UNIVERSITÄT DES SAARLANDES
FACHBEREICH INFORMATIK
D-66041 SAARBRÜCKEN
GERMANY
WWW: http://www.ags.uni-sb.de/

OMDoc: An Open Markup Format for Mathematical Documents

SEKI Report
ISSN 1437-4447

Michael Kohlhase

SEKI Report SR-SR-00-02

OMDoc: An Open Markup Format for Mathematical Documents

Michael Kohlhase
FB Informatik, Universität des Saarlandes
D-66041 Saarbrücken, Germany
http://www.ags.uni-sb.de/~kohlhase

Abstract

In this report we present an extension OMDoc to the OpenMath standard that allows to represent the semantics and structure of various kinds of mathematical documents, including articles, textbooks, interactive books, 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.

Status of this document: *This document describes version 1.0 of the* OMDoc *format, released November 1. 2000. Version 1.0 is the result of using* OMDoc *in various experiements and projects, stabilizes these experiences to serve as a basis for more sophisticated tools. The document type definition can be found at* `http://www.mathweb.org/omdoc/dtd/omdoc1.0.dtd`.

The OMDoc *format will continue to evolve, the next steps are to include XML Schema support and more sophisticated (authoring) tools. A development snapshot can be retrieved by anonymous CVS (see* `http://www.mathweb.org/cvs.html` *) directly accessed on the web at* `http://www.mathweb.org/src/mathweb/omdoc`. *If you are interested in this, please contact the author.*

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 ten 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 if this idea [FHJ+99, FK99, DCN+00, AZ00] by the term MathWeb, since we believe that interoperability based on communication protocols will eventually make the constructions of bridges between the particular implementations simple, so that that the combined systems appear to the user as one homogenous web.

One of the tasks that have to be solved in order to arrive is to define open markup languages for the mathematical objects and knowledge exchanged between the mathematical services. The OMDoc format presented in this report attempts to do this by providing infrastructure for the communication and storage of *mathematical knowledge* based on the OpenMath standard.

Before we introduce the ideas behind this format, let us set the stage by giving a brief overview of available markup schemes and relevant Internet standards.

1.1 Mathematical Markup Schemes and the Internet

Mathematical texts[1] are usually very carefully designed to given them a structure that supports understanding the complex structure of the objects discussed and that of the argumentations about them.

The observation that the task of recovering the semantic structure from the representation it is given in (e.g. as a written text or a recording) is central to understanding what it is about 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. that can be used to convey this structure. 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 mainly with "ink-on-paper" representations (books, journals, letters) for the consumption by humans (mathematicians).

In the age of Internet publication and mathematical software systems, this target turns out to be too limited in many forms. The universal accessibility of the documents on the Internet breaks the assumption 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. The fact that mathematical software systems are more and more embedded into the process of

[1] 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 all sciences, but not for novels or poems. Therefore, we will use the adjective "mathematical" in this report in an inclusive way to make this distinction on text form, not on strictly the scientific labeling.

doing mathematics breaks the assumption that mathematical documents are primarily for human consumption. This, together with the fact that mathematical documents are primarily produced and stored on computers has led to the development of specialized **markup** schemes. Let us discuss some of the paradigmatic examples to get a feeling for the issues involved.

Text processors and and desktop publishing systems (think for example of Microsoft Word) are software systems (often of WYSIWYG type) aiming to produce "*ink-on-paper*" or "*pixel-on-screen*" representations of documents and are very well-suited to execute the typographic conventions mentioned above. Their internal markup scheme mainly[2] defines presentation traits like character position, font choice and characteristics, page breaks. This is perfectly sufficient for producing high-quality presentations of the documents on paper or on a 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 LaTeX as a paradigmatic example here. A typical section heading would be specified by something like this:

```
\section[{\TeX}]{The Joy of  {\index*${\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, and various **style** declarations, e.g. in the form of LaTeX style files (LaTeX itself is just a set of style files on top of TeX). This program will transform the functional specification using the style information into a markup scheme that specifies the form, like DVI, or PostScript that can directly be presented on paper or on the screen. Note that e.g. renumbering is not a problem in this approach, since the actual numbers are only inferred by the formatter. 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 roughly only used for document delivery.

To contrast the 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.

Another member of the *content* markup family that also takes the problem of document **metadata** into account, i.e. to describe the documents themselves and the relations among them (cf. 2.3) is the "**Simple Generalized Markup Language**" SGML. It tries to give the markup scheme a more declarative semantics (as opposed to the purely procedural – and rather baroque – semantics

[2]Of course, we overstress the issues; due to economic pressures, none of the markup schemes survives in a pure form anymore.

of TEX), to make it simpler to reason about (and this reuse) documents. It comes with its own (functional) style sheet language (DSSSL) and formatter.

The Internet, where screen presentation, hyperlinking, computation 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 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 (Internet documents are in general hypertexts), *iii*) we do not know the computational ressources of the computer driving the screen, and therefore have to decide which formatting steps to perform on the server, and which on the client side, and finally, the related problem that *iv*) 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) that is executed by the browser. The need for content markup schemes for maintaining documents on the server, as well 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 ressource-hungry for integration into web browsers.

This diversity problem has recently led to the development to the unifying "**eXtensible Markup Language**" XML [BPSM97] framework for Internet markup languages. Conceptually speaking, XML views a document as a tree of so-called **elements** (see Figure 1.2 for an example). For communication this tree is represented (as in SGML) as a well-formed bracketing structure, 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**, 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.

From SGML, XML also inherits the concept of a "**document type definition**" (**DTD**), i.e. a context-free grammar that defines the set of well-formed documents in a given XML language, by defining the set of admissible trees[3] as those that are accepted by this grammar. As a consequence (cleaned up versions of) most Internet markup schemata e.g. HTML can be defined by a DTD in XML, making general tool support available to them. In particular, this allows documents to be validated by generic tools (**parsers**).

XML comes with the XSL **style sheet** mechanism [Dea99], that was designed as a simplified subset of DSSSL that is lightweight enough to allow integration of XSL-transformers into browsers (they are present in version 5 of Microsoft's Internet Explorer and in version 6 of the Netscape Communicator).

A problem that we have stated in the beginning, but not discussed is that the availability of mathematical software systems breaks the assumption that mathematical documents are targeted only for human consumption. This assumption severely limits their usefulness. Take for instance a user that wants to experiment withe the mathematical formulae that she has just read in a mathematical book in a computer algebra, e.g. to graph them or calculate variants has to retype them into the system, very possibly making errors that make the result unusable. The problem is that presentation markup is that it is specified to be *machine-readable* (e.g. to the browser) but not *machine-understandable*. With the advent of the Internet, which is world's fastest growing and quickly also becoming the largest repository of mathematical documents, it is not possible to

[3]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 our section 2.2 on cross-referencing in OMDOC

manage all the available knowledge manually, because of the volume of information distributed over the Web. Generally, it is very hard to automate anything for documents whose structure is specified by presentation markup, therefore, we have to develop powerful content markup schemes for mathematical documents to make them also machine-understandable.

The OMDoc format presented in this report is an attempt to do just this, based on existing Internet standards like XML and ideas from mathematical practice, text theory and the field of algebraic specification. In order to structure the discussion, we will distinguish two levels of markup structure, the "**microscopic**" level that is used to mark up the mathematical objects, usually encoded by mathematical formulae, and the "**macroscopic**" level of document markup, which is concerned with the (closely related) questions of document structure and the structure of the underlying mathematical theories. In mathematical texts, there is possibly a third level, in which both microscopic and macroscopic are especially tightly intertwined. Proofs, are about mathematical objects, but have a very explicit, specialized (and important) structure; we will leave discussion of them to section 2.7.

1.2 The microscopic level: OpenMath, and MathML

The two best-known open[4] markup formats for representing mathematics for the Web are MathML and OpenMath.

MathML [CIMP01] is an XML-based markup scheme for mathematical formulae, in a nutshell, its main goal is to bring **TEX** formula presentation to the Web. Since the aim is to do most of the formatting inside the browser, where ressource 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), 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. It does however not offer the programming language features of TEX[5] for the obvious computing ressource considerations. Fully aware of the advantages of content markup, it also offers infrastructure for that, called **content** MathML, and a specialized `semantics` element that allows to annotate MathML formulae with content information in any format. This is intended to hold representations of the formulae in other markup schemes, e.g. so that they can be passed on to the computer algebra system we discussed above.

In contrast to this very rich language that defines the meaning of extended presentation primitives, the meaning of the concepts of K-12 mathematics, 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. 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`). For convenience, OpenMath also provides other basic data types useful in mathematics: `OMI` for integers, `OMB` for byte arrays, `OMSTR` for s, 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`[6] element. The content dictionaries that make up the extention mechanism provided in OpenMath are tied into the object representation by the `cd` attribute of the

[4]There are various other formats that are proprietary or based specific mathematical software packages like Wolfram Research' Mathematica. We will not concern ourselves with them for the obvious reasons.

[5]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 XSL language it inherits from XML.

[6]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.

OMS element that specifies the defining content dictionary.

OpenMath and MathML are well-integrated:

- the basic content dictionaries of OpenMath mirror the MathML constructs, 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, and OpenMath supports the presentation attribute that can be used for annotating with MathML presentation.
- OpenMath is the designated extension mechanism for MathML beyond K-12 mathematics.

Therefore, it is not a limitation of the presentational capabilities to use OpenMath for marking up mathematical objects. As MathML can be viewed by the WebEQ plug-in and is going to be natively supported by the primary browsers MS Internet Explorer and Netscape Navigator in version 6 (see http://www.mozilla.org for Mozilla, the open source version), MathML will be the primary presentation language for OMDoc.

