inheritance

The main idea behind structured theories and specification is, that not all definitions and axioms need to be explicitly stated in a theory; they can be inherited from other theories, possibly transported by a signature morphism. The inheritance information is stated in the `imports` element in OMDoc.

The `imports` element has a `from` attribute, which specifies the theory which exports the formulae (the source theory). It has another attribute `type` that we will discuss in section 3.3.4, since here only the default value `'global'` is relevant.

In Figure 3.23 we have specified three algebraic theories that gradually build up a theory of groups importing symbols and axioms from earlier theories and adding their own content. The theory `semigroup` provides symbols for an operation `op` on a base set `set` and has the axioms for closure and associativity of `op`. The theory of monoids imports these without modification. Note that there is now a symbol `set` in the theory `monoid` that can be

```
<theory id="semigroup">
 <symbol id="set"/>
 <symbol id="op"/>
 <axiom id="closed"> ... </axiom>
 <axiom id="assoc"> ... </axiom>
</theory>

<theory id="monoid">
 <imports id="mis" from="semigroup"/>
 <symbol id="neut"/>
 <axiom id="left-unit">
  <CMP>
   <OMOBJ><OMS cd="monoid" name="neut"/></OMOBJ> is a left unit for
   <OMOBJ><OMS cd="monoid" name="op"/></OMOBJ>.
  </CMP>
  <FMP>∀x ∈ set.op(x, neut) = x</FMP>
 </axiom>
</theory>

<theory id="group">
 <imports id="gim" from="monoid"/>
 <symbol id="inv"/>
 <axiom id="left-inv">
  <CMP>
   For every object <OMOBJ><OMV name="X"/></OMOBJ> in
   <OMOBJ><OMS cd="group" name="set"/></OMOBJ> there is an object
   <OMOBJ><OMA><OMS cd="group" name="inv"/><OMV name="X"/></OMA></OMOBJ>
   which is an inverse wrt. <OMOBJ><OMS cd="group" name="op"/></OMOBJ>.
  </CMP>
 </axiom>
</theory>
```

Figure 3.23: A structured development of algebraic theories

referenced by `<OMS cd="monoid" name="set"/>` and shares all properties of the symbol `<OMS cd="semigroup" name="set"/>` by inheritance, but is not identical with it (and analogously one for `op`). In our example, they are used to state the `left-unit` axiom. The theory `monoid` then proceeds to add a symbol `neut` and an axiom that states that it acts as a left unit with respect to `set` and `op`. The theory `group` continues this process by adding a symbol `inv` for the function that gives inverses and an axiom that states its meaning.

The example in Figure 3.23 shows that with the notion of theory inheritance it is possible to re-use parts of theories and add structure to specifications. For instance it would be very simple to add a theory of Abelian semigroups by adding a commutativity axiom.

The set of axioms and definitions available for use in proofs in the importing theory consists of the ones directly specified as `axiom` and `definition` elements in the target theory itself (we speak of local axioms and definitions in this case) and the ones that are inherited from the source theory. Note that the inherited axioms and definitions can consist of the local ones in the source theory and the ones that are inherited there. As a consequence, all theorems, proofs, and proof methods of the source theory can be (after translation) be used in the importing theory.

Classical mathematics views theories as simply the set of theorems (propositions that can be proven from the axioms and definitions), abstracting from the structure we have given it in OMDoc. In this view, the source theory is included in the importing theory, therefore, we will call the relation specified by the `imports` element a theory inclusion.

3.3.2 Inheritance via Translations

Note that not in all situations it is sufficient to import symbols and axioms without modification. If we wanted to continue the algebraic hierarchy in Figure 3.23 with a theory of rings, then we would like to inherit the additive group structure from the theory `group` and the structure of a multiplicative monoid from the theory `monoid`. As this would lead to name clashes OMDoc allows theory inheritance via a translation function called morphism. To specify this function the `imports` element can have a child element `morphism`, which recursively specifies a function by a set of recursive equations using the `requation` element described above. As morphisms often contain common prefixes, the `morphism` element has an optional `base` attribute, which points to another morphism, which is taken to be the base of this morphism. The intended meaning is that the new morphism coincides as a function with the base morphism, wherever the specified `pattern` elements do not match, otherwise their corresponding `value` elements take precedence over those in the base morphism.

With the notion of theory inheritance via a morphism, we can e.g. define a theory of rings where rings are structures $(R, +, 0, -, *, 1)$ by importing from a group $(M, \circ, e, i)$ via the morphism $\{M \mapsto R, \circ \mapsto +, e \mapsto 0, i \mapsto -\}$ and from a monoid $(M, \circ, e)$ via $\{M \mapsto R^*, \circ \mapsto *, e \mapsto 1\}$, where R^* is R without 0 (as defined in the theory of monoids). Figure 3.24 gives the

```
<theory id="ring">
  <symbol id="ring.set"/><symbol id="ring.plus"/><symbol id="ring.zero"/>
  <symbol id="ring.setstar"/><symbol id="ring.times"/><symbol id="ring.one"/>
  <imports id="ring.add.import" from="group" type="global">
    <morphism>
      <requation>
        <pattern><OMS cd="group" name="set"/></pattern>
        <value><OMS cd="ring" name="ring.set"/></value>
      </requation>
      <requation>
        <pattern><OMS cd="group" name="op"/></pattern>
        <value><OMS cd="ring" name="ring.plus"/></value>
      </requation>
      <requation>
        <pattern><OMS cd="group" name="neut"/></pattern>
        <value><OMS cd="ring" name="ring.zero"/></value>
      </requation>
    </morphism>
  </imports>
  <imports id="ring.mult.import" from="monoid" type="global">
    <morphism>
      <requation>
        <pattern><OMS cd="monoid" name="set"/></pattern>
        <value><OMS cd="ring" name="ring.setstar"/></value>
      </requation>
      <requation>
        <pattern><OMS cd="monoid" name="op"/></pattern>
        <value><OMS cd="ring" name="ring.times"/></value>
      </requation>
      <requation>
        <pattern><OMS cd="monoid" name="neut"/></pattern>
        <value><OMS cd="ring" name="ring.one"/></value>
      </requation>
    </morphism>
  </imports>
  <definition id="ring.setstar.def" for="ring.setstar">
    <CMP> <OMOBJ><OMS cd="ring" name="ring.setstar"/></OMOBJ> is
      <OMOBJ><OMS cd="ring" name="ring.set"/></OMOBJ> without
      <OMOBJ><OMS cd="ring" name="ring.zero"/></OMOBJ>.
    </CMP>
  </definition>
  <axiom id="ring.distribution">
    <CMP><OMOBJ><OMS cd="monoid" name="plus"></OMOBJ> distributes over
      <OMOBJ><OMS cd="monoid" name="times"></OMOBJ>
    </CMP>
  </axiom>
</theory>
```

Figure 3.24: A theory of rings by inheritance via renaming

OMDoc representation of this exercise.

Finally, it is possible to hide symbols from the source theory by specifying them in the `hiding` attribute. The intended meaning is that the underlying signature mapping is defined (total) on all symbols in the source theory except on the hidden ones. This allows to define symbols that are local to a given theory, which helps achieve data protection. Of course, if we hide a sort symbol, we also have to hide all symbols using it (see [CoF98, MAH01] for details).

Even though the relation induced by the `imports` elements is not a simple subset relation any more, we will still keep the name theory inclusion for it. They have been called theory interpretations or theory morphisms elsewhere [Far93].

3.3.3 Statements about Theories

The theory inclusions discussed so far were definitional in nature; the inclusion relation among the sets of theorems was induced by the act of importing the relevant axioms and definitions from the source theory. We will call the importing theory the target theory. The benefit of having the theory inclusion is that all theorems, proofs, and proof methods of the source theory can be used (after translation) in the target theory. Obviously the transfer approach only depends on the theorem inclusion property, and we can extend its utility by augmenting the theory graph by more theory morphisms than just the definitional ones (see [FGT93] for a description of the IMPS theorem proving system that makes heavy use of this idea).

Following [Hut00] we structure a collection of theories as a graph – development graph there – where the nodes are theories and the links are theory inclusions (definitional and postulated ones).

There are two top-level elements for postulating relations among theories in OMDoc. The `theory-inclusion` element states that the source theory as included (modulo translation) in the target theory. It has the attributes `from` (it points to the source theory), `to` (this points to target theory). The children are a `CMP` group for descriptive text and a `morphism` child element as described above to define the translation function. The `theory-inclusion` element has a local variant, the `axiom-inclusion` element, that only states that the local `axioms` and `definitions` are theorems of the target theory.

Figure 3.25 shows a theory inclusion from the theory `group` defined in Figure 3.23 to itself. The morphism just maps each element of the base set to its inverse. A good application for this kind of theory morphism is to import claims for symmetric (with respect to the function `inv`, which serves as an involution) cases via this theory morphism to avoid explicitly having to prove them.

The `axiom-inclusion` has the same attributes as `theory-inclusion`.

```
<theory-inclusion id="ti1" from="group" to="group"
  by="inv-closed inv-assoc inv-left-unit inv-left-inv">
 <morphism>
  <requation>
   <pattern><OMA><OMS cd="group" name="inv"/><OMV name="X"/></OMA></pattern>
   <value><OMV name="X"/></value>
  </requation>
  <requation>
   <pattern><OMV name="X"/></pattern>
   <value><OMA><OMS cd="group" name="inv"/><OMV name="X"/></OMA></value>
  </requation>
 </morphism>
</theory-inclusion>

<assertion id="inv-closed">... </assertion>
<assertion id="inv-assoc">... </assertion>
<assertion id="inv-left-unit">... </assertion>
<assertion id="inv-left-inv">... </assertion>
```

Figure 3.25: A theory inclusion for groups

Furthermore, it can have children that justify that this relation holds, much like a `proof` justifies that an `assertion` element does about some property of mathematical objects. Concretely, a `axiom-inclusion` can hold a set of `obligation` children, or a single `path-just` child after the children allowed in `theory-inclusion`.

An `obligation` is an empty element that points to the proof obligation, i.e. an `assertion` hat states that the `axiom` or `definition` specified by the `induced-by` (translated by the morphism in the parent `axiom-inclusion`) is valid in the target theory. A `path-just` element justifies an `axiom-inclusion` by reference to other `axiom-` or `theory-inclusions`. The intuition is that the local axioms and definitions are included in a theory, if there is a chain of

$$S \xrightarrow{\sigma} T_1 \xrightarrow{\sigma_1} T_2 \xrightarrow{\sigma_2} \ldots T_n \xrightarrow{\sigma_n} T$$

such that the $S \xrightarrow{\sigma} T_1$ is an `axiom-inclusion` with morphism σ, the $T_i \xrightarrow{\sigma_i} T_{i+1}$ are `theory-inclusions` with morphism σ_i, and $\sigma_n \circ \cdots \circ \sigma_1 \circ \sigma$ is the morphism in the parent `axiom-inclusion`. We call this situation, where a theory T can be reached by an `axiom-inclusion` with a subsequent chain of `theory-inclusions` a local chain (with morphism $\sigma_n \circ \cdots \circ \sigma_1 \circ \sigma$). Local chains are encoded in the empty `path-just` element that has the attributes `local` (for the first `axiom-inclusion`) and the attribute `globals`, which contains a whitespace-separated list of pointers to `theory-inclusions`.

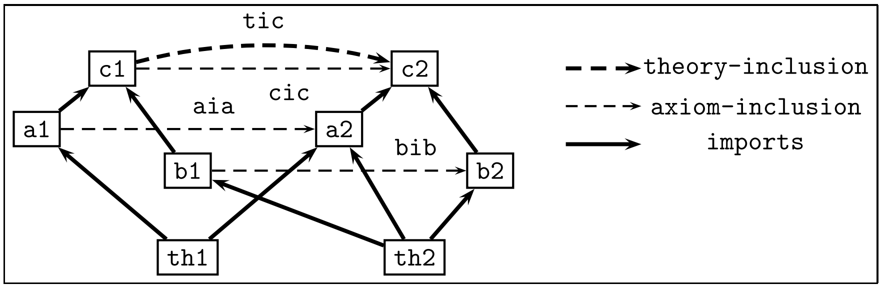
The relevance of the `axiom-inclusion` elements is that they can be used to justify `theory-inclusions`: A `theory-inclusion` $S \xrightarrow{\sigma} T$ is valid in a development graph, iff for any theory U that can reach S by a local chain with morphism θ there is an axiom-inclusion from U to T with morphism $\sigma \circ \theta$. This situation is encoded in the empty top-level decomposition element, which has the attributes for (which points to the theory-inclusion it justifies) and the attribute links, that contains a whitespace-separated list of pointers to axiom-inclusions or theory-inclusions.

In Figure 3.26 we have worked a simple example that shows a situation where all these elements are used. On the basis of theories th1 and th2, theory c1 is built up via theories a1 and b1. Similarly, theory c2 via a2 and b2, and theory-inclusion tic is postulated from c1 to c2. A decomposition justifies it by the axiom-inclusions cic, bic, aic from the theories a1, b1, c1 to c2, since these can reach c2 by local chains. Note that theories th1 and th2 reach c2 by local chains, but there the necessary axiom-inclusions are given by the theory-inclusion induced by the successive imports into c2. The axiom-inclusion aic is justified by the local chain which starts with the axiom-inclusion aia and then uses the theory inclusion induced by the import im1c2 in theory c2. The axiom-inclusion aia in turn is justified by the proof obligations for the axioms axa11 and axa12 in theory a1.

3.3.4 Parametric theories in OMDoc

Very often, the inheritence mechanisms presented so far do not suffice to model mathematical practice, since they do not allow for parameterization. In mathematics, the technique of studying certain aspects of complex mathematical objects in isolation by factoring out the remaining objects into generic parameters that can later be instantiated with concrete values is a key method for reducing the complexity inherent in the reasoning process. The technique also helps to modularize and reuse parts of specifications and theories. Before we discuss the parameterization issues in OMDoc let us look at a concrete example: a theory of lists of natural numbers.

We first specify a theory of lists that is generic in the elements, then we will instantiate this by applying this theory to the special element theory of natural numbers to obtain the intended theory of lists of natural numbers. The advantage of this approach is that we can now re-use the generic theory of lists to apply it to other element theories like that of sets of natural numbers to obtain a theory of lists of sets of natural numbers. In algebraic specification languages, we speak of **parametric theories**, i.e. the theory of lists has a formal parameter (in our example the set of elements) that can be instantiated later with concrete values to get a theory instances (in our example the theory of lists of natural numbers). We call this process theory

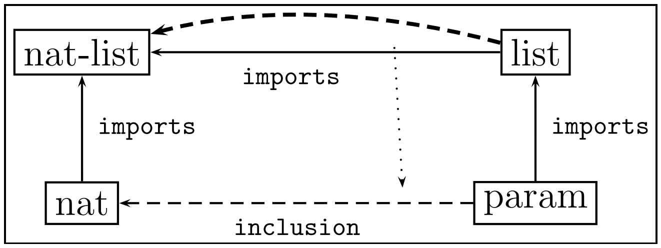



```
<theory id="th1">...</theory>     <theory id="th2">...</theory>

<theory id="a1">                  <theory id="b1">
 <imports id="ima1" from="th1"/>   <imports id="imb1" from="th2"/>
 <axiom id="axa11"> ... </axiom>    <axiom id="axb1"> ... </axiom>
 <axiom id="axa12"> ... </axiom>   </theory>
</theory>

<theory id="a2">                  <theory id="b2">
 <imports id="im1a2" from="th1"/>  <imports id="imb2"a from="th2"/>
 <imports id="im2a2" from="th2"/>
 <axiom id="axa2"> ... </axiom>    <axiom id="axb2"> ... </axiom>
</theory>                         </theory>

<theory id="c1">                  <theory id="c2">
 <imports id="im1c1" from="a1"/>  <imports id="im1c2"a from="a2"/>
 <imports id="im2c1" from="b1"/>  <imports id="im2c2"a from="b2"/>
 <axiom id="axc1"> ... </axiom>    <axiom id="axc1"> ... </axiom>
</theory>                          </theory>

<theory-inclusion id="tic" from="th1" to="th2"/>
<decomposition id="ti1d" for="ti1" links="aic bic cic"/>

<axiom-inclusion id="aic" from="a1" to="c2">
 <path-just local="aia" globals="im1c2"/>
</axiom-inclusion>

<axiom-inclusion id="bic" from="a1" to="c2">
 <path-just local="bib" globals="im2c2"/>
</axiom-inclusion>

<axiom-inclusion id="aia" from="a1" to="a2">
 <obligation induced-by="axa11" assertion="th-axa11"/>
 <obligation induced-by="axa12" assertion="th-axa12"/>
</axiom-inclusion>

<axiom-inclusion id="bib" from="b1" to="b2">
 <obligation induced-by="axb1" assertion="th-axb1"/>
</axiom-inclusion>

<axiom-inclusion id="cic" from="c1" to="c2">
 <obligation induced-by="axc1" assertion="th-axc1"/>
</axiom-inclusion>
```

Figure 3.26: A development graph with theory inclusion

actualization.

Parts of this process can be modeled in the OMDoc development graph




```
<theory id="param">
 <symbol id="elem" type="sort"/>
 <symbol id="ord"/>
 <axiom id="toset"><CMP>\(ord\) is a  total order on \(elem\).</CMP></axiom>
</theory>

<assertion id="ord-nat" theory="nat">
 <CMP>\(geq\) is a total order on \(nats\).<OMOBJ>.
 </CMP>
<assertion>

<theory id="list">
 <imports id="list.im" from="param"/>
 <symbol id="list-sort" type="sort"/>
 <symbol id="cons"/><symbol id="nil"/>
 <symbol id="ordered"/>
</theory>

<theory id="nat-list">
 <imports id="nat-list.im-nat" from="nat"/>
 <imports id="nat-list.im-elt" from="list" type="local">
  <morphism id="elem-nat">
   <requation>
    <pattern><OMOBJ><OMS cd="param" name="elem"/></OMOBJ></pattern>
    <value><OMOBJ><OMS cd="nat.thy" name="nats"/></OMOBJ></value>
   </requation>
  </morphism>
 </imports>
 <inclusion via="elem-nat-incl"/>
</theory>

<axiom-inclusion id="elem-nat-incl" from="nat" to="param">
 <morphism id="elem-nat-incl-morph" base="elem-nat"/>
 <obligation induced-by="toset" assertion="ord-nat"/>
</axiom-inclusion>

<theory-inclusion id="nat-natlist-incl" from="list" to="nat-list">
 <morphism id="nat-natlist-incl-morph" base="elem-nat"/>
</theory-inclusion>
<decomposition id="dec" for="nat-natlist-incl" links="elem-nat-incl"/>
```

Figure 3.27: A Structured Specification of Lists

model of theory inheritance by constructing dedicated parameter theories. Consider the situation in Figure 3.27, where we have theories `nat` of natural numbers (for instance one that contains the abstract data type in Figure 3.21) and a generic theory `list` of lists that imports its elements from a generic parameter theory `param`. There the theory `nat-list` of lists of natural numbers is built up by importing from the theories `nat` and `list` making the `nat` the actual parameter theory in the process. Note that the attribute `type` of the `imports` element `nat-list.im-elt` is set to `'local'`, since we do not want to import the local axioms of the theory `list` and not the whole theory `list` (which would include the axioms from `param`). The effect of the actualization comes from the morphism `elem-nat` in the import of `List` that renames the symbol `elem` (from theory `param`) with `nats` (from theory `nat`).

Note that this naive encoding of actualization does not always lead to the expected result that there is a theory inclusion from the theory `list` to `nat-list`. Say we are actually trying to specify a theory of ordered lists, then we need an ordering relation on the set `elem` of elements. We introduce a symbol `ord` and the necessary axiom in the theory `param` to make `elem` a totally ordered set. As `param` is imported into `list`, these are available to specify ordered lists in the theory `list`. Now, if we do the actualization from `List` to `nat-list`, we have to ensure that the parameter theory `nat` also has a suitable ordering function. This can be specified using the OMDoc `inclusion` element. `inclusion` is an empty element whose `via` attribute points to an axiom inclusion from the generic parameter theory to the actual parameter theory whose morphism extends the import morphism of the parameter theory. In our examples we can see that the `axiom-inclusion` specified in the `inclusion` element is sufficient to guarantee the theory inclusion from `nat` to `nat-list` that states the correctness of the parameter actualization. For more details see [Hut00].

Note that this mechanism for parametric theories only works in situations with an a-priori finite number of possible theory instances, since these have to be explicitly generated in advance. However theorem provers like the Pvs system [ORS92] also allow to quantify over variables that are later used to instantiate theory parameters; in this case the number of theory instances is potentially infinite, and cannot be directly be represented in OMDoc. Unfortunately this problem cannot simply be fixed by adding additional representation concepts to OMDoc, since it breaks a fundamental assumption in OpenMath, namely that theories can always explicitly be represented (as content dictionaries), and that this can be done ahead of using them. Thus extending MBase/OMDoc to this form of mathematical practice will reconciling even something as fundamental as the OpenMath standard with mathematical practice. Note that the concept of parametric theories cannot easily be dismissed as representational aberrations; the work on so-called functors in the Theorema project `http://www.theorema.org` views parametric theories as the principal building blocks and successfully uses this higher-order structure to guide theorem proving.

3.4 Auxiliary Elements

Up to now, we have been mainly concerned with providing elements for marking up the inherent structure of mathematical knowledge in mathematical statements and theories. We have not bothered yet about representing the structure of mathematical documents themselves (as structured text entities), or the information that is necessary transforming the mathematics into formats suitable for communicating with humans or mathematical software systems. We will introduce the necessary infrastructure in this section.

In 3.4.1 we will present OMDoc elements for representing the text structure of mathematical documents as structured text entities. This makes it possible to transform legacy documents into OMDoc form without losing information: the paragraphs of the input document are classified into mathematical statements wherever possible and the remaining are represented as `omtext`. The text structure of the input document (paragraphs, sections, and chapters) is represented by `omgroup` elements of suitable types; some of these may also be reflected in the theory structure organizing the knowledge.

In 3.4.2 we introduce an infrastructure for interfacing OMDoc documents with the Internet in general and mathematical software systems in particular. The application of this is that we can generate representations from OMDoc documents where formulae, statements or even theories that are active components that can directly be manipulated by the user or mathematical software systems.

Finally, we present a present a limited infrastructure for mathematical exercises in section 3.4.4. This allows to use OMDoc as a basis for mathematical education and assessment. Note that the infrastructure introduced here is relatively little developed, and born out of the immediate need of OMDoc projects. We envision that in the future we will use specialized Xml vocabularies like the IMS standard for questions and exercises [SSBL01].

3.4.1 Preservation of Text Structure

Like other documents, mathematical ones are often divided into units like chapters, sections, and paragraphs by tags and nesting information. OMDoc makes these document relations explicit with specialized `omgroup` elements. These have an attribute `type` that can take values including `'itemize'`, `'sequence'`, `'enumerate'` with the obvious meanings as text groups. The `type` attribute can be used to specify other grouping devices, such as `'dataset'` for table and `'theory-collection'`, which we will discuss elsewhere in this document (consult the attribute table on in appendix D).

We consider the `omdoc` element as an implicit `omgroup`, in order to allow plugging together different different OMDoc documents as `omgroups`. As a consequence, the `omdoc` element has also has an `type` attribute that can take the same values as that for the `omgroup` element. `omgroup` elements can appear at top-level, and can contain any top-level elements. As the document structure need not be a tree in hypertext documents, `omgroup` elements also allow `ref` elements whose `xref` attribute can be used to reference OMDoc elements defined elsewhere. The `type` attribute can be used to describe the reference type. Currently OMDoc supports two values: `'include'` (the default) for in-text replacement and `'cite'` for a proper reference. The first kind of reference requires the OMDoc application to process the document as if the `ref` element were replaced with the OMDoc fragment specified in the `xref`. The processing of the second type is application specific it is recommended to generated an appropriate label and (optionally) supply a hyper-reference. There may be more supported values for `type` in time.

This structuring approach allows to "flatten" the tree structure in a document into a list of leaves and relation declarations (see Figure 3.28 for an example). It also makes it possible to have more than one "view" on a document using `omgroup` structures that reference to a shared set of OMDoc elements as leaves.

While the OMDoc approach to specifying document structure is a much more flexible (database-like) approach to representing structured documents[4], than the tree model, it puts a much heavier load on a system for

[4]The simple tree model is sufficient for simple markup of existing mathematical texts and to replay them verbatim in a browser, but is insufficient e.g. for generating individualized presentations at multiple levels of abstractions from the representation. The OMDoc text model – if taken to its extreme – allows to specify the respective role and contributions of smaller text units, even down to the sub-sentence level, and make the structure of mathematical texts "machine understandable". Thus, an advanced presentation engine like the ActiveMath system [SBC+00] can – for instance – extract document fragments based on the preferences of the respective user.

```
<omgroup id="text"
         type="sequence">
 <omtext id="t1">T1</omtext>
 <omgroup id="enum"
          type="enumeration">
  <omtext id="t2">T2</omtext>
  <omtext id="t3">T3</omtext>
 </omgroup>
 <omtext id="t4">T4</omtext>
</omgroup>
```

⟷

```
<omtext id="t1">T1</omtext>
<omtext id="t2">T2</omtext>
<omtext id="t3">T3</omtext>
<omtext id="t4">T4</omtext>

<omgroup id="text" type="sequence">
  <ref xref="t1"/>
  <ref xref="enum"/>
  <ref xref="t4"/>
</omgroup>

<omgroup id="enum" type="enumeration">
  <ref xref="t2"/>
  <ref xref="t3"/>
</omgroup>
```

Figure 3.28: Flattening a tree structure

presenting the text to humans. In essence the presentation system must be able to recover the left representation from the right one in Figure 3.28. Generally, any OMDoc element defines a fragment of the OMDoc it is contained in: everything that this element contains and (recursively) those elements that are reached from it by following the cross-references. In particular, the text fragment corresponding to the element with (`id="text"`) in the right OMDoc of Figure 3.28 is just the one on the right.

Element	Attributes		D	`Content`
	Required	Optional	C	
`omgroup`	`id`	`type`, `style`	+	OMDoc element
`ref`		`xref`, `type`	–	

Figure 3.29: OMDoc elements for specifying document structure.

The `omgroup` element has other uses in OMDoc, it can be used specify data sets, a content-oriented generalization of tables to arbitrary dimensions. OMDoc views tables as instances of sets of data structured into k dimensions and reserves the value `'dataset'` for this purpose. Conventional tables are just the two-dimensional special case.

Let us analyze this approach using the example in Figure 3.30, which is a $k = 2$-dimensional special case. The $n = 2 \times m = 3$-table in the top is modeled as the outer `omgroup` element, the value `'labeled-dataset'` signifies that this is a table where the axes are labeled. The name of the table can be given in the `metadata/Title`. The outer `omgroup` element organizes

name	a1	a2	a3
b1	e11	e12	e13
b2	e21	e22	e23

```
<omgroup id="example-table" type="labeled-dataset">
 <metadata><Title>name</Title></metadata>

 <omgroup id="f1" type="dataset">
  <omgroup type="dataset"><omtext><CMP>a1</CMP></omtext></omgroup>
  <omgroup type="dataset"><omtext><CMP>a2</CMP></omtext></omgroup>
  <omgroup type="dataset"><omtext><CMP>a3</CMP></omtext></omgroup>
 </omgroup>

 <omgroup id="f2" type="dataset">
  <omgroup type="dataset"><omtext><CMP>b1</CMP></omtext></omgroup>
  <omgroup type="dataset"><omtext><CMP>b2</CMP></omtext></omgroup>
 </omgroup>

 <omgroup id="data" type="dataset">
  <omgroup id="dc1" type="dataset">
   <omgroup type="dataset"><omtext><CMP>e11</CMP></omtext></omgroup>
   <omgroup type="dataset"><omtext><CMP>e12</CMP></omtext></omgroup>
   <omgroup type="dataset"><omtext><CMP>e13</CMP></omtext></omgroup>
  </omgroup>

  <omgroup id="dc2" type="dataset">
   <omgroup type="dataset"><omtext><CMP>e21</CMP></omtext></omgroup>
   <omgroup type="dataset"><omtext><CMP>e22</CMP></omtext></omgroup>
   <omgroup type="dataset"><omtext><CMP>e23</CMP></omtext></omgroup>
  </omgroup>
 </omgroup>
</omgroup>
```

Figure 3.30: Specifying Tables with `<omgroup type="dataset">`

the table information into two parts. The first part (the first k elements) contains the label information and the last element the data proper. All of these elements are `omgroup`s of type `'dataset'`, which signifies that they do not contain label information. The nesting depth of the `omgroup` elements corresponds to the dimension of the data sets involved. The k label elements are $k-1 = 1$-dimensional data sets in our example, while the body is $k = 2$-dimensional, and is organized into $n = 2$ rows, which again are `omgroup`s of `type 'dataset'` (one-dimensional tables). The generalization for higher dimensions is obvious (e.g. for a $k = 3$-dimensional $l \times n \times m$-array we would have an `omgroup` that has a $n \times m$-table of labels like the one in Figure 3.30, followed k such `omgroup` elements for the k levels). Note that the final table entries are modeled as 0-dimensional data sets, and thus contain a seemingly spurious `omgroup`. This makes the approach more flexible (we can have structured text objects in table entries) and more uniform (and thus entails better substitution properties).

This content-based approach makes the components of tables more explicit than presentation-based ones. In particular, we can reference individual parts like rows and columns by their `id` attributes for later reference. We can also add metadata to sub-structures like labeling of faces of the tables, e.g. the face `f2` could have a `metadata` element with a `Description` that says that the data are temperatures in degree Celsius.

Of course, the `'dataset'` `omgroup`s only specify the content of the tables, the concrete appearance in the output format generated will be determined by a presentation component.

3.4.2 Non-Xml Data and Program Code in OMDoc

The OMDoc elements we have presented standardize the representation of general mathematical knowledge. Since we have taken care to allow the formal representation of mathematical objects at every level, this is a representational infrastructure that is sufficient as an output and library format for mathematical software systems like computer algebra systems, theorem provers, or theory development systems. In particular, having a standardized output and and library format will enhance system interoperability, and allows to build and deploy general storage and library management systems like MBase (see 4.3).

However, most mathematical software systems need to store and communicate data, which is system-specific, but may be relevant to more than user. Examples of this are pieces of program code, like tactics or proof search heuristic of tactical theorem provers or linguistic data of proof presentation systems. Only if these data can be integrated into OMDoc, will it become a full storage and communication format for mathematical software systems. One characteristic of such system-specific data is that it is often not in Xml syntax, or its format is not fixed enough to warrant for a general Xml encoding.

For this kind of data, OMDoc provides the `private` and `code` elements. Their attributes contain metadata information identifying system requirements and relations to other OMDoc elements. We will first describe the shared attributes and then describe the elements themselves.

`id` (required) for identification.

`theory` this optional attribute allows the specification of the mathematical theory (see section 3.3) that the data is associated with.

Element	Attributes		D	Content
	Required	Optional	C	
omlet		id, argstr, type, function, action, data, width, height, style	+	CMP content
private		id, for, theory, pto, pto-version, requires, type, replaces, style	+	CMP*, data+
code	id, theory	id, for, theory, pto, pto-version, requires, type, classid, codebase, style	+	CMP*, input?, output?, effect?, data+
input			−	CMP*
output			−	CMP*
effect			−	CMP*
data		format, href, size	−	<![CDATA[...]]>

Figure 3.31: The OMDoc Auxiliary Elements for non-XML Data

`for` which allows to attach the data to some other OMDoc element. Attaching `private` elements to OMDoc elements is the main mechanism for system-specific extension of OMDoc.

`pto` is a whitespace-separated list of token names which specifies the set of systems to which the data are private. The intention of this field is that a `private` element is visible to all systems, but should only manipulated by a system that are mentioned here.

`pto-version` is a whitespace-separated list of tokens for version numbers; This only makes sense, if the value of the corresponding `pto` is a singleton. Specifying this may be necessary, if the data or even their format change with versions.

`type` the type of the data, the meaning of these fields is determined by the system itself.

`requires` specifies the identifiers of the elements that the data depend upon, which will often be `code` elements. This allows to factor private data into smaller parts, allowing more flexible data storage and retrieval. This is especially interesting for program code or private data that relies on program code. This can broken up into procedures and the call-hierarchy can be encoded in `requires` attributes. Based on this information, a storage application based on OMDoc can then always communicate a minimal complete code set to the requesting application.

The `private` element is intended for system-specific data that is not program code. It contains a `metadata` element and a set of `data` elements that contain or reference the actual data. The `private` element adds the `replaces` attribute to those described above. This specifies the identifiers (given in the `id` attribute) of OMDoc elements that are subsumed by the information in the private element. A special case is the empty private element (empty `data` element), that can be used to specify that certain OMDoc elements are irrelevant to a given application.

The `data` element contains the data in a `CDATA` section. It has the attribute `format` to specify the format the data are in, e.g. `image/jpeg` or `image/gif` for image data, `binary` for system-specific binary data, `text/plain` for text data, etc. It is good practice to use the MIME types for this purpose, wherever applicable. In a `private` or `code` element, the `data` elements must differ in their `format` attribute. If the content of this field is too large to store directly in the OMDoc or changes often, then it can be substituted by a link, specified in the `href` attribute. The optional `size` attribute can be used to specify the size of the outside resource.

The `code` element is used for embedding pieces of program code into an OMDoc document. This element has the attributes described above plus attributes `codebase` and `classid`, if it contains Java code. It contains the documentation elements `input`, `output`, and `effect` that specify the behavior of the procedure defined by the code fragment. The `input` element describes the structure and scope of the input arguments, `output` the outputs produced on these elements, and `effect` any side effects the procedure may have. They contain a multilingual `CMP` elements with an optional `FMP` for a formal description. The latter may be used for program verification purposes. If any of these elements are missing it means that we may not make any assumptions about them, not that there are no inputs, outputs or effects. For instance, to specify that a procedure has no side-effects we need to specify something like `<effect><CMP>None.</CMP></effect>`. These elements are followed by a set of `data` elements that contain or cross-reference the program code itself. Figure 3.33 shows an example of a `code` element used to store Java code for an applet.

3.4.3 Applets in OMDoc

Web-based text markup formats like HTML have a concept of an applet, i.e. programs that can in some way executed in the browser during document manipulation. This one of the primary ways used to enliven parts of the document.

```
<CMP>The missing parts are
 <omlet type="link" argstr="file://missing.html">here (html)</omlet>.
  For the mathematical background see
 <omlet type="preslink" argstr="file://back.omdoc">this</omlet>.
</CMP>
```

Figure 3.32: Some hyperlinks an `omlet`.

In OMDoc, we use the `omlet` element for applets and generalizes the applet concept in two ways: The computational engine is not restricted to plug-ins of the browser and the program code can be included in the OMDoc document, making document-centered computation easier to manage.

The simplest application of `omlet` elements is the specification of hyperlinks. Figure 3.32 shows two. The first one is a very simple hyperlink to a HTML file, the second one is a hyperlink to an OMDoc document, that will have to be translated into the current presentation format before it can be viewed by the user. Since hyperlinks are so important in mathematical documents, we reserve the keywords `'link'` and `'preslink'` for hyperlinks. Note that even for such a simple case as hyperlinks, the differences between hyperlinks and applets are blurred, therefore OMDoc only provides the `omlet` element for both.

```
<code id="callMint" codebase="org.riaca.cas">
 <CMP>
  The multiple integrator applet. It puts up a user interface
  queries the user for a function, which it then integrates
  by calling one of several computer algebra systems.
 </CMP>
 <input>None, the applet handles input itself.</input>
 <output>The result of the integration.</output>
 <effect>None.</effect>
 <data format="java">
  <![CDATA[...  the callMint code goes here ...]]>
 </data>
</code>

<omtext id="monp_1">
 <CMP> Let's <omlet type="js" function="callMint" action="execute">
     Integrate</omlet>!
 </CMP>
</omtext>
```

Figure 3.33: An `omlet` that calls a Java applet.

Like the HTML `applet` tag, the `omlet` element can be used to wrap any (set of) well-formed element. It has the following attributes.

`type` This specifies the computation engine that should execute the code. Depending on the application, this can be a programming language, such as `javascript` (`'js'`) or Oz, or a process that is running, e.g. a theorem prover.

`function` The code that should be executed by the omlet is specified in the `function` attribute. This points to an OMDoc code element that is somehow accessible (e.g. in the same OMDoc). This indirection allows us to reuse the machinery for storing code in OMDocs. For a simple example see Figure 3.33.

`argstr` This optional attribute allows to specify an argument string for the function called by the applet, so that the program in the can be kept general. A call to the $\mathcal{L}\Omega\mathcal{UI}$ interface, would for example have the form in Figure 3.34. Here, the code in the `code` element `sendtoloui` (which we have not shown) would be Java code that simply sends the value of the `argstr` to $\mathcal{L}\Omega\mathcal{UI}$'s remote control port.

`width/height` gives the screen height and width of the applet.

The expected behavior of the omlet can be implemented in the XSLT style sheet, that in the case of e.g. translation to MOZILLA will put the callMint code directly into the generated html.

```
<CMP> Let's prove it
  <omlet id="bla" type="java" function="sendtotp"
         argstr="load(problem='monoid_uniq')">
    interactively
  </omlet>.
</CMP>
```

Figure 3.34: An omlet for connecting to a theorem prover.

3.4.4 Exercises

Exercises are vital parts of mathematical textbooks. In OMDOC, we use the exercise element for representing them. The question statement is represented in the multilingual CMP group followed by an optional FMP element and an optional hint element that contains a hint in a CMP/FMP group.

Element	Attributes		D	Content
	Required	Optional	C	
exercise	id	style, for	+	symbol*,CMP*,FMP*, hint?, (solution* | mc*)
hint		id, style	+	symbol*,CMP*,FMP*
solution		id, for, style	+	symbol*,CMP*, (FMP* | proof | proofobject)
mc		id, style	–	symbol*, choice, hint?, answer
choice		id, style	+	symbol*,CMP*,FMP*
answer	verdict	id, style	+	symbol*,CMP*,FMP*

Figure 3.35: The OMDOC Auxiliary Elements for Exercises

The next element in an exercise is either a (set of) possible solutions, or a multiple-choice block. The first is represented in a solution element with a a CMP/FMP group followed by an optional proof or proofobject. A special case of this is the case, where the question contains an assertion whose proof is not displayed and left to the reader. In this case, the solution contains a proof. Multiple-choice exercises (see Figure 3.36) are represented by a list of mc elements. These represent a single choice in a choice element together with the answer in the `answer` element. The `verdict` of the `answer` element attribute specifies the truth of the answer, it can have the values `'true'` or `'false'`. The `choice` and `answer` elements contain CMP/FMP groups.

```
<exercise for="ida.c6s1p4.l1" id="ida.c6s1p4.mc1">
 <CMP>What is the unit element of the semi-group Q
  with operation a * b  =  3ab?
 </CMP>
 <mc><choice><FMP><OMOBJ><OMI>1</OMI></OMOBJ></FMP></choice>
     <answer verdict="false"><CMP>No, 1 * 1 = 3 and not 1</CMP></answer>
 </mc>
 <mc><choice><CMP>1/3</CMP></choice>
     <answer verdict="true"></answer>
 </mc>
 <mc><choice><CMP>It has no unit.</CMP></choice>
     <answer verdict="false"><CMP>No, try another answer</CMP></answer>
 </mc>
</exercise>
```

Figure 3.36: An Exercise

3.5 Adding Presentation Information to OMDoc

As we have seen, OMDoc is concerned mainly with the content and structure of mathematical documents, and offers a complex infrastructure for dealing with that. However, mathematical texts often carry typographic conventions that cannot be determined by general principles alone. Moreover, non-standard presentations of fragments of mathematical texts sometime carry meanings that do not correspond to the mathematical content or structure proper. In order to accomodate this, OMDoc provides a limited functionality for embedding style information into the document.

The normal (but of course not the only) way to generate presentation from XML documents is to use XSLT style sheets (see section 4.1 for other applications). XSLT [Dea99] is a general transformation language for XML. XSLT programs (often called style sheets) consist of a set of so-called templates (rules for the transformation of certain nodes in the XML tree). These templates are recursively applied to the input tree to produce the desired output.

The general approach is not to provide general-purpose presentational primitives that can be sprinkled over the document, since that would distract the author from the mathematical content, but to support the specification of general style information for OMDoc elements and mathematical symbols in separate elements.

In the case of a single OMDoc document it is possible to write a specialized style sheet that transforms the content-oriented markup used in the document into mathematical notation. However, if we have to deal with a large collection of OMDoc representations, then we can either write a specialized style sheet for each document (this is clearly infeasible to do by hand), or we can develop a style sheet for the whole collection (such style sheets tend to get large and unmanageable).

OMDoc supports variants of both approaches, it allows to generate specialized style sheets that are tailored to the presentation of (collections of) OMDoc documents. The mechanism will be discussed in section 4.1, here we only concern ourselves with the OMDoc primitives for representing the necessary data. In the next subsection, we will address the specification of style information for OMDoc elements by `omstyle` elements, and then the question of specification of notation of mathematical symbols in `presentation` elements.

3.5.1 Specifying Style Information for OMDoc Elements

OMDoc provides the `omstyle` elements for specifying style information for OMDoc elements. An `omstyle` element has the attributes has the attributes

`element` this required attribute specifies the OMDoc element this style information should be applied to. Note that the value of this attribute must be the full qualified name (i.e. including the) of the element.

`for` this optional attribute allows to further restrict the OMDoc element to a single instance.

`xref` This optional attribute can be used to refer to another existing `omstyle` element (in another document), sometimes avoiding double specification.

`style` This optional attribute is is an additional parameter that controls the output style. This allows to specify different notational conventions for symbols. This attribute corresponds to the optional `style` attributes in those OMDoc elements that have `id` attributes. They can be used to specify the style intended by the document author and help choose a `presentation` element.

Note that the choice of notational style is not a content-carrying feature, and should not be depended on, indeed the value of the `stlye` need not be respected by output routines, but can be overwritten.

The information specified in the body of this element is then used to generate XSLT templates that can be used in the style sheets. This information is either given directly as the bodies of XSLT templates in the `xslt` element, or in a `style` element using a small subset of XSLT internalized into OMDoc. This second language is used if the full power of XSLT is not needed, and has the advantage that it can be transformed into the input of other formatting engines. The `xslt` and `style` elements share the following attributes

`format` this required attribute specifies the output format. Its value is a string of format specifiers divided by the `|` character. We use the specifiers `'TeX'` for TEX and LATEX, `'pmml'` for presentation MathMl, `'cmml'` for content MathMl, `'html'` for HtML, `'mathematica'` for Mathematica notebooks. Finally, there is the pseudo format-specifier `'default'`, which will be taken, if no other format is defined. Note that case matters in these specifiers, so `TeX` is not the same as `tex`, furthermore, `'default'` is not a regular format specifier, so it cannot appear in the disjunctions.

See `http://www.mathweb.org/omdoc/xsl.html` for other available formats. Similarly, the English language serves as a default language.

`xml:lang` this specifies the language for which this notation is used. In contrast to the other uses of `xml:lang` does not have a default value `en`. If the attribute is not present, this means that this element is not language-specific.