```
<OMOBJ id="commutativity-formula">
  <OMBIND>
    <OMS cd="quant1" name="forall"/>
    <OMBVAR>
      <OMV name="a"/>
      <OMV name="b"/>
    </OMBVAR>
    <OMA><OMS cd="logic1" name="implies"/>
      <OMA><OMS cd="logic1" name="and"/>
        <OMA><OMS cd="set1" name="in"/><OMV name="a"/><OMS cd="reals" name="real"/></OMA>
        <OMA><OMS cd="set1" name="in"/><OMV name="b"/><OMS cd="reals" name="real"/></OMA>
      </OMA>
      <OMA><OMS cd="relation" name="eq"/>
        <OMA><OMS cd="reals" name="plus-real"/><OMV name="a"/><OMV name="b"/></OMA>
        <OMA><OMS cd="reals" name="plus-real"/><OMV name="b"/><OMV name="a"/></OMA>
      </OMA>
    </OMA>
  </OMBIND>
</OMOBJ>
```

Figure 1.1: An OpenMath representation of $\forall a, b. a + b = b + a$.

Since OMDoc uses OpenMath at the microscopic level, let us build up our intuition about it with an example. Figure 1.2 shows an OpenMath representation of the law of commutativity for addition on the reals (the logical formula $\forall a, b. a \in \mathbf{R} \wedge b \in \mathbf{R} \rightarrow a + b = b + a$). The mathematical meaning of a symbols (that of applications and bindings is known from the folklore) is specified in a so-called **content dictionary**, 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

```
<CDDefinition>
  <Name>plus</Name>
  <Description>Addition on real numbers</Description>
  <CMP>Addition is commutative</CMP>
  <FMP><OMOBJ xref="commutativity-formula"/></FMP>
</CDDefinition>
```

could be part of the content dictionary[7] reals.ocd for elementary properties of real numbers (cf. section 2.5.2 for the relation of content dictionaries with OMDoc documents).

[7] In fact the reference <OMOBJ xref="commutativity-formula"/> pointing to the OMOBJ with the id attribute commutativity-formula uses an extension of OMDoc to OpenMath that allows to represent formulae as directed acyclic graphs preventing exponential blowup (see section C). It is licensed by the OpenMath standard, since pure OpenMath trees can be generated automatically from it.

1.3 The Macroscopic level: OMDoc

In the last section we have seen that the microscopic level of formula markup has been sufficiently dealt with by OpenMath and MathMl. This level of support is sufficient for the communication needs of symbolic computation services like computer algebra systems, which manipulate (simplify) or compute objects like equations or groups. Even though the logical formulae constructed or manipulated by reasoning systems like the ΩMEGA system can be expressed as OpenMath objects, mathematical services like reasoners or presentation systems need more information e.g.:

1. is this formula an axiom, a definition, or a theorem to be proven?
2. what is a good strategy to proceed with the proof in this domain?
3. is this mathematical concept basic, or defined (so that it can be expanded to a formula involving simpler concepts)?
4. is this concept, proof, or theory, a generalization or special case, of some other?
5. what is the common name of this concept (and its grammatical category), how is it usually written?
6. who is the author of this theorem, article, proof?

Unfortunately, OpenMath fulfills this goal only partially, since it exclusively deals with the representation of the mathematical objects proper.

> [...] a standard for representing mathematical objects, allowing them to be exchanged between computer programs, stored in databases, or published on the worldwide web. [...] [CC98]

Of course it would be possible to characterize an axiom by applying a predicate "axiom" to a formula or using a special variant of the equality relation for definitions, but this would only solve item 1 above.

To define a "*macroscopic*" structure level that classifies the kind of knowledge discussed above, we will use mathematical documents as a guiding intuition for mathematical knowledge, since almost all of mathematics is currently communicated in this form (publications, letters, e-mails, talks,...). To ensure widespread applicability, we will use the term document in an inclusive, rather than exclusive way (including papers, letters, interactive books, e-mails, talks, communication between mathematical services (see for instance [FK99, FHJ+99]) on the Internet,...), claiming that all of these can be fitted into a common representation.

The general pattern "definition, theorem, proof" has long been considered paradigmatic of mathematical documents like math. textbooks and papers. To support this structure, OMDoc provides elements for them which we will describe in sections 2.6 and 2.5. This structure is augmented by the large-scale structure of mathematical theories, here we build on concepts (see section 2.8) from the field of algebraic specification where structured representation of large corpora of formal scientific knowledge (about the meaning of programs) has been studied extensively. But mathematical documents contain more than this: specialized document parts like proofs (see section 2.7, exercises, applets, notation (see section 2.9) are intermixed with explanatory text (section 2.4). Instead of motivating and explaining them here (that will be the role of the next chapter), let us evaluate the scope of OMDoc by looking at a few possible applications. OMDoc can serve as

- 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 discourse structure[8] explicit, and then formalizing

[8] classifying text fragments as definitions, theorems, proofs, linking text, and their relations; we follow the terminology from computational linguistics here.

the textually given mathematical knowledge in logical formulae (by adding `FMP` elements; see sections 2.6 and 1.2).

- 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 the 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. 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.

These and similar applications are pursued in the ΩMEGA project at the Saarland University, Saarbrücken (see `http://www.ags.uni-sb.de/~omega`) in cooperation with the RIACA project at Eindhoven (see `http://www.riaca.win.tue.nl`) and the automated reasoning project at Edinburgh (see `http://dream.dai.ed.ac.uk`).

Chapter 2

OMDoc: Open Mathematical Documents

In this chapter, we discuss the OMDoc language features and their meaning. We will group by their meaning: first we will show the general text elements, after all, OMDoc is a markup language for mathematical documents/texts. Then we will concern ourselves with mathematical structures in the documents, i.e. theorems and proofs in section 2.6. In section 2.5, we will present markup elements for specifying the macroscopic structure of a mathematical document or theory. Finally, we will devote section 2.9 to auxiliary elements in OMDoc, which escape the classification in the other sections. The only part of OMDoc, which will not be shown in this chapter is the infrastructure for presenting OpenMath symbols, since it is very much tied to the processing of OMDocs. It will be presented in section 3.2.

In appendix A, we give a quick-reference table of the OMDoc elements, their attributes and contents.

2.1 Document Structure

Since OMDoc is an extension of OpenMath, it inherits its connections to Xml and MathMl. The structure of OMDoc documents is defined in the the OMDoc **document type definition DTD** (cf. appendix C).

An OMDoc document is bracketed by the Xml tags `<omdoc>` and `</omdoc>`, and consists of a sequence of OMDoc elements, that contain specialized representations for text, assertions, theories, definitions,... (see below). In contrast to markup languages like LaTeX, OMDoc does not partition the documents into specific units like chapters, sections, paragraphs, by tags and nesting information, but makes these document relations explicit with `omgroup` elements (see section 3.2). This choice is motivated by the generality of the document classes and the fact that the relative position of OpenMath documents can be determined in the presentation phase. In particular, since OpenMath documents can be hypertext documents, or generated from a database, it can be impossible to determine the structure of a document in advance, therefore we consider document structure information as presentation information and describe it in section 3.2.