`requires` This attribute points to a `code` element that contains a code fragment that is needed to be included for the presentation engine. For instance, a `use` element for the format LATEX may contain macro calls that need to be defined. Their definitions would need to be included in the output document by the presentation style sheet before they can be used.

Figure 3.37 shows very simple example, where a `with` element is used to mark a text passage as "important". This style attribute is then picked up in the `omstyle` element to prompt special treatment in the output. Note that here the attributes of the `use` element are used to specify bracketings, the presentation specified in the attributes are placed where the Xml tags `<with>` and `</with>` are.

```
<CMP>
  I want to mark <with id="w1" style="important">this important
  text</with> as special.<with style="linebreak"/>
  I can also refer to
  <with style="link"><OMOBJ><OMSTR>missing.html</OMSTR></OMOBJ>
   here</with>, if something is missing.
</CMP>

<omstyle element="omdoc:with" style="important">
  <style format='html|pmml'><element name="em"><recurse/></element></style>
  <xslt format='TeX'><[CDATA[{\em<xsl:apply-templates/>}]]></xslt>
</omstyle>

<omstyle element="omdoc:with" style="linebreak">
  <style format='html|pmml'><element name="br"/></style>
  <style format='TeX'><text>\par\noindent</text></style>
</omstyle>

<omstyle element="omdoc:with" style="link">
 <style format="html|pmml">
  <element name="a">
   <attribute name="href">
    <value-of select="om:OMOBJ/om:OMSTR"/>
   </attribute>
   <recurse select="*[not(om:OMOBJ)]/>
  </element>
 </style>
</omstyle>
```

Figure 3.37: Specifying Style information with the `with` Element.

Let us now look at the sub-language used in `style` elements. We can see in the second `omstyle` element that the content of the `xslt` element are XSLT fragments. They have to be either enclosed in a CDATA section, of escaped. Note that when referring to OMDoc elements, the XSLT must use the full qualified name (i.e. including the) of the elements for the presentation to work.

In the first `style` in the `omstyle` for `linebreak`, we see that `element` element can be used to insert an XML element into the output; in this case it is the empty HTML element `<br/>`. In the second `style` child the `text` element (it does not have attributes) allows to add arbitrary text into the output (in this case some TeX macros). In the second `omstyle` element, we see that the `element` may be non-empty, in this case, it contains the element `recurse`, which corresponds to the directive to contain presentation generation recursively over the children of the element specified in the dominating omstyle element (in this case again a with element). The effect of this is that the content of the element <with style="important"> is encased in the HTML <em> element. Generally, the recurse element is empty, and can have the attribute select, which contains an XPATH [Cla99] expression specifying a set of OMDOC elements the presentation should continue on recursively. If this attribute is missing, presentation continues on the children as in the example above.

The element element has a required attribute name, which contains the element name, attributes can be specified by the attribute element: any attribute element adds an attribute-value pair of the form name="value" to the output element specified by the enclosing element element, where name is the value of the name attribute, and value is the result of presentation on the content of the attribute element. The third omstyle element in Figure 3.37, contains a contrived way of specifying a HTML hyperlink by an element element, with an enclosed attribute element, which obtains its value from an OMOBJ in the with element in the OMDOC source. Note that this is not the way hyperlinks should be specified in OMDOC (a construction with the omlet element is intended for that, see Figure 3.32), since this construction depends on the availability presentation information for every output format.

This leads us to the remaining style element in OMDOC. The value-of element is an empty element and has a required attribute select, whose value is an XPATH expression. It adds the value (a string) the XML node specified by the expression to the output.

Note that this OMDOC-internalized subset of XSLT restricts the expressivity of the presentation style by leaving out the computational features of XSLT. Firstly, the infrastructure for iteration, recursion, variables declaration, . . . is not present, and secondly, path expressions are restricted to pure XPATH [Cla99], leaving out the XSLT extensions, again leaving us with a more declarative subset of XSLT.

Note that the infrastructure discussed in this section is a new extension introduced in OMDOC1.1. It is intended to allow introduction of style information into OMDOC in a controlled way, which is necessary to preserve information when migrating legacy documents into OMDOC. The fact that a transformation engine can choose to ignore these presentation directives since they do not carry any content information shows that authors should not use them instead of identifying the content contribution of the various notational conventions found in legacy documentss. At the moment there is not a mature meta-language for succinctly specifying presentation as for the notations of symbols, so for the time being straight XSLT content will used predominantly. We expect that with time suitable abbreviations will evolve and find their way into OMDoc.

3.5.2 Specifying the Notation of Mathematical Symbols

In this section we discuss the problem of specifying the notation of mathematical symbols in OMDoc. The approach taken is very similar to the one for OMDoc elements above. The mathematical concepts and symbols introduced in an OMDoc document (by `symbol` elements or implicitly by abstract data types) often carry typographic conventions that cannot be determined by general principles alone. Therefore, they need to be specified in the document itself, so that typographically good representations can be generated from this (and subsequent) documents. The normal way to generate presentation from XML documents is to use XSLT style sheets (see section 4.1 for other applications).

```
<xsl:template match="OMBIND[OMS[position()=1 and
                                @name='forall' and
                                @cd='quant1']]">
  <xsl:text>∀</xsl:text>
  <xsl:for-each select="OMBVAR"/>
   <xsl:apply-templates/>
   <xsl:if test="position()!=last()">,</xsl:if>
  </xsl:for-each>.
   <xsl:apply-templates select="*[3]"/>
</xsl:template>
```

Figure 3.38: An XSLT template for the universal quantifier

Let us build up our intuition by an example. We want to include presentation information for the universal quantifier. Since we want to present the structure of complex formulae using this information, we would use XSLT templates like the one shown in Figure 3.38. The match attribute specifies that this presentation rule is applicable to `OMBIND` elements, where the first child is of the form `<OMS cd="quant1" name="forall"/>`. In such a node, it will print the quantifier ∀, then the bound variables as a comma-separated list (for each of the children of `OMBVAR` it recursively applies XSLT templates from the style sheet), print a dot, and then recurse on the third child of the `OMBIND`. This template will cause an OpenMath expression in Figure 3.39 as $\forall P, Q.P \vee Q \Rightarrow Q \vee P$ assuming appropriate templates for implication and and disjunction.

```
<OMBIND>
 <OMS cd="quant1" name="forall"/>
 <OMBVAR><OMV name="P"/><OMV name="Q"/></OMBVAR>
 <OMA>
  <OMS cd="logic1" name="implies"/>
  <OMA><OMS cd="logic1" name="or"/><OMV name="P"/><OMV name="Q"/><OMA>
  <OMA><OMS cd="logic1" name="or"/><OMV name="Q"/><OMV name="P"/><OMA>
 </OMA>
</OMA>
```

Figure 3.39: An OpenMath object presented as $\forall P, Q.P \vee Q \Rightarrow Q \vee P$

To annotate a symbol with presentation information OMDoc supplies the `presentation` element, this is a top-level element whose `for` attribute points to the symbol in question. The simplest (and least effective) way to introduce style sheet information in OMDocs would be to literally include this template declaration (using an Xml `CDATA` section) in a `presentation` in the OMDoc where the symbol is defined.

Note that hand-coding XslT-templates is a tedious and error-prone process, and that we need a template for each output format (e.g. LaTeX, HtML, presentation MathMl, and ASCII), and even various output languages (the greatest common divisor of two integers is expressed by the symbol *gcd* in English but *ggT* ("größter gemeinsamer Teiler") in German). Obviously, the respective templates for all of these transformations share a great deal of structure (in our example, they only differ in the representation of the glyph for the quantifier itself). Therefore OMDoc goes another way and supplies a set of abbreviations that are sufficient for most presentation applications. The user only needs to specify the relevant information and a separate translation process generates the needed XslT templates from that (see section 4.1). We have already seen the use of `style` and `xslt` elements for specifying the presentation of OMDoc elements in the last subsection. In this section we will present yet another way to specify presentation information that is specialized to notations of mathematical symbols. The main idea is specify the properties of mathematical symbols symbolically in relation to the representations of their children and siblings.

As much of the presentation information is shared between various output formats and languages, it is specified in two steps by using a set of attributes we will explain below. The `presentation` element contains the information that is common to all notations in its attributes. It contains a set child elements that specify the presentation directives. These children

Element	Attributes		D	Content
	Required	Optional	C	
ignore		type, comment	–	ANY
omstyle	element	for, id, xref, style	–	(style|xslt)*
element	name		–	(attribute | element | text | recurse | value-of)*
attribute	name		–	(#PCDATA | value-of | text)*
text			–	(#PCDATA)
value-of	select		–	EMPTY
recurse		select	–	EMPTY
presentation	for	id, xref, fixity, parent, lbrack, rbrack, separator, bracket-style, style, precedence, crossref-symbol, theory	–	(use | xslt | style)*
xslt	format,	xml:lang, requires	–	CDATA
style	format,	xml:lang, requires	–	(element | text | recurse | value-of)*
use	format	xml:lang, requires, larg-group, rarg-group, fixity, lbrack, rbrack, separator, crossref-symbol, element, attributes	–	ANY

Figure 3.40: The OMDoc Elements for Presentation Information

are the `style` and `xslt` elements defined in the last subsection, or `use` elements that may only occur in `presentation` elements. The `use` elements make use of the same symbolic attributes and specialize (over-define) these attributes according to the respective format and language. The following set of attributes are particular to the `presentation`, since they are independent of the language and the output format.

`for` this required attribute specifies the name of the symbol for which the notation information is specified.

`theory` allows us to specify the theory of a symbol. This allows the use of presentation elements outside of an enclosing `theory` element. This is important, since the theory information is essential to identify the symbol, but sometimes `presentation` elements in other documents need to be used to override or augment those in the original theory file (which can in general not be changed).

`xref` This optional attribute can be used to refer to another existing `presentation` element. This is often convenient if the same symbols are defined in different theories (but the presentation stays the same).

`parent` This attribute specifies parent element, in which the symbol plays the head role (it can be one of `'OMA'`, `'OMBIND'`, and `'OMATTR'`). In examples in Figure 3.42, we have assumed the head to be an `OMA` element (for functional application). It can also be an `OMBIND`, as in the case of a quantifier in Figure 3.43.

`style` (see the specification for `omstyle` in the last section)

`fixity` This optional attribute can be one of the keywords `'prefix'` (the default), `'infix'`, `'postfix'`, and `'assoc'`. If it is given, then it determines the placement of the function symbol. For `'prefix'` it is placed in front of the arguments, (this is the generic mathematical function notation). For `'postfix'` the function is put behind the arguments, e.g. for derivatives: f'. The case `'infix'` is reserved for binary operators, where the function is inserted between the two arguments. Finally, `'assoc'` is used for associative operators like addition, it puts the function symbol between any two arguments.

Note that `'infix'` is almost a special case of `'assoc'`, but since it is reserved for binary operators, it disregards any arguments but the first two.

`bracket-style` The `fixity` information can be combined with the bracketing style, which can be either of `'lisp'` (LISP-style brackets) or `'math'` (generic mathematical function notation, which is the default).

Figure 3.42 shows some combinations of attributes and their results on the function style.

`precedence` allows us to specify the operator precedence in order to elide unnecessary brackets. The OMDoc presentation system orients itself on the Prolog standard: lower precedences mean stronger binding, and brackets can be omitted. Following Prolog, we give the default precedence 1000, and other precedences as specified in Figure 3.41. As a consequence, formulae like

```
<OMA>
 <OMS cd="arith1" name="power"/>
 <OMA>
  <OMS cd="arith1" name="plus"/>
  <OMV name="x"/>
  <OMV name="y"/>
 </OMA>
 <OMI>2</OMI>
</OMA>
```

```
<OMA>
 <OMS cd="arith1" name="plus"/>
  <OMV name="x"/>
  <OMA>
   <OMS cd="arith1" name="power"/>
   <OMV name="y"/>
   <OMI>2</OMI>
  </OMA>
 </OMA>
```

are presented as $(x+2)^2$ and $x+y^2$.

Number	operators	comment
200	+,-	unary
200	ˆ	exponentiation
400	*	multiplicative
500	$+,-,\wedge,\vee,\cup,\cap$	boolean
600	/	fraction
700	$=,\neq,\leq,<,>,\geq,$	relation

Figure 3.41: Predefined operator precedences in OMDoc

The next set of attributes can occur both in `presentation` and `use` elements. If they occur in both, then the values of those specified on the `use` elements take precedence over those specified in the dominating `presentation` element.

`lbrack/rbrack` These two attributes can be used to specify the brackets to be used in presentation of a complex expression. They will be used unless elided according to the precedence.

`separator` This specifies the separator to be used for separating the arguments. The default for this is the comma. See Figure 3.42 for some combinations.

fixity	bracket-style	separator	yields
prefix	lisp	" "	$(f\ 1\ 2\ 3)$
postfix	lisp	" "	$(1\ 2\ 3\ f)$
prefix	math	","	$f(1,2,3)$
postfix	math	","	$(1,2,3)f$
assuming `lbrack="("` and `rbrack=")"`			

Figure 3.42: Attribute-combination and Function Style

`crossref-symbol` This attribute specifies which parts of the symbol presentation elements cross-references should be attached to: in some formats like HTML, and recently also in LaTeX (thanks to the `hyperref.sty` package), it may be useful to attach a hyperlink from the symbol name to its definition. Some symbols are constructed by using the `lbrack` and `rbrack`, or the `separator` attributes as part of the symbol presentation. For instance, in the notation (a, b) for pairs, the binary function symbol for pairing is really composed of three parts "(", ")", and ",", which should be cross-referenced. The attribute values `'no'`, `'yes'`, `'brackets'`, `'separator'`, `'lbrack'`, `'rbrack'` `'all'`, can be used to specify this behavior. `'no'` means cross-referencing is forbidden, `'yes'` – which is the default value – means cross-referencing only on the print-form of the function symbol, `'lbrack'`, `'rbrack'`, `'brackets'`, only on the (left, right, both) brackets, `'separator'`, on the separator, and finally `'all'` on all presentation elements.

In Figure 3.43, the effect of the default `'yes'` can be seen in the lower part of the figure :the LaTeX and the HTML presentations have attached hyperlinks to the representation of the universal quantifier.

The next set of attributes can only appear on the `use` attribute, since they are only meaningful for selected output formats.

`format`, `xml:lang`, `requires` (see the specification for `xslt` and `style` above).

`larg-group/rarg-group` These two attributes, which only appear in the `use` element, can be used to specify the grouping constructs for driving

Notation specification	Example

`<presentation for="forall"`
`              parent="OMBIND"`
`              separator=".">`
` <use format="TeX">\forall</use>`
` <use format="html">∀</use>`
`</presentation>`
	`<OMBIND>`
	` <OMS cd="quant1" name="forall"/>`
	` <OMBVAR><OMV name="X"/></OMBVAR>`
	` <OMS cd="logic1" name="true"/>`
	`</OMBIND>`

using XSLT templates induced from the left the `presentation` element on the right OpenMath expression yields

```
LaTeX:  \href{../ocd/logic1.ps#true}{\forall}X.
        \href{../ocd/logic1.ps#true}{{\sf true}}
HTML:   <a href="../ocd/logic1.html#forall">&#8704;</a> X.
        <a href="../ocd/logic1.html#true"><b>true</b></a>
```

which in turn is formatted to $\forall X.\mathsf{true}$, only that the symbol $\forall$ carries a hyperlink to it definition (given a suitable output device like a browser or a recent version of `dvips`).

Figure 3.43: Notation for `forall` (cf. Figure 3.38) using `presentation`

the tokenizer of the output formatter. Take for instance the presentation for sums in TeX. We want to use the `\sum` macro for this. `\sum` takes three arguments: e.g. `$\sum^n{i=1}g(i)$`. To be able to use this, we need to have a way to generate the TeX grouping characters "`{`" and "`}`" in the second argument.

`element/attributes/fixity/bracket-style` These attributes simplify the specification of notations in XML-based formats, like MathML. The `element` attribute contains the name and the `attributes` the attribute declarations of an XML element that takes the place of the brackets specified in the attributes `lbrack` and `rbrack`. The attribute `fixity` may only be used on a `use` element in conjunction with the `element` and `attributes` attributes, then it specifies the position of the element brackets rather than the brackets specified in the `lbrack` and `rbrack` attributes.

For instance, the binomial coefficient is usually presented as $\binom{n}{m}$ (and spoken "n choose m") is represented as

```
<mfrac linethickness='0'><mi>n</mi><mi>m</mi></frac>
```

in presentation MathML. The first `presentation` element in Figure 3.44 shows a `presentation` element that has this effect. The second presentation element in Figure 3.44 shows a notation declaration, which applied to 3^5 in HTML would yield 3⁵.

Note that the element and attributes attributes can be simulated by the are a variant of the lbrack and rbrack attributes. The attributes in the binomial example could have been substitutes by the values <mfrac linethickness='0'> for lbrack and </mfrac> for rbrack. Thus these attributes are are not strictly necessary, but convenient and more legible.

```
<presentation for="binomial" parent="OMA">
 <use format="default" fixity="infix">choose</use>
 <use format="TeX"
      lbrack="\bigl({" rbrack="}\bigr)">\atop</use>
 <use format="pmml"
      element="mfrac" attributes="linethickness='0'"/>
</presentation>

<presentation for="power" parent="OMA" fixity="infix"
   crossref-symbol="no" precedence="200" bracket-style="lisp">
 <use format="html" fixity="prefix" bracket-style="math"
      element="sup"/>
 <use format="TeX">^</use>
 <use format="pmml" element="msup" fixity="prefix"/>
</presentation>
```

Figure 3.44: Presentation for binomial coefficients

Conceptually, the attributes of the presentation and use elements form a meta-language for XSLT style sheets that aims at covering the most common notations succinctly and legibly. There are situations, where this language does not suffice, since the notations are too complex. In this case, we can set the attribute system of the use element to 'xsl' (all attributes except parent become meaningless in this situation) and directly include the body of a XSLT template. The information in Figure 3.45 will induce a template that generates the TeX representation {\root{3}\of{5}} for $\sqrt[3]{5}$.

3.6 Identifying and Referencing OMDoc Elements

In this section we will finally address an issue we have only treated very superficially until now: the intended values of the identity attribute id and referencing attributes like xref or for. As we have seen, we need element

```
<presentation for="root" parent="OMA" bracket-style="lisp">
 <use format="TeX" system="xsl">
  <xsl:text>{\root{</xsl:text>
  <xsl:apply-templates select="*[3]"/>
  <xsl:text>}\of{</xsl:text>
  <xsl:apply-templates select="*[2]"/>
  <xsl:text>}}</xsl:text>
 </use>
 <use format="html" system="xsl">
  <sup><xsl:apply-templates select="*[3]"/></sup>
  <xsl:text disable-output-escaping="yes">&#8730;</xsl:text>
  <xsl:apply-templates select="*[2]"/>
 </use>
 <use format="pmml" element="mroot"/>
</presentation>
```

Figure 3.45: `use` elements with `<use system="xsl"...>`

references in OMDoc, since not all mathematical structures can be directly modeled by the Xml tree structure provided by the OMDoc elements. Moreover, since mathematical documents are seldom fully self-contained, intra-document references do not suffice, and we must be able to reference objects in other documents and theories.

In OMDoc version 1.0 [Koh00c] we have presented a mechanism for inter-document reference for OpenMath symbols (`OMS`) based on a catalogue of theory locations and left the identification of other OMDoc elements unspecified. This has turned out to be a stumbling block for tool development, so we will attempt a specification of an more general URI-based approach to element identification and reference here. Note that this specification is only a first attempt to obtain experience, and is likely to be adapted in later versions. As version 1.1 is only a minor update, we will leave the catalogue-based mechanism in place unchanged. The intention is to obtain implementation experience with the new URI-based mechanism in order to reach well-founded decision for OMDoc 2.0.

We will first present the catalog-based mechanism for identifying `OMS` elements, and then present the more general URI-based solution in sections 3.6.2 and 3.6.3.

The problem we need to address for referencing in OMDoc is that there are two ways to access mathematical knowledge: by location (relative to a particular document or file), and by context (relative to a mathematical theory). The first one essentially makes use of the organization structure of file systems (this is the default organization of the Internet), and the second makes use of mathematical structuring principles supplied by the OMDoc format (cf. section 3.3). Both approaches to resource identification have their justification and are therefore supported by OMDoc. Resource identification by document has the advantage that it can be readily be mapped to current practice and transport protocols of the Internet. It has the problem that location-independence is hard to achieve, reference by context must be supported by some form of cataloging service, but gives more structured and semantical access.

Unfortunately, we cannot readily reduce one the two modes of identification onto the other. The idea to require one theory per document is much too restrictive. It is standard practice in mathematics to develop mathematical theories decentrally. Once there is a definition of a theory in place (e.g. in an academic journal), other researchers add theorems to the theory in other documents. Furthermore it is a necessary requirement for a representation format to be closed under concatenation. This is only possible, if we allow multiple theories per document.

3.6.1 Locating `OMS` elements by the OMDoc Catalogue

As we have seen above, OpenMath uses identification by context to reference symbols: `OMS` elements are identified by their `cd` and `name` attributes. The first identifies the theory or OpenMath content dictionary (which are taken to be equivalent in the OMDoc format), and the second the symbol in that theory. This is a valid approach to identification, but for referencing, it assumes that it is always known, where the defining document (i.e. the OMDoc document that contains the theory) can be found, which may not always be obvious.