```
<?xml version="1.0">
<!DOCTYPE omdoc SYSTEM "http://www.mathweb.org/omdoc/omdoc.dtd" []>
<omdoc xmlns="http://www.mathweb.org/omdoc" id="omdoc.example">
 ...
</omdoc>
```

Figure 2.1: The General Structure of an OMDoc.

Let us now take a look at the structure of an OMDoc document: Since OMDoc documents are XML documents specified by the OMDoc document type definition, all text is contained in a root node, the element `omdoc`, which has an attribute that uniquely identifies the document. Thus an OMDoc is has the general structure shown in Figure 2.1.

The `omdoc` element is the XML document element (i.e. the root node of the tree representing the document). As usual in XML there may only be one document node in an XML file. The `xmlns` attribute declares the OMDoc namespace, giving the elements used in the document an unambiguous meaning; the one described in the material at `http://www.mathweb.org/omdoc/omdoc`, i.e. the one described in this manual. Note that it can be left out in some cases[1], but if you put it, then it has to be exactly the string `http://www.mathweb.org/omdoc`, character for character, since otherwise applications will not recognize the vocabulary as OMDoc.

The `omdoc` element contains the various other elements that we will describe in this report (see appendices A for an overview and C for the document type definition, which is the primary reference).

Note that some OMDoc elements (those that have the `ANY` content model, most notably `CMP` and `extradata`) can contain elements other than the ones described in this manual, if they are declared in the internal subset of the `DOCTYPE` declaration (the empty `[]` in Figure 2.1). This allows to embed other XML vocabularies into these elements. In particular, in `CMP`, we would like to embed XHTML or MATHML for formatted markup of the text sections. The extradata is intended for user-defined metadata. If we want to use elements `<abstraction>` or `<difficulty>` in `extradata` and MATHML in the `CMP`, then we need a document type declaration like the one in figure Figure 2.1.

```
<!DOCTYPE omdoc SYSTEM "http://www.mathweb.org/omdoc/omdoc.dtd"
  [<!ELEMENT abstraction EMPTY>
   <!ATTLIST abstraction level CDATA #REQUIRED>
   <!ELEMENT difficulty EMPTY>
   <!ATTLIST difficulty level CDATA #REQUIRED>
   <!ENTITY % mathmldtd SYSTEM "http://www.w3.org/Math/DTD/mathml1/mathml.dtd"
   %mathmldtd;]>
```

Figure 2.2: A Document Type Declaration with Internal Subset

Let us now come to the more conceptual points: OMDoc is concerned with general markup for mathematical documents. Such documents are usually very carefully structured to make understanding of the complex material as simple as possible for the reader. The OMDoc format aims a providing XML-based content markup for this structure.

There are various problems that have to be overcome to reach this goal:

1. mathematical documents (at least larger ones, such as books) are usually structured along two **axes**, the *document structure*, and the and the *theory structure*. These can be compatible (and in well-structured mathematical documents often are), but need not in general. In this section we will concern ourselves with the document structure and leave the theory structure to chapter 2.5. At the moment it is sufficient to note that the theory structure will be represented by specialized `theory` elements that group mathematical objects like symbol declarations, axioms, etc. and specifies the dependency relations among them (see section 2.5). The document structure will be expressed directly by XML constructs.

2. mathematical documents (as other argumentative discourses) are normally not linear, but have a tree structure. OMDoc adopts the well-known "**Rhetorical Structure Theory**" **RST** [MT83, Hor98] content model, which models a text as a tree whose leaves are the

[1] It is declared in the document type definition, so validating XML applications can pick it up from there, if you are not sure whether your document will be handled only by validating applications, better put it there.

sentences (or phrases) and whose internal nodes model the relations between their daughters. In OMDoc, we use `omgroup` elements for the internal nodes of the tree and `omtext` nodes for the leaves. We use the `rsrelation` attributes to make this information explicit. These attributes specify the relation type in a `type` attribute and the RST tree daughters in attributes `for` (for the head daughter) and `from` for the others. At the moment OMDoc uses a variant of the RST [MT83] content model that supports the relation types `abstract`, `introduction`, `conclusion`, `thesis`, `antithesis`, `elaboration`, `motivation`, `evidence`, `linkage`, `narrative`, `sequence`, `alternative`, `general` with the obvious meanings, motivated by the application to mathematical argumentative texts (see also [Hor98]). The relation type also determines the default presentation.

This markup scheme generalizes those for text fragments offered e.g. by LaTeX into categories like "Introduction", "Remark", or "Conclusion".

3. mathematical document trees have nodes that are not present in "normal" argumentative texts: Definitions, Lemmata, Proofs, represented by the mathematical elements covered in section 2.6. As they can be seen as specialized text elements, they also have `rsrelation` attributes. Their internal discourse structure is determined by the specialized markup they provide.

4. mathematical documents normally do not have a tree structure, which could be naturally represented in Xml. Therefore, we have `ref` elements that can be used to reference to elements defined elsewhere (cf. footnote 4 in the context of OpenMath elements).

Element	Attributes		D	Content
	Required	Optional	C	
omdoc	id	type, catalogue	+	OMDoc element
catalogue			–	loc*
loc	theory	omdoc, cd	–	EMPTY
omgroup			+	OMDoc element
ref		xref, theory, name kind	–	ANY
omtext	id	rsrelation	+	CMP+,FMP?
CMP		type, xml:lang	–	ANY

Figure 2.3: The general OMDoc elements

This structuring approach allows to "**flatten**" the tree structure in a document into a list of leaves and relation declarations (see Figure 2.1 for an example). While this is a much more flexible (database-like) approach to representing structured documents[2], it puts a much heavier load on a system for presenting the text to humans. In essence the presentation system must be able to recover the left representation from the right one in Figure 2.1. Generally, any OMDoc element defines a fragment of the OMDoc it is contained in: everything that this element contains,

[2]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.

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 2.1 is just the one on the right.

```
<omtext id="text">
 TEXT1
 <omgroup type="enumeration">
  <omtext id="t2">TEXT2</omtext>
  <omtext id="t3">TEXT3</omtext>
 </omgroup>
 TEXT4
</omtext>
```

$\longleftrightarrow$

```
<omtext id="t1">TEXT1</omtext>
<omtext id="t2">TEXT2</omtext>
<omtext id="t3">TEXT3</omtext>
<omtext id="t4">TEXT4</omtext>

<omgroup id="text" type="narrative">
  <ref xref="#t1"/>
  <ref xref="#enum"/>
  <ref xref="#t4"/>
</omgoup>