If we know where the defining OMDoc is, then reference by location and reference by context are equivalent, since the theory identifier is unique in any valid OMDoc document. Therefore, OMDoc supports a catalogue mechanism that allows to specify the location of defining OMDocs. This can be done in two ways

globally The global specification of a catalogue is done by the `catalogue` attribute in the `omdoc` element. It is a URI reference to another OMDoc document whose catalog is inherited by the referencing one.

locally The local catalogue is declared in the `catalogue` element, it contains a sequence of location declarations, i.e. empty `loc` elements, which have the attributes `theory` and `omdoc`. They declare that the theory specified by the theory attribute is contained in the OMDoc document referenced in the omdoc attribute.

The effective catalogue for an OMDoc is a sequence of location declarations. It is (recursively) computed in the following way. First, the effective catalogues of the OMDocs at the URIs given in the catalogue attribute of the omdoc element are concatenated in the given order, and the local catalogue declaration is appended at the end. Then double location declarations are eliminated, later declarations overwriting the earlier. This effective catalogue is used to determine the location of any theory referenced in the OMDoc.

```
<omdoc id="allthree">
 ...
 <catalogue>
  <loc theory="monoids" omdoc="http://activemath.org/coll/algebra"/>
  <loc theory="reals" omdoc="http://activemath.org/coll/analysis"/>
  <loc theory="int" omdoc="http://activemath.org/coll/cds"/>
 </catalogue>

 <OMOBJ>
  <OMA>
   <OMS cd="monoids" name="op"/>
   <OMS cd="reals" name="pi"/>
   <OMS cd="int" name="zero"/>
  </OMA>
 </OMOBJ>
 ...
</omdoc>
```

Figure 3.46: A catalogue for OpenMath Symbols

One of the applications of having the location information given in the catalogue is that we can use this for cross-referencing in output formats generated from OMDoc documents.

3.6.2 A URI-based Mechanism for Element Reference

The problem with the catalogue-based approach to identification and reference is that it is limited to OpenMath symbols, which are traditionally referenced by context. It could be extended to referencing theory-constitutive elements, since they obey the implicit assumption that theories do not transcend OMDoc documents. For non-constitutive elements this is not the

Element	Attributes		D	Content
	Required	Optional	C	
omdoc	id, version, xmlns	type, catalogue, style, xmlns, version, xmlns:xsi, xsi:schemaLocation	+	OMDoc element*
catalogue			–	loc*
loc	theory	omdoc, cd	–	EMPTY

Figure 3.47: The OMDoc Elements for Identification

case, since they can be added to a theory in separate documents later. Furthermore, generalizing the catalogue-based approach would involve adding optional attributes for identifying the theory and the element id relative to that theory, which would clutter up the format. In particular, it cannot directly be adapted to content MathMl `csymbol` elements, since those only have the `definitionURL` attribute, which is a uniform resource identifier (URI) [BLFM98]. OMDoc adopts an URI-based approach, since uniform resource locators (URL) are not sufficient to support location-independence (mathematical data tends to move e.g. when it is published) and web-services like caching.

A URI reference is traditionally considered to consist of two parts. A URI proper and a fragment identifier separated by a hash sign `#`. The URI identifies an Xml document on the web, whereas the second part identifies a fragment of the document, which in the case of OMDoc will usually be an OMDoc element. Xml provides the XPointer language [DJM01] that specifies an element in the document identified by `<uri>` by the URI reference `<uri>#xpointer(<path>)`, where `<path>` specifies a path through the document tree leading to the desired element. URI-references of the form `<uri>#<id>` as they are used in HtML to refer to named anchors (`<a name="id"/>`) are regained as a special case (the so-called bare name syntax): If `<uri>` is a URI of an Xml document D then `<uri>#<id>` refers to the unique element in D, that has an attribute of type `ID` with value `<id>`. Thus we can directly use the standard XPointer fragment identifiers for reference to OMDoc elements by location. Note that since most OMDoc `id` attributes do not have type `ID` in the document type definition, we cannot use bare name syntax in most cases, but have to use the full syntax using explicit #xpointer(...).

Furthermore note that to get reference by context, we have to extend the fragment identifier, and can use the URI part unchanged. Concretely, we will use the URI references of the form `<uri>#byctx(<name>@<thy>)`, where `<thy>` identifies a `theory` element in a theory collection and `<name>` is the value of a `id` attribute of an OMDoc element in this theory.

```
<omdoc id="o1" xml:base="http://mbase.mathweb.org/o1.omdoc">
 <theory id="th1">...<symbol id="x"/>...</theory>
 <assertion id="a1" theory="th1">...</assertion>
 <theory id="th2">...<symbol id="y"/>...</theory>
 <ref xref="http://mbase.mathweb.org/o2.omdoc"/>
</omdoc>
```

```
<omdoc id="o2" xml:base="http://mbase.mathweb.org/o2.omdoc">
 <theory id="th3">...<symbol id="z"/>...</theory>
 <assertion id="a2" theory="th2">...</assertion>
 <assertion id="a3" theory="th3">...</assertion>
</omdoc>
```

element	URI
x	http://mbase.mathweb.org/o1.omdoc#byctx(x@th1)
a1	http://mbase.mathweb.org/o1.omdoc#byctx(a1@th1)
y	http://mbase.mathweb.org/o1.omdoc#byctx(y@th2)
z	http://mbase.mathweb.org/o1.omdoc#byctx(z@th3)
a2	http://mbase.mathweb.org/o1.omdoc#byctx(a2@th2)
a3	http://mbase.mathweb.org/o1.omdoc#byctx(a3@th3)

Figure 3.48: An OMDoc specifying a theory collection

Figure 3.48 gives some examples of reference by context; the use of the `xml:base` attribute (see [mar01]) is only for convenience in locating the document URI. The first OMDoc document defines theories `th1` and `th2` and includes the second document by the `ref` element. As a consequence, the elements in the second document are accessible for reference by context (but not for reference by location) through the `o1.omdoc`. Moreover reference by context makes the assertion `a1` is accessible as part of the theory `th1` even though it is not directly dominated the respective `theory` element. Assertion `a2` is accessible as part of theory `th2`, even though it is not even in the same document.

Clearly reference by context facilitates the maintenance of theory collections by shifting the location burden onto the retrieval services. Note that the OMDoc specification only uses reference by context to define the identification of OMDoc elements. The actual implementation of retrieval services is not object of the specification and is left to OMDoc applications, such as the ones described in chapter 4. Note that in principle assertion `a3` in Figure 3.48 is also accessible by the URI reference `http://mbase.mathweb.org/o2.omdoc#byctx(a3@th3)`, i.e. via the second document. This leads to a situation, where it is non-trivial to decide whether two elements are actually identical, which may lead to difficulties for mathematical software systems. A similar complication arises, if the inclusion graph induced by the `ref` elements is not a tree.

Generally, the problem here is that the identification mapping from URIs to objects is not injective, and does therefore not have a partial inverse. As a consequence, deduce that two referred elements are identical by looking at their URI, but not that they are different. Note that the problem of a non-injective identification mapping is even more pertinent to reference by document.

As injectivity of the identification mapping is in general a desirable property, designers of theory collections should take this into account, e.g. by designating canonical entry documents. Since the mechanism is still new in OMDoc1.1, we do not prescribe any mechanism for ensuring injectivity, but leave it to the application developers to form a consensus.

3.6.3 Uniqueness Constraints and Relative URI references

Since many OMDoc documents are still written by hand, notational convenience is an important concern. Therefore URIs can be shortened in OMDoc by abbreviation just like other relative URIs. Moreover, we allow relative fragment identifiers that are licensed by certain uniqueness presuppositions in OMDoc

For the `byctx` fragment identifier to work at all, `id`-values of OMDoc elements must be unique in their home theory, and those of `theory` in the theory collection (i.e. the OMDoc document that would result from executing all the inclusions mandated by the `ref` elements). The uniqueness constraint in home theories also includes mathematical statements whose `theory` attribute points to this theory, and their descendents, if they are members of the same collection. Note that the process of adding a theory to a theory collection includes consistently renaming `id` attributes so that these uniqueness constraints discussed above are respected.

Note that OMDoc does not mandate uniqueness in OMDoc documents of `id` attributes on elements other than `theory` in order to ensure concatenability. As a consequence, the OMDoc document type definition does not give them the type `ID`, which would enforce document-wide uniqueness upon

DTD-validation, and we cannot use XPointer bare name syntax in most cases. In those cases, where we can, e.g. for theories, we will XPointer takes precedence over the `byctx` fragment identifier, to maintain Xml standards compliance.

We use the following rules for relative and abbreviated URI references:

`<uri>#mythy` This references the theory whose attribute `id` has the value `'mythy'`. This is actually a direct application of the XPointer bare name syntax, as the `id` has type `ID`. In this case, the `<path>` component must be empty.

`<reluri>#<frag-id>` Here `<reluri>` is a relative URI (cf. [BLFM98]). Thus the relative URI reference expands to `<uri>#<frag-id>` independently of the fragment identifier, if `<reluri>` expands to `<uri>` by the rules in [BLFM98].

In particular, the URI abbreviations defined in Xml Base are allowed (for details see [mar01]).

If the fragment identifier marker character `#` is not present in a URI reference, then we assume it to be an abbreviation of the `byctx` fragment identifier in the local document (theory) collection. Thus we have the following abbreviations.

`name` abbreviates `#byctx(name@<this_theory>)`, if it does not contain the hash character (`#`) or is an absolute URI. Used as the value of an attribute of an element E, such that E is inside a `theory` element whose `id` has value `'mythy'`, or E has a `theory` attribute with value value `'mythy'`, this relative URI identifies the unique element N whose `id` attribute has value `'name'`, and which is either in the same `theory` element, or which is in an OMDoc document in the same collection as E, and whose `theory` has value `'mythy'`. Note that there is no conflict with XPointer's bare name syntax, since no # is present.

Note that this case syntactically subsumes cases like `arith1.omdoc`. This is interpreted as `#byctx(arith1.omdoc@<this_theory>)`, and not as `file://arith1.omdoc`. Use `arith1.omdoc#` instead.

`name@mythy` abbreviates `#byctx(name@mythy)`. From anywhere in the collection, a reference with this value points to an element whose `id` attribute has value `'name'` and that is a descendent of the unique `theory` element whose `id` attribute has value `'mythy'`.

```
<!DOCTYPE omdoc PUBLIC "-//OMDoc//DTD OMDoc V1.1//EN"
                       "http://www.mathweb.org/omdoc/omdoc.dtd"
  [<!ENTITY % theoryNSD "xmlns:ida CDATA #IMPLIED
                         xmlns:ain CDATA #IMPLIED
                         xmlns:acd CDATA #IMPLIED">]>
<omdoc id="allthree"
  xmlns:ida="http://www.riaca.org/ida.omdoc"
  xmlns:ain="ftp://ftp.activemath.org/pub/ana.xml"
  xmlns:acd="x-mbase://cds@mathweb.org">
 ...
 <OMOBJ>
  <OMA>
   <OMS cd="ida:monoids" name="op"/>
   <OMS cd="ain:reals" name="pi"/>
   <OMS cd="acd:int" name="zero"/>
  </OMA>
 </OMOBJ>
 ...
</omdoc>
```

Figure 3.49: Symbols from three different collections

For OpenMath symbols OMDoc uses a syntactical variant of the `byctx` fragment identifiers to maintain some kind of backwards compatibility. We use the document URI of the collection as namespace for the theories it contains. Instead of writing the identifying URI of a symbol in one piece, we write it in three chunks, using the `cd` attribute for the theory name (as in pure OpenMath but prefixed by the collection as a namespace) and the `name` attribute for the `id` part. Figure 3.49 shows a fragment of an OMDoc document that uses symbols from theories from three different collections.

Note that the namespace declarations in the OMDoc element cannot be declared in the OMDoc DTD, since they are not fixed. Therefore the DTD (see E) supplies an entity `theoryNSD` for extra namespace declarations. I can be defined in the local subset of the DTD as in Figure 3.49. Xml schemata are namespace aware, so if we only want to perform schema-validation, we do not need the `DOCTYPE` declaration or the internal subset.

Incidentally in MathMl, which has a `definitionURL` attribute, we can directly use the full URI, as in Figure 3.50, as OMDoc2.0 will include content MathMl as a representation format for mathematical objects. This is an important prerequisite for resource identification.

```
<math xmlns:m="xmlns:mml="http://www.w3.org/1998/Math/MathML">
 <apply>
  <csymbol definitionURL="http://www.riaca.org/ida.omdoc#byctx(op@monoids)"/>
  <csymbol definitionURL="ftp://ftp.activemath.org/pub/ana.xml#byctx(pi@reals)"/>
  <csymbol definitionURL="x-mbase://cds@activemath.org#byctx(zero@int)"/>
 </apply>
</math>
```

Figure 3.50: C-MathML symbols from three different collections

Chapter 4

OMDoc Applications, Tools, and Projects

In this chapter we will address current applications, tools and projects using the OMDoc format. We will first discuss the possibilities and tools of processing documents in the OMDoc format via stylesheets with the purpose of generating documents specialized for consumption by other mathematical software systems, and by humans. Then we will present three projects descriptions that use OMDoc at the core. The QMath project described in section 4.2 defines an interface language for a fragment of OMDoc, that is simpler to type by hand, and less verbose than the OMDoc, that can be generated by the `qmath` batch processor. The MBase system in section 4.3 is a a web-based mathematical knowledge base that offers the infrastructure for a universal, distributed repository of formalized mathematics represented in the OMDoc format. Finally, the ActiveMath projects described in section 4.4 uses the OMDoc infrastructure in an educational setting. It makes use of the content-orientation and the explicit structural markup of the mathematical knowledge to generate on the fly specialized learning materials that are adapted to the students prior knowledge, learning goals, and notational tastes.

The applications of OMDoc are not limited to the ones described in this chapter, in fact there is research and tool development where OMDoc is used in the role of

- a communication standard between mechanized reasoning systems, e.g. the Clam-Hol interaction [BSBG98], or the Ωmega-TPS [BBS99] integration.

- a data format that supports the controlled refinement from informal presentation to formal specification of mathematical objects and theories. Basically, an informal textual presentation can first be marked up, by making its structure explicit (classifying text fragments as definitions, theorems, proofs, linking text, and their relations), and then formalizing the textually given mathematical knowledge in logical formulae (by adding `FMP` elements; see section 3.2.1.

- an interface language of a mathematical knowledge base like the MBase system [FK00, KF00]. The system offers a service that allows to store and (flexibly) reproduce (parts of) OMDoc documents.

- a document preparation language; a system like MBase supports the maintenance of large-scale document- and conceptual structures, if they are made explicit in OMDoc. As OMDoc can directly be transformed to e.g. XHtML+MathMl, or LaTeX, external input to MBase can directly be published.

- a basis for *individualized (interactive) books*. Personalized OMDoc documents can be generated from MBase making use of the discourse structure encoded in MBase together with a user model.

- an interface for proof presentation [HF97, Fie99]: since the proof part of OMDoc allows small-grained interleaving of formal (`FMP`) and textual (`CMP`) presentations in multiple languages.

Note that the material discussed in this chapter is under continuous development, and the account here only reflects the state of December 2001, see `http://www.mathweb.org/omdoc` for more and current information. The text in in the project descriptions has been contributed by the authors marked in the section headings, for questions about the projects or systems, please visit the web-sites given in the section headings or contact the authors directly.

4.1 Transforming OMDoc by XslT Style Sheets

In the introduction we have stated that one of the design intentions behind OMDoc is to separate content from presentation, and leave the latter to the user. In this section, we will briefly touch upon presentation issues. The technical side of this is simple: OMDoc documents are regular Xml documents that can be processed by an XslT style sheet to produce other formats from OMDoc representations. In this section we will review a set of XSLT style sheets that are distributed with OMDoc, they can be found in `http://www.mathweb.org/omdoc/xsl`.

There are several high-quality XSLT transformers freely available (e.g. `saxon` (`http:saxon.sourceforge.net`) or `xalan` (`http://xml.apache.org/xalan-j`)). Moreover, XSLT is natively supported by the newest versions of the primary browsers MS Internet Explorer and Netscape Navigator (see `http://www.mozilla.org` for Mozilla, the open source version).

XSLT style sheets can be used for several tasks in maintaining OMDoc, such as for instance converting other XML-based input formats into OMDoc (e.g. `cd2omdoc.xsl` for converting OpenMath content dictionaries into OMDoc format), or migrating between different versions of OMDoc e.g. the style sheet `omdoc1.0adapt1.1.xsl` that operationalizes all the syntax changes from OMDoc version 1.0 to version 1.1 (see appendix B for a tabulation).

4.1.1 OMDoc Interfaces for Mathematical Software Systems

One of the original goals of the OpenMath, MathML and OMDoc languages is to provide a communication language for mathematical software systems. The main idea behind this is to supply systems with interfaces to a universally accepted communication language standard (an interlingua), and so achieve interoperability for n systems with only $2n$ translations instead of n^2. As we have seen in section 2.3, OpenMath and content MathML provide a good solution at the level of mathematical objects, which is sufficient for systems like computer algebra systems. OMDoc adds the level of mathematical statements and theories to add support for automated reasoning systems and formal specification systems.

To make practical use of the OMDoc format as an interlingua, we have to support building OMDoc interfaces. An XSLT style sheet is a simple way to come up with (the input half) of an OMDoc interface, a more efficient way would be to integrate an XML parser directly into the system (suitable XML parsers are readily available for almost all programming languages now).

Usually, the task of writing an XSLT style sheet for such a conversion is a relatively simple task, since the input language of most mathematical software system is isomorphic to a subset of OMDoc. This suggests the general strategy of applying the necessary syntax transformations (this has to be supplied by the style sheet author) on those OMDoc elements that carry system-relevant information and transforming those that are not (e.g.

Metadata and `CMP` elements for most systems) into comments. Much of the functionality is already supplied by the style sheet `omdoc2sys.xsl`, which need only be adapted to know about the comment syntax. For examples see the `omdoc2pvs.xsl` style sheet that transforms OMDoc to Pvs input.

The other direction of the translation needed for communication is usually much more complicated, since it involves parsing the often idiosyncratic output of these systems. A better approach (which we followed with the systems above) is to write specialized output generators for these systems that directly generate OMDoc representations. This is usually a rather simple thing to do, if the systems have internal data structures that provide all the information required in OMDoc. It is sometimes a problem with these systems that they only store the name of a symbol (logical constant) and not its home theory. At other times it internal records of proofs in theorem provers are optimized towards speed and not towards expressivity, so that some of the information that had been discarded has to be recomputed for OMDoc output.

One of the practical problems that remains to be solved for interfaces to mathematical software systems is that of semantical standardization of input languages. For mathematical objects, this has been in principle solved by supplying a theory level in the form of OpenMath content dictionaries or OMDoc documents that define the necessary mathematical concepts. For systems like theorem provers or theory development environments this has not been done yet.

OMDoc can help with this task, as we have seen in series of experiments of connecting the theorem proving systems ΩMEGA [BCF+97], InKa [HS96], Pvs [ORS92], λ*Clam* [RSG98], TPS [ABI+96] and CoQ [Tea] to the MBase system by equipping them with an OMDoc interface.

The first observation in the interpretation is that even though the systems are of relatively different origin, their representation languages share many features

- TPS and Pvs are based on a simply typed λ-calculus, and only use type polymorphism in the parsing stage, whereas ΩMEGA and λ*Clam* allow ML-style type polymorphism.

- ΩMEGA, InKa and Pvs share a higher sort concept, where sorts are basically unary predicates that structure the typed universe.

- Pvs and CoQ allow dependent- and record types as basic representational features.

but also differ on many others

- InKa, Pvs, and CoQ explicitly support inductive definitions, but by very different mechanisms and on differing levels.

- CoQ uses a constructive base logic, whereas the other systems are classical.

At one level, the similarities are not that surprising, all of these systems come from similar theoretical assumptions (most notably the Automath project [dB80]), and inherit the basic setup (typed λ calculus) from it. The differences can be explained by differing intuitions in the system design and in the intended applications.

Following recent work on the systemization and classification of λ-calculi [Bar92], we have started to ground these languages in language hierarchy. The structural similarities between theories and logical languages and their structuring morphisms allow to re-use the OMDoc/MBase theory mechanism for language definition: The logical symbols and language constructs can be defined just like other (object-level) symbols/concepts. As a consequence, the development of the OMDoc interface to the theorem provers mentioned above included the specification of the representation language as a theory (which could be used as an integrated documentation). The structured theory mechanism can now be used to re-use and inter-relate the various representation formats between the theorem provers. For instance the simply typed λ-calculus can be factored out (and thus shared) of the representation languages of all of the theorem proving systems above. This makes the exchange of logical formulae via the OMDoc format very simple, if they happen to be in a suitable common fragment: In this case, the common (OpenMath/OMDoc) syntax is sufficient for communication.

4.1.2 Presenting OMDoc to Humans

One of the main goals of content markup for mathematical documents is to be independent of the output format. In the last chapter, we have specified the conceptual infrastructure provided by the OMDoc language, in this section we will discuss the software infrastructure needed to transform OMDoc documents into human-readable form in various formats. We speak of of OMDoc presentation for this task.

Due to the complex nature of OMDoc presentation, only part of it can actually be performed by XslT style sheets. For instance, subtasks like reasoning about the prior knowledge of the user, or her experience with certain proof techniques is clearly better left to specialized applications. Our processing model is the following: presenting an OMDoc is a two-phase process. The first one is independent of the final output format (e.g. HTML, MathML, or LaTeX) and produces another OMDoc representation specialized to the respective user or audience, taking into account prior knowledge, structural preferences, bandwidth and time constraints, etc. This is followed by a formatting process that can be done by XSLT style sheets that transforms the resulting specialized document into the respective output format with notational- and layout preferences of the audience. We will only discuss the second one and refer the reader for ideas about the first process to systems like P.rex [Fie01, FH01].