<omgroup id="enum" type="sequence">
  <ref xref="#text2"/>
  <ref xref="#text3"/>
</omgoup>
```

Figure 2.4: Flattening a tree structure

2.2 Cross-referencing and the OMDoc Catalogue

Let us now take a closer look at the hyperlinks and cross-referencing across OMDoc files. The `ref` elements support the two structural axes mentioned in item 1 above, since these are the principal two coordinate systems, an OMDoc author thinks in. The attribute `xref` is used for the document structure axis and the attributes `theory` and `name` are used for the theory axis.

As we have seen above, OpenMath uses the theory axis to reference symbols (theories closely correspond to content dictionaries, see 2.5.2). This assumes that it is always known, where the defining OMDoc[3] can be found. This is not always granted in the times of the Internet, where documents can be widely distributed.

OMDoc specifies the document structure using the Xml primitives for grouping and hyperlinks (in particular the XLink/XPointer specification; see[DMOT01, DJM01]).

If we know where the defining OMDoc is, then the two methods of referencing are equivalent. In particular, if the target element is defined in the same OMDoc, then it is sufficient to specify its `id` attribute in the `xref` attribute, otherwise, it must include the relevant `URL` or `xpointer` material. Say it can be found at the URL `http://www.somewhere.org/theo`, then we can substitute `<ref theory="theo" name="nam"/>` with `<ref xref="http://www.somewhere.org/theo#nam"/>` and vice versa. The situation for the OpenMath element `OMS` is similar, only that we have the `cd` attribute instead of `theory` for historical reasons.

It is important to support the structural axis in OMDoc for tool support (see section 3), and the theory axis for the users, since it is closer to the conceptualization of the content. Therefore, OMDoc supports a **catalog** 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 whose catalog is inherited for the referencing OMDoc.

locally The local catalogue is declared in the `catalogue` element, it contains a sequence of `location` elements, which empty and has three attributes to convey the following declaration: The theory specified by the `theory` attribute is defined in the OMDoc, the URL in the attribute `omdoc` points to, and analogously for the `cd` attribute.

[3] i.e. the OMDoc that contains the theory definition; equivalently, the defining document could actually be an OpenMath content dictionary (which amounts to the same, as one can be transformed into the other).

The **effective catalogue** for an OMDoc is a sequence of location declarations. It (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 same order, then the local catalogue declaration is appended at the end. Then double location declarations are eliminated, later declarations overwriting the earlier.

This effective catalogue is then used to determine the location of any theory referenced in the OMDoc.

2.3 Metadata

The World Wide Web was originally built for human consumption, and although everything on it is machine-readable, this data is not machine-understandable. The accepted solution to use *metadata* to describe the data contained on the Web. In OMDoc, we use one of the best-known metadata schemas for documents – the **Dublin Core**. This decision also makes OMDoc compatible with e.g. the standard [Gro99] proposed by the OPEN eBOOK initiative.

The OMDoc `metadata` element is used to provide information about the publication as a whole, as well as specific fragments of the document. It contains specific publication-level metadata as defined by the Dublin Core initiative (`http://purl.org/dc/`). The descriptions below are included for convenience, and the Dublin Core's own definitions take precedence (see `http://www.ietf.org/rfc/rfc2413.txt`).

The OMDoc `metadata` element can contain any number of instances of any Dublin Core elements in any order (see appendix B for a specification and appendix C for a document type definition); in fact, multiple instances of the same element type (multiple `Creator` elements, for example) can be interspersed with other metadata elements without change of meaning.

As OpenMath documents, are often used to formalized 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.

- If 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 (yielding D'), then the metadata declaration of D' should have the following form:

```
<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>
```

- Researcher R formalizes the theory of natural numbers and for this looks into the standard textbook B (written by author A) for number theory. In this case we recommend something like the left declaration for the whole document and and the right one for for specific math elements, e.g. a definition inspired by or adapted from one in book B. See appendix B for details and Figure 2.3 for quick-reference.

```
<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>
```

2.4 Text Elements

The OMDoc text elements are Xml elements that can be used to accommodate and classify the explanatory text parts in mathematical documents. We have two kinds of them:

Element	Attributes		D	Content
	Required	Optional	C	
metadata			–	(element)*
Title		xml:lang	–	ANY
Contributor		role	–	ANY
Creator		role	–	ANY
Subject			–	ANY
Description			–	ANY
Publisher			–	ANY
Date		action	–	ANY
Type			–	ANY
Format			–	ANY
Identifier		scheme	–	ANY
Source			–	ANY
Lanugae			–	ANY
Relation			–	ANY
Coverage			–	ANY
Rights			–	ANY
extradata			–	ANY

Figure 2.5: The OMDoc metadata

CMP These text elements are used for comments and describing mathematical properties inside other OMDoc elements. They have an xml:lang attribute that specifies the language, they are written in, thus using groups of CMPs with different languages can be used for OMDoc index*internationalization. In conformance 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...).

CMPs may contain arbitrary text interspersed with OpenMath objects (OMOBJ elements) (see the OpenMath standard [CC98] for details), omlets (see section 2.9) and hyperlinks (see below). No other elements are allowed. 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. Xsl style sheets, which can base their decisions on presentation information (see section 3.2) and then rsrelation information described in this section.

omtext OMDoc text elements can appear on top level (inside omdoc elements). They have an id attribute, so that they can be cross-referenced, (optional) rsrelation attributes specifying the **rhetorical structure relation** of the text to other OMDoc elements, and contain

1. an (optional) metadata declaration (we use the well-known Dublin Core schema, cf. http://purl.org/dc/ or see appendix 2.3)
2. a non-empty set of CMP elements that contain the text proper.

Element	Attributes		D	Content
	Required	Optional	C	
omdoc	id	type, catalogue	+	OMDoc element
catalogue			–	loc*
loc	theory	omdoc, cd	–	EMPTY
omgroup			+	OMDoc element
ref		xref, theory,	–	ANY

In software engineering a closely related concept is known under the label of an (algebraic) **specification**, that is used to specify the intended behavior of programs. There, the concept of a theory (specification) is much more elaborated to support the structured development of specifications. Without this structure, real world specifications become unwieldy and unmanageable. In this section we will only discuss simple theories and leave the discussion of the more advanced structuring concepts to sections 2.8 and 2.5.1.

Element	Attributes		D	Content
	Required	Optional	C	
theory	id		+	commonname*,CMP*,most below
symbol	id	type, scope	+	CMP*,(commonname|type|selector)*
commonname		xml:lang	–	ANY
signature	id, for, system		–	
type	system		–	OMOBJ
axiom	id		+	private*,symbol*,CMP*,FMP?
definition	id, for	type, just-by	+	CMP*,(FMP+|requation+|OMOBJ)?
requation			–	pattern, value
pattern			–	OMA|OMS
value			–	OpenMath element
imports	id, from	type, hiding	–	CMP*,morphism?
morphism	id	base	–	requation*
inclusion	for		–	
adt	id	type	+	CMP*,commonname*,sortdef+
sortdef	id	kind, scope	–	commonname*,(symbol|insort)*
constructor	id	type, scope	+	commonname*,argument*
argument	sort		+	selector?
insort	for		–	
selector	id	type, scope, kind	–	commonname*
theory-inclusion	id, from, to, by	timestamp	+	(morphism,decomposition?)
axiom-inclusion	id, from, to	timestamp	+	morphism?, (path-just|assertion-just))
assertion-just	ids	timestamp	–	EMPTY
decomposition	links	timestamp	–	EMPTY
path-just	local, globals	timestamp	–	EMPTY

Figure 2.7: The OMDoc Theory Elements

Theories are specified by the `theory` element in OMDoc. Since signature and axiom information is particular to a given theory, the `symbol`, `definition`, `axiom` elements must be contained

in a theory as sub-elements.