At the moment, we have XSLT style sheets to convert OMDoc to HTML, presentation MathML, and LaTeX, they can be found at `http://www.mathweb.org/omdoc/xsl`. They consist of two parts: a generic part that implements the presentation decision for the OMDoc (and OpenMath) elements, and a theory-specific part for the presentation of OpenMath symbols.

The first part is carried out by the style sheets `omdoc2html.xsl` for HTML and and `omdoc2tex.xsl` for LaTeX. They share a large common code base `omdoc2share.xsl`, basically the first two include the latter and only redefine some format-specific options. For instance, `omdoc2share.xsl` supplies an infrastructure for internationalization. In section 3.2.1 we have introduced multilingual groups of `CMP` elements. This allows to generate localized presentations of the OMDoc documents, if enough information is present. `omdoc2share.xsl` takes a parameter `TargetLanguage`, whose value can be a whitespace-separated preference list of ISO 639 two-letter country codes. If `TargetLanguage` consists of a single entry, then the result will only contain this language with gaps where the source document contains no suitable `CMP`. Longer `TargetLanguage` preference lists will generally result in more complete documents. Apart from the language-specific elements in the source document, localization also needs to know about the presentation of certain keywords used in OMDoc markup, e.g. the German “Lemma” and the French “Lemme” for `<assertion type="lemma">`. This information is kept in the keyword table `http://www.mathweb.org/omdoc/lib/locale.xml`, which contains all the keywords necessary for presenting the OMDoc elements discussed so far. An alternative keyword table can be specified by the parameter `locale`.

Presentation of OpenMath symbols in formulae is a process based on the presentation information described in section 3.5.2 to re-create their typographic conventions in the output format. To present a file `test.omdoc` in e.g. HTML, we first generate an XSLT style sheet `test2html.xsl` and the apply it to `test.omdoc` to generate the HTML file `test.html`. Note that `test2html.xsl` needs to include specific XSLT templates for all symbols that are used in formulae, so `test2html.xsl` includes the three style sheets

`omdoc2html.xsl` for presentation of the OMDoc elements that are not symbols.

`test4html.xsl` a style sheet that contains templates for symbols that are are defined in `test.omdoc`, it is generated by applying an XSLT meta-stylesheet `expres.xsl` with parameter `format` = `html` to `test.omdoc`. Concretely, if `test.omdoc` defines the symbol `forall` and contains the `presentation` element in Figure 3.43 (page 85), then it generates an XSLT style sheet `fol4html.xsl` that contains the template in Figure 3.38 (page 79).

`omdocIhtml.xsl` this is a style sheet that provides templates for all symbols that are used but not defined in `test.omdoc`. Concretely this is just a list of XSLT `xsl:include` statements that include style sheets `xxx4html.xsl` extracted by `expres.xsl` from files `xxx.omdoc` that define symbols used in formulae in `test.omdoc`. We use the style sheet `exincl.xsl` parameter `format` = `html` to generate `testIhtml.xsl` from `test.omdoc`.

This two-level approach to notation presentation in OMDoc provides a maximum of flexibility and locality in information management.

4.2 QMath: An Authoring Tool for OMDoc

Alberto González Palomo
http://www.matracas.org

QMath is a batch processor that produces an OMDoc file from a plain Unicode text document. The purpose of QMath is to allow fast writing of mathematical documents, using plain text and a straightforward syntax (like in computer algebra systems) for mathematical expressions .

The "Q" was intended to mean "quick", since QMath began in 1998 as an abbreviated notation for MathMl. The first version (0.1) just expanded the abbreviations to full MathMl element names, and added the extra markup such as `<mrow>` and the like. There have been many changes (and two complete rewrites) since then. You can find a more detailed history at http://www.matracas.org/qmath/history.html

QMath is very simple in its design: it just parses a text (UTF-8) file according to a user-definable table of symbols, and builds an XML document from that. The symbol definitions are grouped in files called "contexts". The idea is that when you declare a context, its file is loaded and from then on these symbol definitions take precedence over any previous one, thus setting the context for parsing of subsequent expressions.

The text is split into "paragraphs", which are pieces of text separated by at least one empty line. Each paragraph can have a metadata section at the beginning. There are a variety of classes of paragraphs, which are identified by a name followed by a colon (":"), optionally followed by an identifier which becomes the `id` attribute of the generated OMDoc element. The text is put in a `<CMP>` inside a container element which depends on the paragraph type. This can be anything allowed by OMDoc, such as `<assertion>` , `<axiom>`, or the default `<omtext>` if the paragraph doesn't have a QMath paragraph type label. Inside the text, a mathematical expression is enclosed in dollar ("`$`") signs. Each such a section becomes an `OMOBJ` element in the output document.

Figure 4.1 shows a minimal QMath document, and the OMDoc document generated from it. The first line ("`QMATH 0.3.6`") in the QMath document is required for the parser to recognize the file. The lines beginning with "`:`" are metadata items: first the document title, then the author name (one line for each author), and finally the primary language for the document. This last item is required, as it sets the basic symbol set accordingly. For example, the "`Context`" item of an English document is written "`Contexto`" if the document is in Spanish. (Similarly, the arith-

```
QMATH 0.3.6
:"Diary"
:Winston Smith
:1984-04-04
:en

Context: "Mathematics/Arithmetic"
Context: "Mathematics/OMDoc"

Theory:[<-thoughtcrime]

:"Down with Big Brother"
Freedom is the freedom to say $2+2=4$.
If that is granted, all else follows.
```

From contexts/en/Mathematics/OpenMath/arith1.qmath:

```
Symbol:  plus OP_PLUS "arith1:plus"
Symbol:  + OP_PLUS "arith1:plus"
Symbol:  sum APPLICATION
"arith1:sum"
Symbol:  Σ APPLICATION
"arith1:sum"
...
```

From contexts/en/Mathematics/OpenMath/relation1.qmath:

```
Symbol:  = OP_EQ "relation1:eq"
Symbol:  neq OP_EQ "relation1:neq"
Symbol:  = OP_EQ "relation1:neq"
Symbol:  ≠ OP_EQ "relation1:neq"
...
```

```
<?xml version='1.0' encoding='UTF-8' standalone='no'?>
<!DOCTYPE omdoc PUBLIC "-//OMDoc//DTD OMDoc V1.1//EN"
                       "http://www.mathweb.org/omdoc/omdoc.dtd" []>
<omdoc lang='en'>
 <metadata lang='en'>
  <Title xmlns='http://purl.org/DC'>Diary</Title>
  <Contributor xmlns='http://purl.org/DC' role='aut'>
   Winston Smith
  </Contributor>
  <Date xmlns='http://purl.org/DC'>1984-04-04</Date>
 </metadata>
 <theory id='thoughtcrime'>
  <omtext>
   <metadata>
    <Title xmlns='http://purl.org/DC'>Down with Big Brother</Title>
   </metadata>
   <CMP>
    Freedom is the freedom to say
    <OMOBJ xmlns='http://www.openmath.org/OpenMath'>
     <OMA>
      <OMS cd='relation1' name='eq'/>
      <OMA><OMS cd='arith1' name='plus'/><OMI>2</OMI><OMI>2</OMI></OMA>
      <OMI>4</OMI>
     </OMA>
    </OMOBJ>.
    If that is granted, all else follows.
   </CMP>
  </omtext>
 </theory>
</omdoc>
```

Figure 4.1: A minimal QMath document and its OMDoc result

metic context would be ”Matemáticas/Aritmética”) The document is split into paragraphs, which are separated by empty lines. Then, mathematical expressions are written enclosed by “$” (dollar) signs.

The QMath command works as a pure filter: reads the document from standard input, and writes the resulting OMDoc in standard output. So, the typical usage is

```
QMath <document.qmath > document.omdoc
```

It needs the QMATH_HOME environment variable to contain the path for the root QMath directory, where it can find the ”contexts” directory. For example, if you have the contexts directory at /tmp/qmath_3/contexts, you should set QMATH_HOME to /tmp/qmath_3

QMath is distributed under the GNU General Public License (GPL): http://www.gnu.org/licenses/licenses.html#GPL

4.3 MBase, an Open Mathematical Knowledge Base

Andreas Franke and Michael Kohlhase
http://www.mathweb.org/mbase

We describe the MBase system, a web-based mathematical knowledge base (see http://www.mathweb.org/mbase). It offers the infrastructure for a universal, distributed repository of formalized mathematics. Since it is independent of a particular deduction system and particular logic, the MBase system can be seen as an attempt to revive the Qed initiative from an infrastructure viewpoint. See [KF00] for the logical issues related to supporting multiple logical languages while keeping a consistent overall semantics. The system is realized as a mathematical service in the MathWeb system [FK99], an agent-based implementation of a mathematical software bus for distributed mathematical computation and knowledge sharing. The content language of MBase is OMDoc.

We will start with a description of the system from the implementation point of view (we have described the data model and logical issues in [KF00]).

The MBase system is realized as a distributed set of MBase servers (see figure 4.2). Each MBase server consists of a Relational Data Base Management System (RDBMS) connected to a mOZart process (yielding a MathWeb service) via a standard data base interface. For browsing the MBase content, any MBase server provides an http server (see http://mbase.mathweb.org:8000 for an example) that dynamically generates presentations based on HtML or Xml forms.

This architecture combines the storage facilities of the RDBMS with the flexibility of the concurrent, logic-based programming language Oz [Smo95], of which mOZart (see http://www.mozart-oz.org) is a distributed implementation . Most importantly for MBase, mOZart offers a mechanism called pickling, which allows for a limited form of persistence: mOZart objects can be efficiently transformed into a so-called pickled form, which is a binary representation of the (possibly cyclic) data structure. This can be stored in a byte-string and efficiently read by the mOZart application effectively restoring the object. This feature makes it possible to represent complex objects (e.g. logical formulae) as Oz data structures, manipulate them in the mOZart engine, but at the same time store them as strings in the RDBMS. Moreover, the availability of “Ozlets” (mOZart functors) gives MBase great flexibility, since the functionality of MBase can be enhanced at run-time by loading remote functors. For instance complex data base queries can be compiled by a specialized MBase client, sent (via the

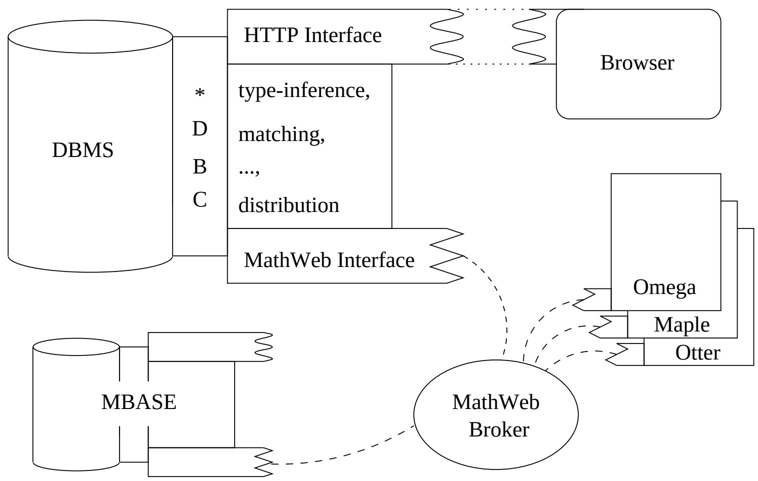


Figure 4.2: System Architecture

Internet) to the MBase server and applied to the local data e.g. for specialized searching (see [Duc98] for a related system and the origin of this idea).

MBase supports transparent distribution of data among several MBase servers (see [KF00] for details). In particular, an object O residing on an MBase server S can refer to (or depend on) an object O' residing on a server S'; a query to O that needs information about O' will be delegated to a suitable query to the server S'. We distinguish two kinds of MBase servers depending on the data they contain: *archive servers* contain data that is referred to by other MBases, and *scratch-pad* MBases that are not referred to. To facilitate caching protocols, MBase forces archive servers to be *conservative*, i.e. only such changes to the data are allowed, that the induced change on the corresponding logical theory is a conservative extension. This requirement is not a grave restriction: in this model errors are corrected by creating new theories (with similar presentations) shadowing the erroneous ones. Note that this restriction does not apply to the non-logical data, such as presentation or description information, or to scratchpad MBases making them ideal repositories for private development of mathematical theories, which can be submitted and moved to archive MBases once they have stabilized.

4.4 Project ActiveMath

Erica Melis, Eric Andrès, Jochen Büdenbender, Adrian Frischauf,
George Goguadze, Paul Libbrecht, Martin Pollet, Carsten Ullrich
http://www.activemath.org/

In a nutshell, ActiveMath is a generic web-based learning system that dynamically generates interactive (mathematics) courses adapted to the student's goals, preferences, capabilities, and knowledge. The content is represented in OMDoc format with several extensions needed in an educational context. For each user, the appropriate content is retrieved from the knowledge base MBase and the course is generated individually according to pedagogical rules. Then the course is presented to the user via a standard web-browser. One of the exceptional features of ActiveMath is its integration of stand-alone mathematical service systems.

Currently, a minimal authoring kit and a translation tool from restricted LaTeX to OMDoc are provided to support authoring OMDocs[1].

In the near future, the authoring tools will have intelligent features and ActiveMath will integrate a user-adaptive suggestion and feedback mechanism in addition to the adaptive course generation. A comprehensive description of the system can be found in [MAF+01].

4.4.1 OMDoc Extensions

The ActiveMath DTD is an extension of the general OMDoc DTD, in particular, with pedagogically motivated extensions such as difficulty or abstractness of an example or exercise.

ActiveMath will also differentiate exercises according to their type with values defining the user's required activity such as *check-question*, *make-hypothesis*, *prove*, *model* or *explore* where e.g., *explore* means interactive exploration with the help of specified external system (such as `maple`, ΩMEGA, or statistics software).

Additional pedagogically motivated metadata elements will be introduced such as `field` (e.g., *computer science*,*math*, *economy*) and `learning-context` with values corresponding to school and university levels[2] in accordance with the Learning Object Metadata Standard[3]. Furthermore, the

[1] See `http://www.activemath.org/~paul/AuthoringComments/` for a description.

[2] These will be used to provide different content to users with different background fields and at different levels.

[3] `http://ltsc.ieee.org/wg12/`

author can specify the pedagogical goal of an exercise, example, or elaboration, that is whether learning this item increases *knowledge*, *comprehension*, *application*, or *transfer*.

Finally, in ActiveMath, relations are going to be represented by the element `relation`, defined in the ActiveMath DTD and classified by introducing a type of the relation. The type values are: *depends-on*, *counterexample-for*, *similar-example*, *similar-exercise*, *citation*, etc. The need for distinguishing the types of relations arises not only in educational contexts.

Since we need certain additional OMDoc elements in the educational context, a common representation for proof methods, proof plans, algorithms will be added in the future, hopefully some of them even to the common OMDoc itself.

4.4.2 Adaptive Presentation

ActiveMath offers dynamically constructed courses that suit the learner's learning goals, her choosen learning scenarios, her presentation preferences, and knowledge mastery. To realize this, ActiveMath maintains a user model and its presentation tools include a course generator and pedagogical rules employed by the course generator.

Presentation of the content is currently made in HtML through XslT transformations with adaptation to the users' taste through CSS filters. OMDoc's semantic encoding allows to envision other output formats and some of them are under work.

4.4.3 Integration of External Systems

Currently, ActiveMath integrates the Computer Algebra Systems MuPad and Maple and the proof planner of ΩMEGA, and statistics software. Moreover, an external student/exercise management systems will be integrated in 2002. The distributed web-architecture of ActiveMath is well-suited for integrating external systems and also the OMDoc representation is – in principle – a basis for integrating different systems.

Currently however, exercises and examples cannot simply pass OMDocs or OpenMath elements to the mathematical service systems because OpenMath-phrasebooks are not available for most systems. An instruction on how to write the OMDocs for exercises for which an external system is called can be found in `http://www.ags.uni-sb.de/~adrianf/activemath`. The abstract description of an exercise includes startup, shutdown, and eval instructions.

4.4.4 Current Status

The ActiveMath learning environment is alpha status of development. Most of the basic features are becoming stable and new ones are being planned. Authoring tools are under development but usage of QMath (see other implementations) is recommended and compatible.

More information and a demo version of ActiveMath can be found from our web-page `http://www.activemath.org/`.

Chapter 5

Conclusion

With OMDoc we have proposed a content-based markup format that allows to represent mathematical knowledge at various levels. As a consequence the format allows to capture the semantics and structure of various kinds of mathematical documents, including articles, textbooks, interactive books, and courses.

We have argued that the problem of representing mathematical knowledge has to be addressed at three levels, corresponding to the three levels of structure found in documents.

The formula level is concerned with representing mathematical objects as mathematical/logical formulae. OMDoc leverages the existing OpenMath and MathMl standards for this.

The statement level consists of statements about the mathematical objects, like definitions, theorems and proofs. On this level, OMDoc supplies original markup schemes that allow structured representations of the mathematical content (including both formal and informal elements of representation).

The theory level allows to group statements to conceptual units according to the assumptions on the mathematical objects they describe. An inheritance mechanism allows to specify the acessibility and scoping of symbols, and re-use flexibly parts of specifications. The theory level even allows to structure collections of theories by theory-inclusions and transport theories and proof methods along these relations.

We have motivated and described version 1.1 of the OMDoc language and presented an Xml document type definition for it. We have surveyed a set of transformation tools that generate presentation-oriented documents for human consumption and machine-oriented documents for communication with mathematical software systems.

We have developed first authoring tools for OMDoc that try to simplify generating OMDoc documents for the working mathematician. There is a simple OMDoc mode for `emacs`, and a LaTeX style [Koh00a] that can be used to generate OMDoc representations from LaTeX sources and thus help migrate existing mathematical documents. A second step will be to integrate the LaTeX to OpenMath conversion tools.

The next steps in the development will be to develop OMDoc version 2.0 including more disruptive changes to the language, including a re-organization of central OMDoc elements like `definition`.

Bibliography

[ABI+96] Peter B. Andrews, Matthew Bishop, Sunil Issar, Dan Nesmith, Frank Pfenning, and Hongwei Xi. TPS: A theorem-proving system for classical type theory. *Journal of Automated Reasoning*, 16:321–353, 1996.

[AZ00] Alessandro Armando and Daniele Zini. Towards Interoperable Mechanized Reasoning Systems: the Logic Broker Architecture. In A. Poggi, ed., *to appear on the Proceedings of the AI*IA-TABOO Joint Workshop 'From Objects to Agents: Evolutionary Trends of Software Systems'*, Parma, Italy, May 29–30, 2000.

[Bar92] Henk P. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, eds., *Handbook of Logic in Computer Science*, Vol. 2, pp.117–309. Oxford University Press, 1992.

[Bau99] Judith Baur. Syntax und Semantik mathematischer Texte — ein Prototyp. Master Thesis, Saarland University, 1999.

[BBS99] Christoph Benzmüller, Matthew Bishop, and Volker Sorge. Integrating TPS and ΩMEGA. *Journal of Universal Computer Science*, 5(2), 1999.

[BCF+97] C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. ΩMEGA: Towards a mathematical assistant. In William McCune, ed., *Proceedings of the 14th Conference on Automated Deduction*, no.1249 in LNAI, pp.252–255, Townsville, Australia, 1997. Springer Verlag.

[BLFM98] Tim Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifiers (uri), generic syntax. RFC 2717, Internt Engineering Task Force, 1998. available at `http://www.ietf.org/rfc/rfc2717.txt`.

[Bos98] Cascading style sheets, level 2; css2 specification. W3c recommendation, World Wide Web Consortium (W3C), 1998. available as `http://www.w3.org/TR/1998/REC-CSS2-19980512`.

[BPSM97] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML). W3C Recommendation TR-XML, World Wide Web Consortium, December 1997. Available at `http://www.w3.org/TR/PR-xml.html`.

[BSBG98] R. Boulton, K. Slind, A. Bundy, and M. Gordon. An interface between CLAM and HOL. In Jim Grundy and Malcolm Newey, eds., *Theorem Proving in Higher Order Logics: Emerging Trends*, Technical Report 98-08, Department of Computer Science and Computer Science Lab, pp.87–104, Canberra, Australia, October 1998. The Australian National University.

[CAB+86] Robert L. Constable, S. Allen, H. Bromly, W. Cleaveland, J. Cremer, R. Harper, D. Howe, T. Knoblock, N. Mendler, P. Panangaden, J. Sasaki, and S. Smith. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[CC98] Olga Caprotti and Arjeh M. Cohen. Draft of the Open Math standard. The Open Math Society, `http://www.nag.co.uk/projects/OpenMath/omstd/`, 1998.

[CIMP01] David Carlisle, Patrick Ion, Robert Miner, and Nico Poppelier. Mathematical Markup Language (MathML) version 2.0. W3c recommendation, World Wide Web Consortium, 2001. Available at http://www.w3.org/TR/MathML2.

[Cla99] Xml path language (xpath) version 1.0. W3c recommendation, The World Wide Web Consortium, 1999. available at `http://www.w3.org/TR/xpath`.

[CoF98] Language Design Task Group CoFI. Casl — the CoFI algebraic specification language — summary, version 1.0. Tech. rep., `http://www.brics.dk/Projects/CoFI`, 1998.

[dB80] Nicolaas Govert de Bruijn. A survey of the project AUTOMATH. In R. Hindley and J. Seldin, eds., *To H.B. Curry: Essays in Combinator Logic, Lambda Calculus and Formalisms*, pp.579–606. Academic Press, 1980.

[DCN+00] Louise A. Dennis, Graham Collins, Michael Norrish, Richard Boulton, Konrad Slind, Graham Robinson, Mike Gordon, and Tom Melham. The prosper toolkit. In *Proceedings of the Sixth International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS-2000*, LNCS, Berlin, Germany, 2000. Springer Verlag.