```
<theory id="monoid-thy">...
  <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>...
</theory>
```

Figure 2.8: An OMDoc symbol declaration

`symbol` This element specifies the symbols for mathematical concepts, such as 1 for the natural number "one", + for addition, = for equality, or `group` for the property of being a group. The `symbol` element has an `id` attribute which uniquely identifies it. This information is sufficient to allow referring back to this symbol as an OpenMath symbol. For instance the symbol declaration in Figure 2.5 gives rise to an OpenMath symbol that can be referenced as `<OMS cd="monoid" name="monoid"/>`. If the document containing this `symbol` element were stored in a data base system, the OpenMath symbol could be looked up by its common name. The type information specified in the `signature` element characterizes a monoid as a three-place predicate (taking as arguments the base set, the operation and a neutral element).

`definition` Definitions 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.

The OMDoc `definition` element supports several kinds of definition mechanisms specified in the `type` attribute currently:

`simple` The `FMP` (see section 2.6) contains an OpenMath representation of a logical formula that can be substituted for the symbol specified in the `for` attribute of the definition.

`inductive` The formal part is given by a set of **recursive equations** whose left and right hand sides are specified by the `pattern` and `value` elements in `requation` elements. The termination proof necessary for the well-definedness of the definition can be specified in the `just-by` attribute of the 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` slot of the definition. Again, the necessary proof of unique existence can be specified in the `just-by` attribute.

`obj` This can be used to directly give the concept defined here as an OpenMath object, e.g. as a group representation generated by a computer algebra system.

Figure 2.5 gives an example a (simple) definition of a monoid.

2.5.1 Abstract Data Types

All specification languages support mechanisms for specifying signature and axiom information using mechanisms similar to the ones discussed above, most also support **abstract data types** as a convenient shorthand for sets of inductively defined objects and recursive functions on these.

Abstract data types are a definition mechanism for sets that are inductively built up by a set of constructors. The `adt` element is a piece of special syntax for the concise statement of such sets that follows the model used in Casl [CoF98]. There, abstract data types declare a set of

```
<definition id="mon.d1" for="monoid" type="simple">
  <CMP>
    A structure (M, *, e), in which (M, *) is a semi-group
    with unit e is called a monoid.
  </CMP>
</definition>
```

Figure 2.9: A Definition of a monoid

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. if 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 a 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 **generateds**, since it only contains elements that are expressible in the constructors. If a abstract data type is **loose**, then it may contain other elements together with the ones generated by the constructors.

In OMDoc, we use the `adt` element to specify abstract data types. It has a `type` attribute that can have the values `free` and `generated` and `loose` and contains one or more `sortdef` elements. For instance, we could have expressed the theory of natural numbers, which we have imported from in Figure 2.8 by the construction using `adt` in Figure 2.5.1. The abstract data type `nat-adt`

```
<theory id="nat-thy">
 <commonname>natural number theory</commonname>
 <CMP>The Peano Axiomatization of Natural Numbers</CMP>
 <adt id="nat-adt" type="free">
  <sortdef id="pos-nat">
   <commonname>the set of positive natural numbers</commonname>
   <constructor id="succ">
    <commonname>The successor function</commonname>
    <argument sort="nat">
     <selector type="total" id="pred">
      <commonname>The predecessor function</commonname>
     </selector>
    </argument>
   </constructor>
  </sortdef>
  <sortdef id="nat">
   <commonname>the set of natural numbers</commonname>
   <constructor id="zero"/>
   <insort for="nat"/>
  </sortdef>
 </adt>
</theory>
```

Figure 2.10: The Natural numbers using `adt`

is free and has two sorts `pos` and `nat` for the (positive) natural numbers. Positive numbers are generated by the successor function on the natural numbers (all positive naturals are successors).

On pos, the inverse pred of succ is total. The set nat of all natural numbers is defined to be the union of pos and the constructor zero. Note that this definition implies the 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 nat is generated by zero and succ.

2.5.2 OMDoc Theories and OpenMath Content Dictionaries

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 like barshe.omdoc or ida.omdoc even reference OpenMath symbols they define themselves. Thus it is necessary to generate OpenMath content dictionaries from OMDoc documents, or develop an alternative mechanism to establish symbol identity in OMS. The generation of content dictionaries is already supported in the MBase system, but can also be achieved by writing specialized Xsl style sheets. For the purposes of this paper, we will only assume that one of these measures has been taken.

2.6 Mathematical Elements

Element	Attributes		D	Content
	Required	Optional	C	
FMP			–	assumption*,conclusion)|OMOBJ
assertion	id	type, theory	+	private*,symbol*,CMP*,FMP?
assumption	id		+	CMP*,OMOBJ?
concluson	id		+	CMP*,OMOBJ?
example	id	type, assertion, proof	+	CMP|OMOBJ
alternative-def	id, for, theory, entailed-by, entails, entailed-by-thm, entails-thm	type	+	CMP*,(FMP|requation*|OMOBJ)

Figure 2.11: The OMDoc Math Elements

We will now present the mathematical elements that are not integral parts of a theory, since they are optional (they can be derived from the material specified in the theory), they can be specified outside a theory element. We have the following elements:

FMP This is the general element for representing mathematical formulae as OpenMath objects[4], for instance the formula in Figure 1.2. As logical formulae often come as **sequents**, i.e.

[4]In fact OMDoc uses a variant of the OpenMath standard that extends their Xml representation by co-references. That is, then OMOBJ, OMA, OMBIND and OMATTR elements carry extra id and xref attributes that allow to reuse parts of a formula (see Figure 4.1 on page 37). Thus we can represent formulae as directed acyclic graphs preventing exponential blowup (see section C). Furthermore, OMDoc allows the attribute bracket-hint on OMA elements, which can take the values on (default) and off and gives the presentation module (see section 3.2) a hint on whether to put brackets.

Note that these extensions are licensed by the OpenMath standard, since pure OpenMath trees can be generated automatically from it.

a conclusion is 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 the conclusion is entailed by the assumptions in the current context. As a consequence, `<FMP><conclusion>`A`</conclusion></FMP>` is equivalent to `<FMP>`A`</FMP>`. The `assumption` and `conclusion` elements allow to specify the content by an OpenMath object (`OMOBJ`) or in natural language (using `CMP`s).

`assertion` This is the element for all statements (proven or not) about mathematical objects (see Figure 2.6). 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 them in the general `assertion` element and leave the type distinction to a `type` attribute. These type specifications in OMDoc documents should only be regarded as defaults, 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.

```
<assertion id="ida.c6s1p4.l1" type="lemma">
  <CMP> A semi-group has at most one unit.</CMP>
</assertion>
```

Figure 2.12: An assertion about semigroups

`alternative-def` Since there there can be more than one definition per symbol, OMDoc supplies the `alternative-def` element that can be specified outside a theory that can be specified outside a given theory. Of course, its `theory` attribute must be set to the theory of the symbol it defines.

An alternative definition for a symbol can only be added to a theory in a consistent way, if it is guaranteed that it is equivalent to the existing ones. Therefore, `alternative-def` has the attributes `entails`, and `entailed-by`, that specify `assertion`s that state this. It is an *integrity condition* of OMDoc that any `alternative-def` element references at least one `definition` or `alternative-def` 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 theorems. This way we can always reconstruct equivalence of all definitions for a given symbol.

`example` 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 counterexamples for conjectures). Therefore, examples are given status as primary objects in OMDoc. Conceptually, we model an example for a mathematical concept $\mathbf{C}$ as a triple $(\mathcal{W}, \mathbf{A}, \mathcal{P})$, where $\mathcal{W} = (\mathcal{W}_1, \ldots, \mathcal{W}_n)$ is an n-tuple of mathematical objects, $\mathbf{A}$ is an assertion of the form $\mathbf{A} = \exists W_1 \ldots W_n.\mathbf{B}$, and $\mathcal{P}$ is a proof that shows $\mathbf{A}$ by exhibiting the witnesses $\mathcal{W}_i$ for W_i. The example $(\mathcal{W}, \exists W_1 \ldots W_n.\neg\mathbf{B}, \mathcal{P})$ is a counter-example to a conjecture $\mathbf{T}{:=}\forall W_1 \ldots W_n.\mathbf{B}$, and $(\mathcal{W}, \mathbf{A}, \mathcal{P}')$ a supporting example for $\mathbf{T}$.

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

- `for` (for what concept or assertion is it an example),
- `type` (one of the keywords `for` or `against` for the function)
- `assertion` (a reference to the assertion $\mathbf{A}$ mentioned above)
- `proof` (a reference to the constructive proof $\mathcal{P}$)

Consider for instance the structure $\mathcal{W} := (A^*, \circ)$ of the set of words over an alphabet A together with word concatenation $\circ$. Then $(\mathcal{W}, \exists W.\texttt{monoid}(W), \mathcal{P}_1)$ is an example for the concept of a monoid (with the empty word as the neutral element), if e.g. $\mathcal{P}_1$ uses $\mathcal{W}$ to show the existence of W. The example $(\mathcal{W}, \exists V_{\texttt{monoid}}.\neg\texttt{group}(V), \mathcal{P}_2$ and a proof that uses $\mathcal{W}$ as a witness for V, it is a counterexample to the conjecture $\mathbf{C} := \forall V_{\texttt{monoid}}.\texttt{group}(V)$, since $\mathbf{Q} \to \neg\mathbf{C}$. Figure 2.6 gives the OMDoc representation of this example of an example.