[Dea99] Stephen Deach. Extensible stylesheet language (xsl) specification. W3c working draft, W3C, 1999. Available at `http://www.w3.org/TR/WD-xsl`.

[DJM01] Steve DeRose, Ron Daniel Jr., and Eve Maler. Xml pointer language (XPointer). W3c candidate recommendation, W3C, 2001. Available at `http://www.w3.org/TR/xptr`.

[DuC97] Bob DuCharme. Formatting documents with dsssl specifications and jade. *The SGML Newsletter*, 10(5):6–10, 1997.

[Duc98] Denys Duchier. The NEGRA tree bank. Private communication, 1998.

[Far93] William M. Farmer. Theory interpretation in simple type theory. In *HOA'93, an International Workshop on Higher-order Algebra, Logic and Term Rewriting*, Vol.816 of *LNCS*, Amsterdam, The Netherlands, 1993. Springer Verlag.

[FGT93] William M. Farmer, Joshua D. Guttman, and F. Javier Thayer. IMPS: An Interactive Mathematical Proof System. *Journal of Automated Reasoning*, 11(2):213–248, October 1993.

[FH01] Armin Fiedler and Helmut Horacek. Argumentation in explanations to logical problems. In Vassil N. Alexandrov, Jack J. Dongarra, Benjoe A. Juliano, Renè S. Renner, and C. J. Kenneth Tan, eds., *Computational Science — ICCS 2001*, no.2074 in LNCS, pp.969–978, San Francisco, CA, 2001. Springer Verlag.

[FHJ+99] Andreas Franke, Stephan M. Hess, Christoph G. Jung, Michael Kohlhase, and Volker Sorge. Agent-oriented integration of distributed mathematical services. *Journal of Universal Computer Science*, 5:156–187, 1999.

[Fie97] Armin Fiedler. Towards a proof explainer. In Siekmann et al. [SPH97], pp.53–54.

[Fie99] Armin Fiedler. Using a cognitive architecture to plan dialogs for the adaptive explanation of proofs. In Thomas Dean, ed., *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pp.358–363, Stockholm, Sweden, 1999. Morgan Kaufmann.

[Fie01] Armin Fiedler. Dialog-driven adaptation of explanations of proofs. In Bernhard Nebel, ed., *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pp.1295–1300, Seattle, WA, 2001. Morgan Kaufmann.

[FK99] Andreas Franke and Michael Kohlhase. System description: MathWeb, an agent-based communication layer for distributed automated theorem proving. In Harald Ganzinger, ed., *Automated Deduction — CADE-16*, no.1632 in LNAI, pp.217–221. Springer Verlag, 1999.

[FK00] Andreas Franke and Michael Kohlhase. System description: MBase, an open mathematical knowledge base. In David McAllester, ed., *Automated Deduction – CADE-17*, no.1831 in LNAI, pp.455–459. Springer Verlag, 2000.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen I & II. *Mathematische Zeitschrift*, 39:176–210, 572–595, 1935.

[GM93] M. J. C. Gordon and T. F. Melham. *Introduction to HOL – A theorem proving environment for higher order logic*. Cambridge University Press, 1993.

[Gol90] C. F. Goldfarb. *The SGML Handbook*. Oxford University Press, 1990.

[Gro99] The Open eBook Group. Open ebook[tm] publication structure 1.0. Draft recommendation, The OpenEBook Initiative, 1999. Available at `http://www.openEbook.org`.

[Har01] Eliotte Rusty Harold. *XML Bible*. Hungry Minds, gold edition ed., 2001.

[HF96] Xiaorong Huang and Armin Fiedler. Presenting machine-found proofs. In McRobbie and Slaney [MS96], pp.221–225.

[HF97] Xiaorong Huang and Armin Fiedler. Proof verbalization in *PROVERB*. In Siekmann et al. [SPH97], pp.35–36.

[HS96] Dieter Hutter and Claus Sengler. INKA - The Next Generation. In McRobbie and Slaney [MS96], pp.288–292.

[Hut00] Dieter Hutter. Management of change in structured verification. In *Proceedings Automated Software Engineering (ASE-2000)*. IEEE Press, 2000.

[KF00] Michael Kohlhase and Andreas Franke. Mbase: Representing knowledge and context for the integration of mathematical software systems. *Journal of Symbolic Comutation; Special Issue on the Integration of Computer algebra and Deduction Systems*, 2000. forthcoming.

[Knu84] Donald E. Knuth. *The TEXbook*. Addison Wesley, 1984.

[Koh00a] Michael Kohlhase. Creating OMDOC representations from LATEX. Internet Draft available at `http://www.mathweb.org/omdoc`, 2000.

[Koh00b] Michael Kohlhase. OMDoc: An infrastructure for OPENMATH content dictionary information. *Bulletin of the ACM Special Interest Group on Symbolic and Automated Mathematics (SIGSAM)*, 34(2):43–48, 2000.

[Koh00c] Michael Kohlhase. OMDoc: An open markup format for mathematical documents. Seki Report SR-00-02, Fachbereich Informatik, Universität des Saarlandes, 2000. `http://www.mathweb.org/omdoc`.

[Lam94] Leslie Lamport. *LATEX: A Document Preparation System, 2/e*. Addison Wesley, 1994.

[Lee98] Tim Berner's Lee. The semantic web. W3C Architecture Note, 1998. http://www.w3.org/DesignIssues/Semantic.html.

[MAF+01] E. Melis, J. Buedenbender E. Andres, Adrian Frischauf, G. Goguadze, P. Libbrecht, M. Pollet, and C. Ullrich. The ActiveMath learning environment. *Artificial Intelligence and Education*, 12(4), winter 2001 2001.

[MAH01] Till Mossakowski, Serge Autexier, and Dieter Hutter. Extending development graphs with hiding. In H. Hußmann, ed., *Proceedings of Fundamental Approaches to Software Engineering (FASE 2001)*, 2001.

[mar01] Xml base. W3c recommendation, The World Wide Web Consortium, 2001. available at `http://www.w3.org/TR/xmlbase/`.

[MS96] M.A. McRobbie and J.K. Slaney, eds.. *Proceedings of the 13th Conference on Automated Deduction*, no.1104 in LNAI, New Brunswick, NJ, USA, 1996. Springer Verlag.

[MSLK01] M. Murata, S. St. Laurent, and D. Kohn. Xml media types. RFC 3023, January 2001. `ftp://ftp.isi.edu/in-notes/rfc3023.txt`.

[ORS92] S. Owre, J. M. Rushby, and N. Shankar. PVS: a prototype verification system. In D. Kapur, ed., *Proceedings of the 11th Conference on Automated Deduction*, Vol.607 of *LNCS*, pp.748–752, Saratoga Spings, NY, USA, 1992. Springer Verlag.

[Pfe91] Frank Pfenning. Logic programming in the LF logical framework. In Gérard P. Huet and Gordon D. Plotkin, eds., *Logical Frameworks*. Cambridge University Press, 1991.

[PN90] Lawrence C. Paulson and Tobias Nipkow. Isabelle tutorial and user's manual. Tech. rep.189, Computer Laboratory, University of Cambridge, January 1990.

[Rei87] Glenn C. Reid. *PostScript, Language, Program Design.* Addison Wesley, 1987.

[RHJ98] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.0 Specification. W3C Recommendation REC-html40, World Wide Web Consortium, April 1998. Available at `http://www.w3.org/TR/PR-xml.html`.

[RSG98] Julian D.C. Richardson, Alan Smaill, and Ian M. Green. System description: Proof planning in higher-order logic with λ*clam*. In Claude Kirchner and Hélène Kirchner, eds., *Proceedings of the 15th Conference on Automated Deduction*, no.1421 in LNAI. Springer Verlag, 1998.

[Rud92] Piotr Rudnicki. An overview of the mizar project. In *Proceedingsof the 1992 Workshop on Types and Proofs as Programs*, pp.311–332, 1992.

[SBC+00] Jörg Siekmann, Christoph Benzmüller, Lassaad Cheikhrouhou, Armin Fiedler, Andreas Franke, Helmut Horacek, Michael Kohlhase, Andreas Meier, Erica Melis, Martin Pollet, Volker Sorge, Carsten Ullrich, and Jürgen Zimmer. Adaptive course generation and presentation. In P. Brusilovski, ed., *Proceedings of ITS-2000 workshop on Adaptive and Intelligent Web-Based Education Systems*, Montreal, 2000.

[Smo95] G. Smolka. The Oz programming model. In Jan van Leeuwen, ed., *Computer Science Today*, Vol.1000 of *LNCS*, pp.324–343. Springer-Verlag, Berlin, 1995.

[SPH97] J. Siekmann, F. Pfenning, and X. Huang, eds.. *Proceedings of the First International Workshop on Proof Transformation and Presentation*, Schloss Dagstuhl, Germany, 1997.

[SSBL01] Colin Smythe, Eric Shepherd, Lane Brewer, and Steve Lay. Ims question & test interoperability: An overview. Public Draft Version 1.2, IMS Global Learning Consortium, Inc., 2001. available at `http://www.imsglobal.org/question/qtiv1p2pd/imsqti_oviewv1p2.html`.

[Tea] Coq Development Team. *The Coq Proof Assistant Reference Manual*. INRIA. see `http://coq.inria.fr/doc/main.html`.

[Tho91] Simon Thompson. *Type Theory and Functional Programming*. International Computer Science Series. Addison-Wesley, 1991.

Index

Appendix A

Errata to the released Specification

Here we track the errata in the OMDoc 1.1 specification (see `http://www.mathweb.org/omdoc/a`
These errata have been cleared in this version.

1 Introduction

No errata known.

2 Mathematical Markup Schemes

No errata known.

3 OMDoc Elements

No errata known.

3.1 Metadata for Mathematical Elements

No errata known.

3.2 Mathematical Statements

3.2.1 Specifying Mathematical Properties

- The text fails to make clear for the `id`/`xref` to OpenMath objects are only allowed, if the referencing element has the same name as the referenced one. In particular, there are no implicit conversions.

- The text fails to mention the logic attribute of the FMP element. We need to add "FMPs always appear in groups, which can differ in the value of their logic attribute, which specifies the logical formalism. The value of this attribute specifies the logical system used in formalizing the content. All members of the multi-logic FMP group have to formalize the same mathematical object or property, i.e. they have to be translations of each other."

3.2.2 Symbols, Definitions, and Axioms

1. The values 'obj' and 'simple' were overlapping, and the role of the FMP and OMOBJ children of the definition was unclear. The value 'obj' has been dropped and we have clarified that in simple definitions, the OMOBJ is the substitution element whereas the FMP captures the meaning of the CMP group in Logic.

2. The attribute kind of the symbol element can also have the value 'sort' for sets that are inductively built up from constructor symbols

3.2.3 Assertions and Alternatives

No errata known.

3.2.4 Mathematical Examples in OMDoc

No errata known.

3.2.5 Representing Proofs in OMDoc

No errata known.

3.2.6 Abstract Data Types

The optional recognizer element should be a child of sortdef, and not of the constructor element. The specification text and examples are correct, but the quick reference table is incorrect.

3.3 Theories as Mathematical Contexts

3.3.1 Simple Inheritance

The content model for `theory` in Figure 3.22 is incorrect. It should read `commonname*, CMP*, (statement | inclusion, imports)*`. Moreover, the attribute model has a spurious comma. Furthermore, the text should make clear that OMDoc1.1 does not allow theories to nest and that theories can include `imports` statements.

3.3.2 Inheritance via Translations

No errata known.

3.3.3 Statements about Theories

The text does not make this clear, but the elements `theory-inclusion`, `axiom-inclusion` and `decomposition` may not occur in a `theory` element. Worse, the document type definition allows this as well.

3.3.4 Parametric theories in OMDoc

No errata known.

3.4 Auxiliary Elements

3.4.1 Preservation of Text Structure

Figure 3.29: Specifying Tables with `<omgroup type="dataset">` The first label for the second axis is "`b11`". Should be "`b1`".

omgroup The DTD did not contain value `'narrative'` that was present in the specification.

3.4.4 Exercises

In the `solution` element, where `proof` was allowed, we have to allow `proofobject` as well.

3.5 Adding Presentation Information to OMDoc

1. It should be made clear that the `xml:lang` attribute of the `use`, `xslt` and `style` elements does not have the default value `en`.

2. OMDoc1.1 uses the `style` attribute for all elements that have an `id` attribute to specify generic style classes for the OMDoc elements. This is based on a misunderstanding of the Xml cascading style sheet (Css) mechanism [Bos98], which uses the `class` attribute to specify this information and uses the `style` attribute to specify Css directives that override the class information.

   Even though this is a grave error (it severely limits the usefulness) we will not change it in the OMDoc 1.1 specification and wait for the release of OMDoc 1.2 to fix this, the renaming of the `style` attribute to `class` would break existing implementations.

3.5.2 Specifying the Notation of Mathematical Symbols

Figure 3.44 the example still uses the OMDoc1.0 version of specifying XslT content via the `system` attribute, in OMDoc1.1 the element `xslt` should be used.

3.6 Identifying and Referencing OMDoc Elements

Locating `OMS` elements by the OMDoc Catalogue

No errata known.

A URI-based Mechanism for Element Reference

The text does not make it clear that the namespace prefixes for theory collections can be declared in any element that dominates the referencing element. The DTD does not allow this either. We will not change this in the DTD, since the changes are too disruptive and OMDoc1.2 is coming up soon.

Uniqueness Constraints and Relative URI references

No errata known.

4 OMDoc Applications, Tools, and Projects

No errata known.

B Changes

No errata known.

C Quick-Reference for OMDoc Elements

No errata known.

D Quick-Reference for OMDoc Attributes

No errata known.

E OMDoc DTD

We use the following public identifier for DTDs: `-//OMDoc//DTD OMDoc V1.1//EN`

1. The `xml:lang` attribute of the `use`, `xslt` and `style` elements should not have the default value `en`.
2. The elements `theory-inclusion`, `axiom-inclusion` and `decomposition` may not occur in a `theory` element.
3. The content model for `solution` should make the `FMP`, `proof`, and `proofobjects` elements optional.
4. the attribute `for` should have been optional
5. the optional `recognizer` element should be a child of `sortdef`, and not of the `constructor` element.
6. the `exercise` element should allow multiple mathematical objects, not at most one.
7. the `%cfm;` parameter entity in the DTD did not allow for multiple FMPs, even though the spefication says that they appear in multi-logic groups.
8. the default `precedence` attribute of the `presentation` element should have the default value 1000.

9. the content model of the `definition` element had to be adapted to the clarification in the specification. The old version only allowed `CMP`-only content with `rxp`. The value `'obj'` has also been dropped.

10. the content model for the `omdoc` element prescribed at least one omdoc item. This is not intended, since we want to allow catalogue-only documents for administrative purposes.

11. the theory attribute for the `assertion` `alernative`, `proof` and `proofobject` elements should be of type `CDATA`, after all it contains a URI that points to an external theory.

12. the specification mentions type `'comment'`, but the DTD did not allow this.

Appendix B

Changes from Version 1.0

In this section we will keep a log on the changes that have occurred in the released versions of the OMDoc format. We will briefly tabulate the changes by element name. For the state of an element we will use the shorthands "dep" for deprecated (i.e. the element is no longer in use in the new OMDoc version), "cha" for changed, if the element is re-structured (i.e. some additions and losses), "new" if did not exist in the old OMDoc version, and finally "aug" for augmented, i.e. if it has obtained additional children or attributes in the new OMDoc version.

Version 1.1 is mainly a bug-fix release that has become necessary by the experiments of encoding legacy material in OMDoc. The changes are relatively minor, mostly added optional fields. The only non-conservative changes concern the `private`, `hypothesis`, `sortdef` and `signature` elements. OMDoc files can be upgraded to version 1.1 with the XslT style sheet `http://www.mathweb.org/omdoc/xsl/omdoc1.0adapt1.1.xsl`.

element	state	comments	cf.
`attribute`	new	presentation of attributes for Xml elements	78
`alternative`	cha	new form of the `alternative-def` element, it can now also used as an alternative to `axiom`. Compared to `alternative-def` it has a new optional attribute `generated-by` to show that an assertion is generated by expanding a some other element like `adt`.	40
`alternative-def`	dep	new form is `alternative`, since there can be alternative `axiom`s too.	
`argument`	cha	attribute `sort` is now of type IDREF, since it must be local in the definition.	51

assertion	aug	more values for the type, new optional attribute generated-by to show that an assertion is generated by expanding a definition or an adt. New optional attribute proofs.	38
assertion-just	dep	this is now obligation	
axiom	aug	new optional attribute generated-by to show that an axiom is generated by expanding a definition.	36
axiom-inclusion	cha	now allows a CMP group for descriptive text, includes a set of obligations instead of an assertion-just. The timestamp attribute is deprecated, use dc:Date with appropriate action instead	58
CMP	cha	the attribute format is now deprecated, it makes no sense, since we are more strict and consistent about CMP content.	32
code	cha	Attributes width and height now in omlet, got attributes classid and codebase from private. Attribute format moved to data children. The multilingual group of CMP elements for description is deprecated, use metadata/Description instead. Child element data may appear multiple times (with different values of the format).	70
constructor	aug	new optional child recognizer for a recognizer predicate	51
Coverage	dep	this Dublin Core element specifies the place or time which the publication's contents addresses. This does not seem appropriate for the mathematical content of OMDoc.	
data	aug	new optional attributes size to specify the size of the data file that is referenced by the href attribute and format for the format the data is in.	70
dc:*	aug	Contributor, Creator, Publisher have received an optional id attribute, so that they can be cross-referenced by the new who of the Date element.	24
dc:Date	aug	new optional who attribute that can be used to specify who did the action on this date.	26
dc:Translator	dep	this element is not part of Dublin Core, it got into OMDoc by mistake, we use Contributor with role=trl for this.	25
decomposition	aug	has a new required id attribute. It is no longer a child of theory-inclusion, but specifies which theory-inclusion it justifies by the new required attribute for.	60
definition	aug	new optional children measure and ordering to specify termination of recursive definitions. New optional attribute generated-by to show that it is generated by expanding a definition.	36

element	new	presentation of XML elements	77
FMP	aug	now allows multiple conclusion elements, to represent general Gentzen-type sequents (not only natural deduction.)	31
hypothesis	cha	new **required** attribute discharged-in to specify the derive or conclude element that discharges this hypothesis.	47
measure	new	specifies a measure function (as an OMOBJ)	37
metadata	aug	new optional attribute inherits that allows to inherit metadata from other declarations	24
method	cha	first child that used to be an OMSTR or ref element is now moved into a required xref attribute that holds an URI that points to the element that defines the method. The OMOBJ content of the other children (they were paramter elements) is now directly included in the method element.	49
obligation	new	takes over the role of assertion-just.	
omgroup	aug	also allows the elements that can only appear in theory elements, so that omgroups can also be used for grouping inside theory elements. The type attribute is now restrained to one of 'narrative', 'sequence', 'alternative', 'contrast'.	65
omlet	aug	obtained attributes width and height from private. New optional attributes action for the action to be taken when activated, and data a URIref to data in a private element. New optional attribute type for the type of the applet.	71
omstyle	new	for specifying the style of OMDoc elements	75
omtext	cha	the from is deprecated, we only leave the for attribute, to specify the referential character of the type.	34
ordering	new	specifies a well-founded ordering (as an OMOBJ)	37
parameter	dep	the OMOBJ element child is now directly a child of method	
pattern	cha	the child can be an arbitraryOpenMath element.	37
premise	cha	new optional attribute rank for the importance in the inference rule. The old href attribute is renamed to xref to be consistent with other crossreferening.	
presentation	aug	id attribute is now optional. new attribute xref that allows to inherit the information from another presentation element. New attribute theory to specify the theory the symbol is from; without this, referencing in OMDoc is not unique.	80

`private`	cha	new optional attribute `for` to point to an OMDoc element it provides data for. As a consequence, `private` elements are no longer allowed in other OMDoc elements, only on top-level. New attribute `replaces` as a pointer to the OMDoc elements that are replaced by the system-specific information in this element. Old attributes `width` and `height` now in `omlet`. Attribute `format` moved to `data` children. The multilingual group of `CMP` elements for description is deprecated, use `metadata/Description` instead. Child element `data` may appear multiple times (with different values of the `format`). The attributes `classid` and `codebase` are deprecated, since they only make sense on the `code` element.	70
`proof`,`proofobject`	cha	attribute `theory` is now optional, since it can appear in a `theory`.	46
`recognizer`	new	specifies the recognizer predicate of a sort.	51
`recurse`	new	recursive calls to presentation in `style`.	77
`ref`	cha	attribute `kind` renamed to `type`.	65
`selector`	cha	the old `type` attribute (had values `total` and `partial`) is deprecated, its duty is now carried by an attribute `total` (values `'yes'` and `'no'`).	51
`signature`	dep	for the moment	
`sortdef`	cha	attribute `id` is now mandatory, otherwise the defined symbol no name. The `kind` that was fixed to `sort` is deprecated, this piece of information is redundant.	51
`style`	new	allows to specify style information in `presentation` and `omstyle` elements using a simplified OMDoc-internalized version of XSLT.	76
`symbol`	aug	new optional attribute `generated-by` to show that it is generated by expanding a `definition`.	35
`text`	new	presentation of text in `omstyle`.	77
`theory-inclusion`	cha	now allows `CMP` group for descriptive text, no longer has a `decomposition` child, this is now attatched by its `for` attribute. The `timestamp` attribute is deprecated, use `dc:Date` with appropriate `action` instead.	58
`type`	aug	can now also appear on top-level. Has an optional `id` attribute for identification, and an optional `for` attribute to point to a `symbol` element it declares type information for.	35

`use`	aug	New attribute `element` allows to specify that the content should be encased in an XML element with the attribute-value pairs specified in the string specified in the attribute `attributes`.	80
`value-of`	new	presentation of values in `style`.	78
`with`	new	used to supply fragements of text in `CMP`s with `id` and `id` attributes that can be used for presentation and referencing.	32
`xslt`	new	allows to embed XSLT into `presentation` and `omstyle` elements.	76

Appendix C

Quick-Reference Table to the OMDoc Elements

Element	p.	Type	Required Attribs	Optional Attribs	DC	Content
adt	50	adt	id	type, style	+	CMP*, commonname*, sortdef+
alternative	40	stat	id, for, theory, entailed-by, entails, entailed-by-thm, entails-thm	type, generated-by, just-by, style	+	CMP*, (FMP| requation*| OMOBJ)
answer	73	ex	verdict	id, style	+	symbol*,CMP*,FMP*
argument	51	adt	sort		+	selector?
assertion	38	stat	id	type, theory, generated-by, style	+	symbol*,CMP*,FMP*
assumption	32	stat	id	style	+	CMP*, OMOBJ?
attribute	78	pres	name		−	(#PCDATA| value-of| text)*
axiom	36	thy	id	generated-by, style	+	symbol*,CMP*,FMP*
axiom-inclusion	58	thy	id, from, to	style	+	morphism?, (path-just| obligation*)
choice	73	ex		id, style	+	symbol*,CMP*,FMP*
CMP	32	stat		xml:lang	−	(text| OMOBJ| with| omlet)*

code	70	aux	id, theory	id, for, theory, pto, pto-version, format, requires, type, classid, codebase, width, height, style	+	CMP*, input?, output?, effect?, data+
commonname	35	thy		xml:lang	−	CMPcontent
conclude	47	prf	id	style	−	CMP*, method?, premise*, (proof | proofobject)?
conclusion	32	stat	id	style	+	CMP*, OMOBJ?
constructor	51	adt	id	type, scope, style	+	commonname*, argument*, recognizer?
Contributor	25	meta		id, role, style	−	%DCperson
Creator	25	meta		id, role, style	−	%DCperson
data	70	aux		format, href, size	−	<![CDATA[...]]>
Date	26	meta		action, who	−	ISO8601
decomposition	60	thy	links		−	EMPTY
definition	36	thy	id, for	just-by, type, generated-by, style	+	CMP*, (FMP| requation+| OMOBJ)?, measure?, ordering?
Description	25	meta		xml:lang	−	CMPcontent
derive	46	prf	id	style	−	CMP*, FMP?, method?, premise*, (proof | proofobject)?
effect	70	aux			−	CMP*
element	77	pres	name		−	(attribute| element| text| recurse)*
example	42	stat	id, for	type, assertion, proof, style	+	symbol*, CMP*| OMOBJ?
exercise	73	ex	id	type, for, from, style	+	symbol*,CMP*,FMP*, hint?, (solution*|mc*)
extradata	24	meta			−	ANY
FMP	31	stat		logic	−	(assumption*, conclusion*)|OMOBJ
Format	26	meta			−	fixed:"xml, x-omdoc"
hint	73	ex		id, style	+	symbol*,CMP*,FMP*
hypothesis	47	prf	id, discharged-in, style		−	symbol*,CMP*,FMP*
Identifier	27	meta		scheme	−	ANY
ignore	34	aux		type, comment	−	ANY

imports	54	thy	id, from	type, hiding, style	–	CMP*, morphism?
inclusion	63	thy	for		–	
input	70	aux			–	CMP*
insort	51	adt	for		–	
Language	27	meta			–	ISO8601
mc	73	ex		id, style	–	symbol*, choice, hint?, answer
measure	37	thy			–	OMOBJ
metacomment	47	prf		id, style	–	CMP*
metadata	24	meta		inherits	–	(dc-element)*, extradata
method	49	prf	xref		–	OMOBJ*
morphism	56	thy		id, base, style	–	requation*
obligation	59	thy	induced-by, assertion		–	EMPTY
omdoc	23	struct	id	type, version, style, xmlns, catalogue, xmlns:xsi, xsl:schemaLocation	+	(OMDoc element)*
omgroup	65	struct	id	type, for, from, style	+	OMDocelement*
omlet	71	aux		id, argstr, type, function, action, data, style	+	ANY
omstyle	75	pres	element	for, id, xref, style	–	(style|xslt)*
omtext	34	struct	id	type, for, from, style	+	CMP+, FMP?
ordering	37	thy			–	OMOBJ
output	70	aux			–	CMP*
path-just	59	thy	local, globals		–	EMPTY
pattern	37	thy			–	OMOBJ
premise	49	prf	xref		–	EMPTY
presentation	80	pres	for	id, xref, fixity, parent, lbrack, rbrack, separator, bracket-style, style, precedence, crossref-symbol, theory	–	(use | xslt | style)*

private	70	aux		id, for, theory, pto, pto-version, format, requires, type, classid, codebase, width, height, replaces, style	+	CMP*, data+
proof	46	prf	id, for, theory	style	+	symbol*, CMP*, (metacomment| derive| hypothesis)*, conclude
proofobject	44	prf	id, for, theory	style	+	CMP*, OMOBJ
Publisher	26	meta		id, style	−	ANY
ref	65	struct		xref, type	−	ANY
recognizer	51	adt	id	type, scope, kind, style	−	commonname*
recurse	77	pres		select	−	EMPTY
Relation	27	meta			−	ANY
requation	37	thy		id, style	−	pattern, value
Rights	27	meta			−	ANY
selector	51	adt	id	type, scope, kind, total, style	−	commonname*
solution	73	ex		id, for, style	+	(symbol*,CMP*,FMP*) | proof
sortdef	51	adt	id	kind, scope, style	−	commonname*, (constructor|insort)*
Source	27	meta			−	ANY
style	76	pres	format	xml:lang, requires	−	(element | text | recurse | value-of)*
Subject	25	meta		xml:lang	−	CMPcontent
symbol	35	thy	id	kind, scope, style	+	CMP*, (commonname| type| selector)*
text	77	pres			−	(#PCDATA)
theory	52	thy	id	style	+	commonname*, statement*
theory-inclusion	58	thy	id, from, to, by, style		+	(morphism, decomposition?)
Title	24	meta		xml:lang	−	CMPcontent
type	35	thy	system	id, for, style	−	CMP*, OMOBJ
Type	26	meta			−	fixed:"Dataset"or"Text"

use	80	pres	format	xml:lang, requires, larg-group, rarg-group, fixity, lbrack, rbrack, separator, crossref-symbol, element, attributes	–	(use | xslt | style)*
value	37	thy			–	OMOBJ
value-of	78	pres	select		–	EMPTY
with	32	stat	id	style	–	CMP content
xslt	76	pres	format	xml:lang, requires	–	CDATA

Appendix D

Quick-Reference Table to the OMDoc Attributes

Attribute	*element*	Values
action	omlet	
	specifies the action to be taken when executing the omlet, the value is application-defined.	
argstring	omlet	
	specifies the argument string for the function specified in the function attribute of this omlet	
assertion	example	
	specifies the assertion that states that the objects given in the example really have the expected properties.	
assertion	obligation	
	specifies the assertion that states that the translation of the statement in the source theory specified by the induced-by attribute is valid in the target theory.	
attibutes	use	
	the attribute string for the start tag of the XML element substituted for the brackets (this is specified in the element attribute).	
base	morphism	
	specifies another morphism that should be used as a base for expansion in the definition of this morphism	
bracket-style	presentation, use	lisp, math
	specifies whether a function application is of the form $f(a,b)$ or (fab)	
catalogue	omdoc	
	specifies an outside OMDoc document that contais catalogue information for this one.	
lbrack	presentation, use	

	the left bracket to use in the notation of a function symbol	
cd	loc	
	specifies the location of the content dictionary for a theory	
classid, codebase	code	
	points to a class identifier and codebase, if the code contains Java.	
comment	ignore	
	specifies a reason why we want to ignore the contents	
crossref-symbol	presentation, use	all, brackets, lbrack, no, rbrack, separator, yes
	specifies whether crossreferences to the symbol definition should be generated in the output format.	
data	omlet	
	points to a private element that contains the data for this omlet	
discharged-in	hypothesis	
	specifies the scope of a local hypothesis in a proof. It points to the proof step which discharges it.	
element	use	
	the XML element tags to be substituted for the brackets.	
entails, entailed-by	alternative	
	specifies the equivalent formulations of a definition or axiom	
entails-thm, entailed-by-thm	alternative	
	specifies the entailsment statements for equivalent formulations of a definition or axiom	
fixity	presentation	assoc, infix, postfix, prefix
	specifies where the function symbolof a function application should be displayed in the output format	
function	omlet	
	specifies the function to be called when this omlet is activated.	
format	data	
	specifies the format of the data specified by a data element. The value should e.g. be a MIME type.	
generated-by	symbol, axiom, assertion, definition, alternative	
	points to a higher-level syntax element, that generates this statement.	
for	*	
	can be used to reference an element by its unique identifier given in its id attribute.	

format	use	cmml, default, html, mathematica, pmml, TeX,
	specifies the output format for which the notation is specified	
globals	path-just	
	points to the theory-inclusions that is the rest of the inclusion path.	
height	omlet	
	specifies the height of the rectangle on the screen taken up by the results of an omlet	
hiding	imports	
	specifies the names of symbols that are not imported from the source theory	
href	data	
	a URI to an external file containig the data.	
id		
	associates a unique identifier to an element, which can thus be referenced by an for attribute.	
induced-by	obligation	
	points to the statement in the source theory that induces this proof obligation	
just-by	definition, alternative	
	points to an assertion that states the well-definedness or termination condition of a definition or the equivalence condition of an alternative definition.	
kind	symbol	object, sort, type
	specifies the kind of object defined in this declaration.	
links	decomposition	
	specifies a list of theory- or axiom-inclusions that justify (by decomposition) the theory-inclusion specified in the for attribute.	
local	path-just	
	points to the axiom-inclusion that is the first element in the path.	
logic	FMP	token
	specifies the logical system used to encode the property.	
name	OMS, OMV	
	the name of a symbol or variable.	
omdoc	loc	
	specifies the location of the OMDoc document containing the theory.	
omdoc-element	presentation	
	specifies the OMDoc element the presentation information applies to.	

parent	presentation	OMA, OMATTR,OMBIND
	specifies the parent element of the symbol for which notation information is specified	
precedence	presentation	
	the precedence of a function symbol (for elision of brackets)	
proofs	assertion	
	specifies a list of URIs to proofs of this assertion.	
pto, pto-version	private, code	
	specifies the system and its version this data or code is private to	
replaces	private	
	points to a set of elements whose content is replaced by the content of the `private` element for the system.	
requires	private, code, use	
	points to a `code` element that is needed for the execution of this data by the system.	
rbrack	presentation, use	
	the right bracket to use in the notation of a function symbol	
role	Creator, Collaborator	aft, ant, aqt, aui, aut, clb, edt, ths, trc, trl
	the MARC relator code for the contribution of the individual.	
size	data	
	specifies the size the data specified by a `data` element. The value should be number of kilobytes	
scope	symbol	global, local
	specifies the visibility of the symbol declared. This is a very crude specification, it is better to use theories and importing to specify symbol accessibility.	
separator	presentation, use	
	the separator for the arguments to use in the notation of a function symbol	
sort	argument	
	specifies the argument sort of the constructor	
style	*	
	specifies a token for a presentation style to be picked up in a `presentation` element.	
system	use	pres, xsl
	The transformation system to be used for specification. Use `'pres'` for the OMDoc metalanguage, and `'xsl'` for straight XSLT.	
system	type	
	A token that specifies the logical type system that governs the type specified in the type element.	

theory	*	
	specifies the home theory of an OMDoc statement.	
theory	loc	
	specifies the theory the loc element locates	
to	theory-inclusion, axiom-inclusion	
	specifies the target theory	
total	selector	no, yes
	specifies whether the symbol declared here is a total or partial function.	
type	adt	free, generated, loose
	defines the semantics of an abstract data type free = no junk, no confusion, generated = no junk, loose is the general case.	
type	asssertion	theorem, lemma, corollary, conjecture, false-conjecture, obligation, postulate, formula, assumption, proposition
	tells you more about the intention of the assertion	
type	definition	implicit, inductive, obj, recursive, simple
	specifies the definition principle	
type	example	against, for
	specifies whether the objects in this example support or falsify some conjecture	
type	imports	global, local
	local imports only concern the assumptions directly stated in the theory. global imports also concern the ones the source theory inherits.	
type	omgroup, omdoc	alternative, contrast, narrative, dataset, datalabels, datadata, theory-collection
	the first three give the text category, the second three are used for generalized tables, and the last one for collections of theory.	
type	omlet	js, image
	the type of an omlet, e.g. 'image'	
type	omtext	abstract, antithesis, comment, conclusion, elaboration, evidence, introduction, motivation, thesis
	a specification of the intention of the text fragment, in reference to context.	
verdict	answer	

	specifies the truth or falsity of the answer. This can be used e.g. by a grading application.	
version	omdoc	1.1
	specifies the version of the document, so that the right DTD is used	
via	inclusion	
	points to a theory-inclusion that is required for an actualization	
width	omlet	
	specifies the width of the rectangle on the screen taken up by the results of an `omlet`	
xml:lang	*	
	the language the text in the element is expressed in. This must be a RFC-639 compliant specification of the primary language of the content.	
xmlns	omdoc	http://www.mathweb.org/omdoc
	fixes the OMDoc namespace	
xref	*	**ref, method, premise, presentation,** some OpenMath elements
	a uniform resource identifier (`URI`) used for cross-referencing. The element, this URI points to should be in the place of the object containing this attribute.	

Appendix E

The OMDoc Document Type Definition

We reprint the current version of the OMDoc document type definition. The original can be found at `http://www.mathweb.org/omdoc/dtd/omdoc.dtd`.

The DTD can be referenced by the public identifer `-//OMDoc//DTD OMDoc V1.1//EN`. Thus documents that use it have the document type declaration

```
<!DOCTYPE omdoc Public "-//OMDoc//DTD OMDoc V1.1//EN"
                       "http://www.mathweb.org/omdoc/omdoc.dtd">
```

in the preamble of the document.