```
<example id="mon.ex1" for="monoid" type="for"
         assertion="strings-are-monoids" proof="sam-pf">
  <CMP>The set of strings with concatenation</CMP>
  <OMOBJ><OMS cd="simple-monoids" name="strings"/></OMOBJ>
</example>
<example id="mon.ex2" for="monoid" type="against"
         assertion="monoids-are-groups" proof="mag-pf">
  <CMP>The set of strings with concatenation is not a group</CMP>
  <OMOBJ><OMS cd="simple-monoids" name="strings"/></OMOBJ>
</example>
```

Figure 2.13: An OMDoc representation of an example

2.7 Proofs

Proofs form an essential part of mathematics and modern sciences, conceptually they are a representation of un-controversial evidence for the truth of an *assertion*. As a consequence, some of the knowledge about given objects of interest can be inferred from simpler assumptions about it. Thus the role of proofs is twofold, they allow to push back the assumptions about the world to simpler and simpler assumptions (often called a model), and they allow to test the model by deriving consequences of these basic assumptions that can be tested against the data.

The question of what exactly constitutes a proof has been controversially discussed. The clearest (an 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 modern mathematics the notion of a proof is more flexible, 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). Since this process is extremely tedious, this option is very seldomly exercised. 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 (but still well above the formal level) account for skeptics or novices. Moreover 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 provide a common markup system for them, OMDoc concentrates on the tree/DAG-like structure of proofs and supports a proof format whose structural and formal elements are derived from the $\mathcal{PDS}$ structure developed for semi-automated theorem proving (satisfying the logical side), but also allows natural language representations at every level (allowing for natural representation of mathematical vernacular.) The **P**roof plan **D**ata **S**tructure ($\mathcal{PDS}$) was introduced in the ΩMEGA [BCF+97] system to facilitate hierarchical proof planning

and proof presentation at more than one level of abstraction. The $\mathcal{PDS}$ is a DAG of nodes (the proof steps) that contain a representation of the local subgoal or assertion and a justification by either a logical inference rule or higher-level evidence for the truth of the assertion. 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 hearer was unconvinced. Conceptually, both possibilities are equivalent, since the method can be used to compute the subproof (called its **expansion**).

In a $\mathcal{PDS}$, 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 CMPs) or as logical formulae (in FMPs). 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 there 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		+	symbol*,CMP*, (hypothesis|derive|metacomment)*, conclude
proofobject	id, for, theory		+	CMP*,OMOBJ
metacomment	id		−	CMP*
hypothesis	id		−	symbol*,CMP*,FMP?
derive	id		−	CMP*,FMP?,method?,premise*, (proof|proofobject)?
conclude	id		−	CMP*,method?,premise*, (proof|proofobject)?
method			−	(ref|OMSTR),parameter*
parameter			−	OMOBJ
premise	xref		−	EMPTY

Figure 2.14: The OMDoc Proof Elements

Let us now come to the concrete markup scheme provided by OMDoc (see Figure 2.7 for an overview). Due to the complex structure of proofs in the $\mathcal{PDS}$, we cannot directly utilize the tree-like structure provided by XML, but use cross-referencing (see the discussion in section 2.2). Proofs are specified by `proof` elements in OMDoc that has 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[5]). Note that there can in general 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 two major kinds of OMDoc elements: `derive` specify normal proof steps that derive a new assertion from already known ones, from assertions or axioms in the current theory, or from the assumptions of the assertion that is under consideration

[5]Thereby making it possible to specify expansions of justifications and thus hierarchical proofs.

in the proof. We will explain them in detail below. `hypothesis` elements allow to specify local assumptions, well-known from calculi like Gentzen's Natural Deduction calculus [Gen35]. They allow the hypothetical reasoning discipline need 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 (it is not accessible outside this subproof). The `conclude` element is a variant of `derive` that does not contain a `FMP`, 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 `FMP` and in mathematical vernacular, the last step is often explicitly verbalized to mark the end of the proof. Similarly, OMDoc supplies the `metacomment` element to allow for intermediate text that does not have a logical correspondence, but e.g. guides the reader of the proof.

The `derive` elements can contain the following child elements (in this order)

`CMP` This gives the natural language representation of the proof step.

The rest of the children form the formal content of the derive step, together, they represent the information present e.g. in a $\mathcal{PDS}$ node.

`FMP` A formal representation of the assertion made by this proof step, they contain `CMP` and `FMP` elements. 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 2.7, 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 OpenMath symbol representing a proof method or inference rule that justifies the assertion made in the `FMP` element.

`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 (which can have similar expansions in embedded `proof` environments again).

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).

```
<derive id="barshe.2.1.2.proof.a.proof.D2.1">
  <CMP>By <OMOBJ><OMS cd="barshe" name="alg-prop-reals.A2"/></OMOBJ>
       we have z + (a + (−a)) = a + (−a)
  </CMP>
  <conclusion>(z + a) + (−a) = z + (a + (−a))</conclusion>
  <method><OMS cd="omega-base-calc" name="foralli*"/>c
    <parameter><OMOBJ><OMV name="z"/></OMOBJ></parameter>
    <parameter><OMOBJ><OMV name="a"/></OMOBJ></parameter>
    <parameter>−a</parameter>
  </method>
  <premise xref="alg-prop-reals.A2"/>
</derive>
```

Figure 2.15: A `derive` proof step

```
<proof id="t1_p1" for="t1" theory="sets">
  <conclude id="t1_p1_c">
    <CMP> We prove the assertion by a case analysis over <ref xref="t1_a1"/>.</CMP>
    <proof id="t1_p1_c_p" for="t1_p1_c" theory="sets">
      <derive id="l1">
         <CMP> If a ∈ U, then a ∈ U ∪ V.</CMP>
         <FMP>
         <assumption id="l1_A">
            <CMP> a ∈ U</CMP>
              <OMOBJ/>
         </assumption>
         <conclusion id="l1_C">
            <CMP> a ∈ U ∪ V.</CMP>
            <OMOBJ/>
          </conclusion>
         </FMP>
         <method><ref theory="bla" name="Method-1"/></method>
         <proof id="l1_p" for="l1" theory="sets">
           <conclude id="l1_p_d1">
             <CMP>a ∈ U ∪ V by definition of  ∪.</CMP>
           </conclude>
         </proof>
      </derive>
      <derive id="l2">
        <CMP> If a ∈ V, then a ∈ U ∪ V.</CMP>
        <FMP>
            <assumption id="l2_A">
            <CMP> a ∈ V</CMP>
            <OMOBJ/>
            </assumption>
            <conclusion id="l2_C">
              <CMP> a ∈ U ∪ V.</CMP>
              <OMOBJ/>
            </conclusion>
          </FMP>
          <method><ref theory="bla" name="Method-2"/></method>
          <proof id="l2_p" for="l2" theory="sets">
            <conclude id="l2_p_d1">
              <CMP>a ∈ U ∪ V by definition of  ∪.</CMP>
            </conclude>
          </proof>
      </derive>
      <conclude id="t1_p_c_c1">
        <CMP> We have considered both cases from <ref xref="t1_a1"/>,
          so we have a ∈ U ∪ V
        </CMP>
      </conclude>
    </proof>
  </conclude>
</proof>
```

Figure 2.16: A proof by cases

2.8 Complex Theories and Inheritance

In OMDoc, we support the structured specification of theories; we build upon the technical notion of a **development graph** [Hut99], since this supplies a simple set of primitives for structured specifications and also supports management of theory change. Furthermore, it is logically equivalent to a large fragment of the emerging CASL standard [CoF98] for algebraic specification (see [AHMS00]).

Not all definitions and axioms need to be explicitly stated in a theory; they can be inherited from other theories, possibly transported by signature morphism. The inheritance information is stated in an `imports` element.

`imports` This element has a `from` attribute, which specifies the theory which exports the formulae.

For instance, given a theory of monoids using the symbols `set`, `op`, `neut` (and `axiom` elements stating the associativity, closure, and neutral-element axioms of monoids), a theory of groups can be given by the theory definition using `import` in Figure 2.8.

Furthermore, 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[6].

```
<theory id="group">
  <imports id="group.import" from="monoid" type="global"/>
  <axiom><CMP> Every object in
    <OMOBJ><OMS cd="monoid" name="set"/></OMOBJ> has an inverse.
  </CMP></axiom>
</theory>
```

Figure 2.17: A theory of groups based on that of monoids

`morphism` The morphism is a recursively defined function (it is given as a set of recursive equations using the `requation` element, described above). It allows to carry out the import of specifications modulo a certain renaming. With this, we can e.g. define a theory of rings given as a tuples $(R, +, 0, -, *, 1)$ 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 the $\{M \mapsto R^*, \circ \mapsto *, e \mapsto 1\}$, where R^* is R without 0 (as defined in the theory of monoids). Figure 2.8 gives the OMDoc representation of this exercise.

`inclusion` This element can be used to specify applicability conditions on the import construction. Consider for instance the situation given in Figures 2.8 and 2.8, where the theory of lists of natural numbers is built up by importing from the theories of natural numbers and lists (of arbitrary elements). The latter imports the element specification from the parameter theory of elements, thus to make the actualization of lists to lists of natural numbers, all the symbols and axioms of the parameter theory must be fulfilled by the natural numbers. For instance if the parameter theory specifies an ordering relation on elements, this must also be present in theory `Nat`, and have the same properties there. These requirements can be specified in the `inclusion` element of OMDoc. Due to lack of space, we will not elaborate this and refer the reader to [Hut99].

Figure 2.18: A Structured Specification of Lists

Finally, there are OMDoc elements that allow to augment the structure of the theory-graph. We have already seen the possibility to define (parts of) theories by so-called **theory morphism** specified in `imports` and `include` elements above. Following Hutter's development graph [Hut99], we can use the knowledge about theories to establish so-called **inclusion morphisms** that establish the source theory as included (modulo renaming by a morphism) in the target theory. This information can be used to add further structure to the theory graph and help maintain the knowledge base with respect to changes of individual theories.

An `axiom-inclusion` element contains a `morphism` (see section 2.8), and the attributes `from` and `to` specify the source and target theories. For any axiom in the source theory there must be an assertion in the target theory (whose `FMP` is just the image of the `FMP` of the axiom under the

[6] Of course, if we hide a sort symbol, we also have to hide all symbols using it (see [CoF98] for details)

```
<theory id="Param">
  <symbol id="Elem" type="sort"/>
  <symbol id="ord"/>
  <axiom ... ord is a partial order on Elem ... /axiom>
</theory>

<assertion id="geq-ord" theory="nat-thy">
  <CMP><OMOBJ><OMS name="geq" cd="nat"/></OMOBJ> is a
    partial order on <OMOBJ><OMS name="nat" cd="nat"/><OMOBJ>
  </CMP>
<assertion>

<theory id="List">
  <imports id="List.im" type="global" from="Param"/>
  <symbol id="List-sort" type="sort"/>
  <symbol id="cons"/><symbol id="nil"/>
  <symbol id="ordered"/>
</theory>

<theory id="nat-list.thy">
  <imports id="nat-list.im-nat"
           type="global" from="nat-thy"/>
  <imports id="nat-list.im-Element"
           type="local"  from="List">
    <morphism id="elem-nat">
      <requation>
        <pattern><OMS cd="Param" name="Elem"/></pattern>
        <value>
          <OMOBJ><OMS cd="nat.thy" name="Nat"/></OMOBJ>
        </value>
      </requation>
    </morphism>
  </imports>
  <inclusion for="elem-nat-incl"/>
</theory>

<axiom-inclusion id="elem-nat-incl"
                 from="nat.thy"
                 to="Param" by="ord-nat">
  <morphism id="elem-nat-incl-morph"
            base="elem-nat"/>
</axiom-inclusion>
```

Figure 2.19: A theory of Lists of Natural Numbers

morphism) with a proof. These are represented by an empty `by` element, which has the attributes `axiom`, `assertion`, and `proof` with the obvious meanings.

A `theory-inclusion` is a global variant of `axiom-inclusion` that can be obtained as a path of `axiom-inclusion`s (or other `theory-inclusion`) which are specified in the `by` attribute.

2.9 Auxiliary Elements

In this section we will present OMDoc elements that are not strictly mathematical content, but have useful functions mathematical documents or knowledge bases.

2.9.1 Exercises

Exercises are vital parts of mathematical textbooks. Mathematical exercises are often given as questions or multiple-choice exercises. In OMDoc, we use the `exercise` element for this. The Questions are represented in the `CMP`, the solution and a hint are (optionally) given using the `solution` and `hint` element, which can contain a `CMP` or a `FMP`. A special case of this is the case, where the questions 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 2.9.1) are represented by a list of `mc` elements. These

```
<theory id="ring">
  <symbol id="ring.set"/>
  <symbol id="ring.plus"/>
  <symbol id="ring.times"/>
  <symbol id="ring.zero"/>
  <symbol id="ring.one"/>
  <symbol id="ring.setstar"/>
  <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 2.20: A theory of rings

2.9.3 Applets in OMDoc

`omlet` elements contain OMDoc specifications of applets (program code that can in some way executed during document manipulation). `omlet`s generalize the well-known *applet* concept in two ways: The computational engine is not restricted to *plug-ins*, of the browser (current servlet technology can be used and specified using `code` and `omlet` elements in OMDocs) and the program code can be specified and distributed more easily. Making document-centered computation easier to manage.

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 (in our case the $\mathcal{LOUI}$ or ΩMEGA services).

`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

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

Figure 2.21: The OMDoc Auxiliary Elements for Exercises

```
<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="F"><CMP>No, 1 ∗ 1 = 3 and not 1</CMP></answer>
  </mc>
  <mc><choice><CMP>1/3</CMP></choice>
      <answer verdict="T"></answer>
  </mc>
  <mc><choice><CMP>It has no unit.</CMP></choice>
      <answer verdict="F"><CMP>No, try another answer</CMP></answer>
  </mc>
</exercise>
```

Figure 2.22: An Exercise

OMDoc). This indirection allows us to reuse the machinery for storing code in OMDocs. For a simple example see Figure 2.9.3.

`argstr` allows to specify an (optional) argument string for function, so that the program in the can be kept general. A call to the $\mathcal{LOUI}$ interface, would then have the form in Figure 2.9.3. Here, the code in the `code` element `sendtoloui` (which we have not shown) would be Java code that simply sends the argstr to $\mathcal{LOUI}$'s remote control port.

The expected behavior of the `omlet` can be implemented in the Xsl style sheet, that in the case of e.g. translation to Mozilla will put the `callmint` code directly into the generated `html`.

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

Figure 2.23: The OMDoc Auxiliary Elements for non-Xml Data

```
<code id="callmint">
  <input>None</input>
  <output>The result</output>
  <effect>None</input>
  <data><![CDATA[...  the call-mint code goes here ...]]></data>
</code>
<derive id="monp_1">
  <CMP> <omlet type="js" function="callMint">Intros.</omlet></CMP>
  <method><OMS name="Intros" cd="COQ"/></method>
</derive>
```

Figure 2.24: An Omlet

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

Figure 2.25: An `omlet` calling an external process

Chapter 3

Processing OMDoc

Before we address the issues pertaining to processing OMDoc documents, we have to be more explicit about the intended meaning of OMDoc documents. In the last section, we have specified the structure of text- and math elements, and we have motivated this by explaining their intended meaning. Now, let us explicitly state, what does not have meaning in OMDoc documents,

Whitespace (including line-feeds) has no meaning, and can therefore added and deleted without effecting the semantics.

Xml **comments** (i.e. anything between `<!--` and `-->`) are not considered either. Suitable OMDoc text elements should be used for any comments that are relevant to the reader of an OMDoc document.

the `ignore` element is a variant of comments provided by OMDoc. Unlike the Xml comments it is read by the Xml parsers and can survive the transformation by an Xsl style sheet (this is also their reason for existence.)