The document type definition includes a variant document type definition for OpenMath objects that differs from the original (see `http://www.openmath.org`) in that it allows to represent OpenMath objects as directed acyclic graphs. This extension is licensed by the OpenMath Standard that says that any extension, from which valid OpenMath can be directly be generated, is allowed.

```
<!--
   An XML Document Type Definition for the Open Mathematical documents
   in the OMDoc format (Version 1.1)
   Initial Version: Michael Kohlhase 1999-09-07
   URL: http://www.mathweb.org/omdoc/omdoc.dtd (current released version)
   URL: http://www.mathweb.org/omdoc/dtd/old/omdoc*.dtd (old)
   Public Identifier: -//OMDoc//DTD OMDoc V1.1//EN
   Comments are welcome! (send mail to kohlhase@mathweb.org)
   See the documentation and examples at http://www.mathweb.org/omdoc mainly
   [1] http://www.matwheb.org/omdoc/omdoc.{ps,pdf}
   (c) 1999-2002 Michael Kohlhase, released under the GNU Public License
-->

<!-- ===================== ENTITIES ====================== -->
<!-- we define some entities for modularization, these can be
```

```
     re-defined in  the local subset of the DTD. -->

<!-- we allow OpenMath objects as mathematical objects -->
<!ENTITY % mobj "OMOBJ">

<!ENTITY % omdocns "xmlns CDATA #FIXED 'http://www.mathweb.org/omdoc'
                    xmlns:xsi CDATA 'http://www.w3.org/2001/XMLSchema-instance'
                    xsi:schemaLocation CDATA 'http://www.mathweb.org/omdoc
                                              http://www.mathweb.org/omdoc/xsd/omdoc.xsd'">
<!-- this namespace declaration also needs to go into all the elements
     that do not inherit from the top-level omdoc elements
     e.g. those in %inCMP; -->

<!ENTITY % theoryNSD "">
<!-- the namespace declaration attributes to be added to the omdoc element
     this entity should be redefined in the internal subset -->

<!-- what goes into a CMP element -->
<!ENTITY % alsoinCMP "">
<!ENTITY % inCMP "#PCDATA|%mobj;|omlet|with|ref|ignore%alsoinCMP;">
<!-- Persons in Dublin Core Metadata -->
<!ENTITY % DCperson "(#PCDATA)">
<!-- the date format in Dublin Core -->
<!ENTITY % DCdate "(#PCDATA)">
<!-- the identifier format for Dublin Core -->
<!ENTITY % DCident "(#PCDATA)">
<!-- the rest of Dublin Core content -->
<!ENTITY % DCrest "ANY">
<!-- any form of extra metadata -->
<!ENTITY % extrameta "EMPTY">

<!-- then define a couple of useful abbreviations, these are not
     intended for re-definition.  -->

<!ENTITY % midmatter "mid CDATA #IMPLIED">
<!-- attribute mid is an URIref, pointing to the MBase identifier
     of the element -->


<!-- we do not define the id attribute to be of type ID as one
     would expect, since we only want them to be unique in a theory,
     and we want still to be able to concatenate OMDoc files -->
<!ENTITY % idmatter "id CDATA #REQUIRED
                     style NMTOKEN #IMPLIED
                     %midmatter;">
<!ENTITY % idimatter "id CDATA #IMPLIED
                      style NMTOKEN #IMPLIED
                      %midmatter;">

<!ENTITY % idgmatter "%idmatter; generated-by CDATA #IMPLIED">
<!ENTITY % idrefmatter "%idmatter; for CDATA #REQUIRED">
<!-- attribute for is an URIref -->

<!ENTITY % insymbolmatter '%idmatter;
                           kind (type|sort|object) "object"
                           scope (global|local) "global"'>
```

```
<!--    The current XML-recommendation doesn't yet support the
        three-letter short names for languages (ISO 693-2). So
        the following section will be using the two-letter
        (ISO 693-1) encoding for the languages.

        en : English,   de : German,    fr : French,
        la : Latin,     it : Italian,   nl : Dutch,
        ru : Russian,   pl : Polish,    es : Spanish,
        tr : Turkish,   zh : Chinese,   ja : Japanese,
        ko : Korean     ...                     -->
<!ENTITY % ISO639 "(aa|ab|af|am|ar|as|ay|az|ba|be|bg|bh|bi|bn|bo|br|ca|co|
                    cs|cy|da|de|dz|el|en|eo|es|et|eu|fa|fi|fj|fo|fr|fy|ga|
                    gd|gl|gn|gu|ha|he|hi|hr|hu|hy|ia|ie|ik|id|is|it|iu|ja|
                    jv|ka|kk|kl|km|kn|ko|ks|ku|ky|la|ln|lo|lt|lv|mg|mi|mk|
                    ml|mn|mo|mr|ms|mt|my|na|ne|nl|no|oc|om|or|pa|pl|ps|pt|
                    qu|rm|rn|ro|ru|rw|sa|sd|sg|sh|si|sk|sl|sm|sn|so|sq|sr|
                    ss|st|su|sv|sw|ta|te|tg|th|ti|tk|tl|tn|to|tr|ts|tt|tw|
                    ug|uk|ur|uz|vi|vo|wo|xh|yi|yo|za|zh|zu)">

<!ENTITY % langmatter "xml:lang %ISO639; 'en'">

<!ENTITY % frommatter    "%idmatter; from CDATA #REQUIRED">
<!ENTITY % fromtomatter  "%frommatter; to CDATA #REQUIRED">
<!-- attributes 'to' and 'from' are URIref -->

<!ENTITY % otheromtexttype "">
<!ENTITY % omtexttype "abstract|introduction|conclusion|thesis|
                       antithesis|elaboration|motivation|evidence
                       |note|annote|comment%otheromtexttype;">

<!ENTITY % otheromgrouptype "">
<!ENTITY % omgrouptype "enumeration|sequence|itemize|narrative|
                        dataset|labeled-dataset%otheromgrouptype;">

<!ENTITY % cm "metadata?,CMP*">
<!ENTITY % cfm   "(metadata?,symbol*,CMP*,FMP*)">
<!ENTITY % otherassertiontype "">
<!ENTITY % assertiontype "(theorem|lemma|corollary|conjecture|
                           false-conjecture|obligation|postulate|
                           formula|assumption|proposition
                           %otherassertiontype;)">
<!ENTITY % otherdefinitiontype "">
<!ENTITY % definitiontype "(simple|inductive|implicit|recursive|obj
                           %otherdefinitiontype;)">

<!ENTITY % intheory-mathitem "type|assertion|alternative|example|proof|proofobject">
<!ENTITY % other-mathitem "theory-inclusion|decomposition|axiom-inclusion">
<!ENTITY % auxitem "exercise|solution|omlet|private|code|presentation|omstyle">
<!ENTITY % onlyintheoryitem "symbol|axiom|definition|adt|imports|inclusion">
<!ENTITY % otheromdocitem "">
<!ENTITY % intheory-omdocitem "omtext|%intheory-mathitem;|%auxitem;|theory|omgroup|ignore|ref
                      %otheromdocitem;">
<!ENTITY % omdocitem "%intheory-omdocitem;|%other-mathitem;">
<!ENTITY % intheoryitem "%onlyintheoryitem;|%intheory-omdocitem;">
```

```
<!-- ============= Document Structure [1; sec 2.2] ================ -->

<!ELEMENT omdoc (metadata?,catalogue?,(%omdocitem;)*)>
<!ATTLIST omdoc %idmatter; %omdocns; %theoryNSD;
                type (%omgrouptype;|theory-collection) #IMPLIED
                catalogue CDATA #IMPLIED
                version CDATA #FIXED "1.1">

<!ELEMENT catalogue (loc)*>

<!ELEMENT loc EMPTY>
<!ATTLIST loc theory CDATA #REQUIRED
              omdoc CDATA #IMPLIED
              cd CDATA #IMPLIED>
<!-- omdoc attributes omdoc and cd are URIRefs pointing to the omdoc
     and/or the OpenMath content dictionary defining this theory -->

<!ELEMENT omtext (metadata?,CMP+,FMP?)>
<!ATTLIST omtext %idmatter;
                 type (%omtexttype;) #IMPLIED
                 for CDATA #IMPLIED>
<!-- attribute 'for' is a URIref, to %omdocitem;s
     it is needed by the 'type' attribute-->

<!ELEMENT CMP (%inCMP;)*>
<!ATTLIST CMP %langmatter;>

<!ELEMENT with (%inCMP;)*>
<!ATTLIST with id ID #IMPLIED
               style NMTOKEN #IMPLIED
               %omdocns;>
<!-- identifies a text passage and
     allows to attatch style information to it -->

<!-- grouping defines the structure of a document-->
<!ELEMENT omgroup (metadata?,(%intheoryitem;)*)>
<!ATTLIST omgroup %idimatter;
                  type (%omgrouptype;) #IMPLIED>

<!-- co-referencing  allows to use elements with an
     'id' attribute multiple times -->
<!ELEMENT ref EMPTY>
<!ATTLIST ref xref CDATA #REQUIRED
              type NMTOKEN "include">
<!-- the types supported (there may be more over time) are
     - 'include' (the default) for in-text replacement
     - 'cite' for a reference with a generated label -->

<!-- ======= math Statements [1; sec 3.1] ================== -->

<!ELEMENT symbol (metadata?, CMP*,(commonname|type|selector)*)>
<!ATTLIST symbol %insymbolmatter;
                 generated-by CDATA #IMPLIED>

<!ELEMENT commonname (%inCMP;)*>
<!ATTLIST commonname %langmatter;
```

```
                    %midmatter;>

<!ELEMENT type (CMP*,%mobj;)>
<!ATTLIST type %idimatter;
               for CDATA #IMPLIED
               system NMTOKEN #REQUIRED>

<!ELEMENT FMP ((assumption*,conclusion*)|%mobj;)>
<!ATTLIST FMP logic NMTOKEN #IMPLIED
              %midmatter;>
<!-- If FMP contains a %mobj; then this is the assertion,
     if it contains (assumption*,conclusion*), then it is a
     logical sequent (A1,...,An |- C1,...,Cm):
     all the Ai entail one of the Ci -->

<!ELEMENT assumption (CMP*,(%mobj;)?)>
<!ATTLIST assumption %idmatter;>

<!ELEMENT conclusion (CMP*,(%mobj;)?)>
<!ATTLIST conclusion %idmatter;>

<!ELEMENT axiom %cfm;>
<!ATTLIST axiom %idgmatter;>

<!-- Definitions contain CMPs,  FMPs and concept specifications.
     The latter define the set of concepts defined in this element.
     They can be reached under this name in the content dictionary
     of the name specified in the theory attribute of the definition.
   -->

<!ELEMENT definition (metadata?,CMP*,FMP*,(requation+|%mobj;)?,
                      measure?,ordering?)>
<!ATTLIST definition just-by CDATA #IMPLIED
                     type  %definitiontype; "simple"
                     generated-by CDATA #IMPLIED
                     %idrefmatter;>
 <!-- attribute just-by is an URIref points to an assertion -->

<!ELEMENT requation (pattern,value)>
<!ATTLIST requation %idimatter;>

<!ELEMENT pattern (%mobj;)>
<!ELEMENT value (%mobj;)>

<!ELEMENT measure (%mobj;)>
<!ATTLIST measure %midmatter;>

<!ELEMENT ordering (%mobj;)>
<!ATTLIST ordering %midmatter;>

<!-- adts are abstract data types, they are short forms for
      groups of symbols and their definitions, therefore,
      they have much the same attributes. -->

<!ELEMENT adt (metadata?,commonname*,CMP*,sortdef+)>
<!ATTLIST adt type (loose|generated|free) "loose"
```

```
               %idmatter;>

<!ELEMENT sortdef (commonname*,(constructor|insort)*,recognizer?)>
<!ATTLIST sortdef %idmatter;
                  scope (global|local) "global">

<!ELEMENT insort EMPTY>
<!ATTLIST insort for CDATA #REQUIRED>
<!-- for is a reference to a sort symbol element  -->

<!ELEMENT constructor (commonname*,argument*)>
<!ATTLIST constructor %insymbolmatter;>

<!ELEMENT recognizer (commonname)*>
<!ATTLIST recognizer %insymbolmatter;>

<!ELEMENT argument (selector?)>
<!ATTLIST argument sort CDATA #REQUIRED>
<!-- sort is a reference to a sort symbol element  -->

<!ELEMENT selector (commonname)*>
<!ATTLIST selector %insymbolmatter;
                   total (yes|no) "no">

<!ELEMENT assertion %cfm;>
<!ATTLIST assertion %idgmatter;
                    theory CDATA #IMPLIED
                    type %assertiontype; "conjecture"
                    proofs CDATA #IMPLIED>
<!-- the %assertiontype; has no formal meaning yet, it is solely
     for human consumption. The 'generated-by' is for
     theory-interpretations, which can  generate assertions.
     'proofs' is a list of URIRefs -->

<!ELEMENT alternative (metadata?,CMP*,(FMP|requation*|%mobj;))>
<!ATTLIST alternative theory CDATA #REQUIRED
                      type  %definitiontype; "simple"
                      generated-by CDATA #IMPLIED
                      just-by CDATA #IMPLIED
                      entailed-by CDATA #REQUIRED
                      entails CDATA #REQUIRED
                      entailed-by-thm CDATA #REQUIRED
                      entails-thm CDATA #REQUIRED
                      %idrefmatter;>
<!-- the CDATA attributes are URIrefs
     just-by, points to the theorem justifying well-definedness
     entailed-by, entails, point to other (equivalent definitions
     entailed-by-thm, entails-thm point to the theorems justifying
     the entailment relation -->


<!-- OMDoc proofs consist of sequences of steps. The 'for' attribute
     specifies the assertion  it is for. -->

<!ELEMENT proof (metadata?,symbol*,CMP*,
                 (metacomment|derive|hypothesis)*,conclude)>
```

```
<!ATTLIST proof theory CDATA #IMPLIED
                %idrefmatter;>

<!ELEMENT proofobject (%cm;,%mobj;)>
<!ATTLIST proofobject theory CDATA #IMPLIED
                      %idrefmatter;>

<!ELEMENT metacomment (CMP*)>
<!ATTLIST metacomment %idimatter;>

<!ENTITY % justmatter "method?,premise*,(proof|proofobject)?">

<!ELEMENT derive (CMP*,FMP?,%justmatter;)>
<!ATTLIST derive %idmatter;>

<!ELEMENT conclude (CMP*,%justmatter;)>
<!ATTLIST conclude %idimatter;>

<!ELEMENT hypothesis (symbol*,CMP*,FMP?)>
<!ATTLIST hypothesis %idmatter;
                     discharged-in CDATA #REQUIRED>
<!-- the 'discharged-in' attribute points to the 'derive' or
     'conclude' element that discharges this hypothesis.
     The intended semantics is that the hypothesis will be
     local in the subtree rooted at that. -->

<!ELEMENT method ((%mobj;)*)>
<!ATTLIST method xref CDATA #REQUIRED>
<!-- 'xref' is a pointer to the element defining the method -->

<!ELEMENT premise EMPTY>
<!ATTLIST premise xref CDATA #REQUIRED
                  rank CDATA "0">
<!-- The rank of a premise specifies its importance in the
     inference rule. Rank 0 (the default) is a real premise,
     whereas positive rank signifies sideconditions of
     varying degree. -->

<!ELEMENT example (metadata?,symbol*,CMP*,(%mobj;)*)>
<!ATTLIST example type (for|against) #IMPLIED
                  assertion CDATA #IMPLIED
                  %idrefmatter;>
<!-- attributes assertion is an URIref -->

<!-- =========== Theories [1; sec 3.2] ==================== -->

<!ELEMENT theory (metadata?,commonname*,CMP*, (%intheoryitem;)*)>
<!ATTLIST theory id ID #REQUIRED
                 style NMTOKEN #IMPLIED>
<!-- theory identifiers should be unique per document -->

<!ELEMENT imports (CMP*,morphism?)>
<!ATTLIST imports %frommatter;
                  hiding CDATA #IMPLIED
                  type (local|global) "global">
<!-- hiding is a list of references to symbol ids -->
```

```
<!ELEMENT morphism (requation*)>
<!ATTLIST morphism %idimatter;
                   base CDATA #IMPLIED>
<!-- base points to some other morphism it extends -->

<!ELEMENT inclusion EMPTY>
<!ATTLIST inclusion via CDATA #REQUIRED
                    %midmatter;>
<!-- via points to a theory-inclusion -->

<!ELEMENT theory-inclusion (%cfm;,morphism?)>
<!ATTLIST theory-inclusion %fromtomatter;>
<!-- attribute by is a whitespace-separated  list of URIref -->

<!ELEMENT decomposition EMPTY>
<!ATTLIST decomposition %idrefmatter;
                        links CDATA #REQUIRED>
<!-- attribute 'for' points to a 'theory-inclusion', which this
     element justifies; attribute 'links' is an URIrefs, points to a
     list of axiom-inlcusions and theory-inclusions -->

<!ELEMENT axiom-inclusion (%cfm;,morphism?,(path-just|obligation*))>
<!ATTLIST axiom-inclusion %fromtomatter;>

<!ELEMENT path-just EMPTY>
<!ATTLIST path-just local CDATA #REQUIRED
                    globals CDATA #REQUIRED
                    %midmatter;>
<!-- attribute 'local' is an URIref, points to axiom-inclusion
               'globals' is an URIrefs, points to a list of
                         theory-inclusions -->

<!ELEMENT obligation EMPTY>
<!ATTLIST obligation induced-by CDATA #REQUIRED
                     assertion CDATA #REQUIRED
                     %midmatter;>
<!-- attribute 'assertion' is a URIref, points to  an assertion
     that is the proof obligation induced by the axiom or definition
     specified by 'induced-by. -->

<!-- ========== Auxiliary Elements [1; sec 3.3] ==================== -->

<!ELEMENT exercise (%cfm;,hint?,(solution*|mc*))>
<!ATTLIST exercise %idmatter;
                   for CDATA #IMPLIED>

<!ELEMENT hint %cfm;>
<!ATTLIST hint %idimatter;>

<!ELEMENT solution (%cfm;,(proof|proofobject)?)>
<!ATTLIST solution for CDATA #IMPLIED
                   %idimatter;>

<!ELEMENT mc (symbol*,choice,hint?,answer)>
<!ATTLIST mc %idimatter;>
```

```
<!ELEMENT choice %cfm;>
<!ATTLIST choice %idimatter;>

<!ELEMENT answer %cfm;>
<!ATTLIST answer verdict (true|false) #REQUIRED
                 %idimatter;>

<!ELEMENT omlet (%inCMP;)*>
<!ATTLIST omlet %idimatter;
                action NMTOKEN #IMPLIED
                type NMTOKEN #IMPLIED
                data CDATA #IMPLIED
                argstr CDATA #IMPLIED
                function CDATA #IMPLIED
                width CDATA #IMPLIED
                height CDATA #IMPLIED
                %omdocns;>
<!-- atribute action specifies the action to be taken when activated,
     attribute data is a URIref to data in a private element
     attribute argstr is a string of arguments supplied to the function
     attribute function is an URIref, points to a code element
     attribute width/height for screen display -->

<!ENTITY % privmatter "%idmatter;
                  for CDATA #IMPLIED
                  theory CDATA #IMPLIED
                  pto NMTOKENS #IMPLIED
                  pto-version NMTOKENS #IMPLIED
                  type NMTOKEN #IMPLIED
                  requires CDATA #IMPLIED">

<!ELEMENT private (metadata?,data+)>
<!ATTLIST private %privmatter;
                  replaces CDATA #IMPLIED>
<!-- 'replaces is a URIref to the omdoc elements that are replaced by the
     system-specific information in this element -->

<!ELEMENT code (metadata?,data+,input?,output?,effect?)>
<!ATTLIST code %privmatter;
               classid CDATA #IMPLIED
               codebase CDATA #IMPLIED>

<!ELEMENT input (CMP*,FMP*)>
<!ATTLIST input %midmatter;>

<!ELEMENT output (CMP*,FMP*)>
<!ATTLIST output %midmatter;>

<!ELEMENT effect (CMP*,FMP*)>
<!ATTLIST effect %midmatter;>

<!ELEMENT data ANY>
<!ATTLIST data %midmatter;
               format CDATA #IMPLIED
               href CDATA #IMPLIED
```

```
                size CDATA #IMPLIED>

<!-- this element can be used in lieu of a comment, it is read
     by the style sheet, (comments are not) and can therefore
     be transformed by them -->
<!ELEMENT ignore ANY>
<!ATTLIST ignore type NMTOKEN #IMPLIED
                 comment CDATA #IMPLIED>

<!-- =========== Presentation [1; sec 3.5] ===================== -->

<!ENTITY % crossreftype "(no|yes|brackets|separator|lbrack|rbrack|all)">
<!ENTITY % fixitytype "(prefix|infix|postfix|assoc)">

<!ENTITY % stylematter "%idimatter; xref CDATA #IMPLIED">

<!ENTITY % formatmatter "format CDATA #REQUIRED
                         requires CDATA #IMPLIED
                         xml:lang CDATA #IMPLIED">

<!ELEMENT presentation (use|xslt|style)*>
<!ATTLIST presentation %stylematter;
                       for CDATA #REQUIRED
                       parent (OMA|OMBIND|OMATTR) #IMPLIED
                       fixity %fixitytype; "prefix"
                       lbrack CDATA "("
                       rbrack CDATA ")"
                       separator CDATA ","
                       bracket-style (lisp|math) "math"
                       precedence NMTOKEN '1000'
                       crossref-symbol  %crossreftype; "yes"
                       theory CDATA #IMPLIED>

<!ELEMENT use (#PCDATA)>
<!ATTLIST use %formatmatter;
              bracket-style (lisp|math) #IMPLIED
              fixity %fixitytype; #IMPLIED
              lbrack CDATA #IMPLIED
              rbrack CDATA #IMPLIED
              larg-group CDATA #IMPLIED
              rarg-group CDATA #IMPLIED
              separator CDATA #IMPLIED
              element CDATA #IMPLIED
              attributes CDATA #IMPLIED
              crossref-symbol %crossreftype; #IMPLIED>
<!-- the attributes in the <use> element overwrite those in the
      <presentation> element, therefore, they do not have defaults -->

<!ELEMENT omstyle (xslt|style)*>
<!ATTLIST omstyle %stylematter;
                  for CDATA #IMPLIED
                  element CDATA #REQUIRED>

<!ELEMENT xslt (#PCDATA)>
<!ATTLIST xslt %formatmatter;>
<!-- this element contains xslt in a CDATA section -->
```

```

<!ELEMENT style (element|text|recurse|value-of)*>
<!ATTLIST style %formatmatter;>
<!-- this element contains mock xslt expressed in the elements below -->

<!ELEMENT element (attribute|element|text|recurse|value-of)*>
<!ATTLIST element name NMTOKEN #REQUIRED>

<!ELEMENT attribute (#PCDATA|value-of|text)*>
<!ATTLIST attribute name NMTOKEN #REQUIRED>

<!ELEMENT text (#PCDATA)>

<!ELEMENT value-of EMPTY>
<!ATTLIST value-of select CDATA #REQUIRED>

<!ELEMENT recurse EMPTY>
<!ATTLIST recurse select CDATA #IMPLIED>


<!-- ============= Variant OpenMath [1; sec 2.6] ================ -->

<!--  Now comes a NON-STANDARD (experimental) variant of the
      OpenMath Object DTD omobj.dtd (see http://www.openmath.org)

      It is extended with coreferences! (by adding the xlink
      %idxref; attributes to all open math elements).
      In particular, it adds the attributes id and xref to
      OMOBJ OMA OMBIND and OMATTR

      These extensions are licensed by the OpenMath Standard that
      says that any extension, from which valid OpenMath can be
      directly be generated is allowed.

      Note that this makes it less restrictive for OMA, OMS and
      OMV than the original. Maybe this can be changed in a
      future version by using XML schema. -->

<!ENTITY % omel "OMS|OMV|OMI|OMB|OMSTR|OMF|OMA|OMBIND|OME|OMATTR">
<!ENTITY % idxref "%midmatter;
                   style NMTOKEN #IMPLIED
                   id ID #IMPLIED
                   xref IDREF #IMPLIED">
<!-- attribute xref is an IDREF not an URIref, since we want to
     allow structure sharing in one document, but not long-distance -->

<!-- symbol, original OM, links make no sense -->
<!ELEMENT OMS EMPTY>
<!ATTLIST OMS name CDATA #REQUIRED
              cd CDATA #REQUIRED
              style NMTOKEN #IMPLIED>

<!-- variable original OM, links make no sense -->
<!ELEMENT OMV EMPTY>
<!ATTLIST OMV name CDATA #REQUIRED
              style NMTOKEN #IMPLIED>
```

```
<!-- integer; links make sense, since integers can be big -->
<!ELEMENT OMI (#PCDATA)>
<!ATTLIST OMI %idxref;>

<!-- byte array; links make sense, since byte arrays can be big -->
<!ELEMENT OMB (#PCDATA) >
<!ATTLIST OMB %idxref;>

<!-- string; links make sense, since strings can be big -->
<!ELEMENT OMSTR (#PCDATA) >
<!ATTLIST OMSTR %idxref;>

<!-- floating point; links make sense, since floats can be big -->
<!ELEMENT OMF EMPTY>
<!ATTLIST OMF dec CDATA #IMPLIED
              hex CDATA #IMPLIED
              %idxref;>

<!-- apply constructor; links make sense, no copied substructure -->
<!ELEMENT OMA (%omel;)*>
<!ATTLIST OMA %idxref;>

<!-- binding constructor & variable; links make sense,
     no copied substructure -->
<!ELEMENT OMBIND ((%omel;), OMBVAR, (%omel;))? >
<!ATTLIST OMBIND %idxref;>

<!-- bound variables, original OM, links make no sense -->
<!ELEMENT OMBVAR (OMV|OMATTR)+>

<!-- error; original OM, links make no sense -->
<!ELEMENT OME (OMS, (%omel;)* ) >

<!-- attribution constructor & attribute pair constructor -->
<!ELEMENT OMATTR (OMATP, (%omel;))? >
<!ATTLIST OMATTR %idxref;>

<!ELEMENT OMATP (OMS, (%omel;))+ >

<!-- OM object constructor; links make sense to avoid copying
     substructure -->
<!ELEMENT OMOBJ (%omel;)? >
<!ATTLIST OMOBJ xmlns CDATA #FIXED 'http://www.openmath.org/OpenMath'
                %idxref;>

<!-- ======= Dublin Core Metadata [1; sec 2.2.2 & app C] ======== -->

<!-- OMDoc Metadata comes in two forms:
     1) Bibliographic Metadata corresponding to the model of the
        Dublin Metadata initiative (http://purl.org/DC)
     2) other, mostly guided by the intuitions of the MBase system
  -->

<!ENTITY % dcelement "Contributor | Creator | Subject | Title
                    | Description | Publisher | Date | Type | Format
```

```
                       | Identifier | Source | Language | Relation | Rights">

<!ENTITY % dcns "xmlns CDATA #FIXED 'http://purl.org/DC'">
<!ENTITY % dcidi "%dcns; %idimatter;">
<!ENTITY % dcrole "%dcidi;  %langmatter;
           role (aut|ant|clb|edt|ths|trc|trl) 'aut'">
<!ENTITY % dclang "%dcns; %langmatter;">

<!ELEMENT metadata ((%dcelement;)*,extradata?)>
<!ATTLIST metadata %idimatter; inherits CDATA #IMPLIED>

<!-- first the Dublin Core Metadata model of the
     Dublin Metadata initiative (http://purl.org/dc) -->

<!ELEMENT Contributor %DCperson;><!ATTLIST Contributor %dcrole;>
<!ELEMENT Creator %DCperson;><!ATTLIST Creator %dcrole;>
<!ELEMENT Title (%inCMP;)*><!ATTLIST Title %dclang;>
<!ELEMENT Subject (%inCMP;)*><!ATTLIST Subject %dclang;>
<!ELEMENT Description (%inCMP;)*><!ATTLIST Description %dclang;>
<!ELEMENT Publisher %DCrest;><!ATTLIST Publisher %dcidi;>
<!ELEMENT Type %DCrest;><!ATTLIST Type %dcns;>
<!ELEMENT Format %DCrest;><!ATTLIST Format %dcns;>
<!ELEMENT Source %DCrest;><!ATTLIST Source %dcns;>
<!ELEMENT Language %DCrest;><!ATTLIST Language %dcns;>
<!ELEMENT Relation %DCrest;><!ATTLIST Relation %dcns;>
<!ELEMENT Rights %DCrest;><!ATTLIST Rights %dcns;>

<!ELEMENT Date %DCdate;>
<!ATTLIST Date %dcns; action NMTOKEN #IMPLIED who IDREF #IMPLIED>

<!ELEMENT Identifier %DCident;>
<!ATTLIST Identifier %dcns; scheme NMTOKEN "ISBN">

<!-- other metadata that is not bibliographic can be included in the
     <extradata> element, declare any needed XML elements in the
     internal subset of the DTD declaration -->
<!ELEMENT extradata %extrameta;>

<!-- =============== omdoc.dtd ends here  ==================== -->
```