`CDATA` **sections** are an Xml device for including material in the document that would not pass the Xml parser, since it e.g. contains angle brackets or elements that are unbalanced or not defined in the DTD. Since for the Xml parser, the escaped and CDATA form are equivalent, for instance `<[CDATA[a<b<sup>3</sup>]]` and `a&lt;b&lt;sup&gt;3&lt;/sup&gt;` are equivalent, you should not rely on any particular form.

most OpenMath **attributions** In contrast to the text elements, the formal mathematical properties contain logical formulae that are specified only for content markup. The background for this is that the semantic status of OpenMath attributions (via the `OMATTR` element) is somewhat unclear. The OpenMath standard states that attributions can, but need not be considered.

Consequently, OMDoc compliant applications should preserve OpenMath attributions in OpenMath objects embedded in OMDoc text elements, so that they can be transmitted to other OpenMath applications. In particular, presentation attributes should be preserved.

Applications may disregard all OpenMath attributions in OpenMath objects in `FMP` elements, that are not meaningful in the logical system specified in the `logic` attribute of the `FMP` element. In particular, presentation attributes should be disregarded, since they are irrelevant for the logical content.

3.1 Transforming OMDocs by Xsl 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 Xsl [Dea99] **style sheet** to produce other representations from OMDocs. These style sheets can be used to several tasks in maintaining OMDoc, such as for instance extracting stand-alone catalogs from OMDocs (see `standalone-catalogue.xsl`), converting other Xml-based input formats into OMDoc (e.g. `cd2omdoc.xsl` for converting OpenMath content dictionaries into OMDoc format), or – in the future – even migrating between different versions of OMDoc.

Note that the material discussed in this section is quite preliminary and under development, since the OMDoc version 1.0 format has just been released. It should be considered as proof-of-concept, development of production versions has just begun. For an up-to-date account, see `http://www.mathweb.org/omdoc` under the Xsl and tools headings.

Another related task is to convert OMDoc to the input languages of mathematical software systems like theorem provers or computer algebra systems (not all of these systems) directly understand OpenMath or OMDoc yet). Usually, the task of writing a style sheet for such a conversion is a relatively simple task. At the moment we have style sheets the Ωmega, InKa, TPS and *λClam* systems (they can be found at `http://www.mathweb.org/omdoc/xsl`). 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 way (which we have taken 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[1] in OMDoc.

Finally, there is the task of transforming OMDoc into human-readable formats, i.e. to regenerate all the (implicit) notation conventions that we had talked about in the introduction. We speak of of OMDoc **presentation** for this task.

Due to the complex nature of the task, only part of it can actually be performed by Xsl 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 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 Xsl style sheets that transforms the resulting specialized OMDoc into the respective output format, taking into account 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 [Fie99].

At the moment, we have Xsl style sheets to convert OMDoc to HtML, and LaTeX. 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 `omdoc2tex.xsl` for LaTeX. They share a large common code base `omdoc2share.xsl`, basically the first two only redefine some format-specific things. They also share the internationalization part, which uses `locale.xml` as a keyword table. This file contains all the keywords necessary for presenting the OMDoc elements discussed so far, e.g. "Lemma" and "Lemme" for `<assertion type="lemma">` in English and French. In order to obtain the presentation in your language, set the `language` parameter to the appropriate ISO 639 two-letter country codes (`en` $\widehat{=}$ English, `de` $\widehat{=}$ German, `fr` $\widehat{=}$ French, `nl` $\widehat{=}$ Dutch...) when invoking the style sheet. Of course, mathematical texts elements (see section 2.4) in the OMDoc also have to exist in translated form, in order to get a fully localized version of the OMDoc.

Since the presentation of OpenMath symbols cannot be deduced from general principles it is specified in the OMDoc that introduces them. For a given OMDoc (say `usesall.omdoc`) and a

[1] It is sometimes a problem with these systems that they only store the name of a symbol (logical constant) and not its home theory. Sometimes it is also a problem that the 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.

output format (say `html`), we use a specialized style sheet `expres.xsl` to generate a style sheet `usesall-html.xsl` that contains specialized XSL templates for the OPENMATH symbols used in `usesall.omdoc`:`expres.xsl` looks up their defining OMDOCs (using the catalogue mechanism described in section 2.2) and derives the templates from the `presentation` elements there (see the next section). Of course, it also takes into account the presentation elements in `usesall.omdoc` itself. The result is a standalone XSL style-sheet `usesall-html.omdoc` that can be used to present `usesall.omdoc` in HTML.

3.2 Specifying the Presentation of OPENMATH Symbols

Element	Attributes		D	Content
	Required	Optional	C	
presentation	id, for	fixity, parent, lbrack, rbrack, separator, bracket-style	–	use*
use	format	lbrack, rbrack, separator, crossref-symbol	–	ANY

Figure 3.1: The OMDOC Auxiliary Elements for Presenation Information

The mathematical concepts and symbols introduced in an OMDOC document (`symbol` elements) 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. Since we use XSL style sheets for this task, we want to product XSL templates like the shown in Figure 3.2. This style

```
<xsl:template match="OMA[OMS[position()=1 and
                             @name='monoid' and
                             @cd='ida.monoid']]">
  (<xsl:apply-templates select="*[2]"/>,
   <xsl:apply-templates select="*[3]"/>,
   <xsl:apply-templates select="*[4]"/>)\in{\bf MON}
</xsl:template>
```

Figure 3.2: An XSL template for the symbol in Figure 2.5

sheet information will cause an OPENMATH expression

```
<OMA>
  <OMS cd="ida" name="monoid"/><OMV name="M"><OMV name="o"><OMV name="e">
</OMA>
```

to be rendered as $(M, o, e) \in \mathbf{MON}$ in a TEX or LATEX document derived from `ida.xml` via a suitable XSL style sheet.

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. The meta-stylesheet `expres.xsl` would then simply

have to copy them into the into `usesall-html.xsl`. We could even combine templates for more that one format in one `presentation` element.

Obviously, all of these templates share a great deal of structure, moreover, hand-coding XSL-templates is a tedious and error-prone process, 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 in the `presentation` element and `expres.xsl` generates the needed XSL templates from that.

The OMDOC `presentation` element uses its attributes to specify the information that is common to all output formats, such as whether a symbol is written infix, and the `use` element for information that is format-specific. Therefore `use` carries a `format` attribute that specifies for which output format the presentation is intended; we use the `TeX` for TEX and LATEX, `mathml` for MATHML, `html` for HTML, `mathematica` for MATHEMATICA notebooks. See `http://www.mathweb.org/omdoc/xsl.html` for available formats. Some of the attributes of the `presentation` element can also occur in the `use` element, the meaning of this is that the `use` attributes overwrite the `presentation`-attributes. Let us now come to the attributes themselves.

The presentation element has the attributes

`mode` This optional attribute is not used at the moment, it is provided to specify the mode attributes in the XSL templates. This allows to specify different notational conventions for symbols.

`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 put 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 one of `lisp` (LISP-style brackets) and `math` (generic mathematical function notation). We have the following combinations:

attribute combination	yields
`prefix` and `lisp`	$(f\ 1\ 2\ 3)$
`postfix` and `lisp`	$(1\ 2\ 3\ f)$
`prefix` and `math`	$f(1,2,3)$
`postfix` and `math`	$(1,2,3)f$

The default value of this attribute is `math`.

`parent` This attribute specifies parent element, in which the symbol plays the head role (it can be one of `OMA`, `OMBIND`, and `OMATTR`). In the examples above, we have assumed the head to be an `OMA`, since we treated function application. It can also be an `OMBIND`, as in the case of a quantifier in Figure 3.2.

`lbrack` **and** `rbrack` These two attributes can be used to specify the brackets to be used in presentation of the complex expression. They will only be used, if the `bracket-hint` (see footnote 4 on page 4) attribute of the OPENMATH expression in the `parent` is set to `on` (which is the default).

`separator` This specifies the separator to be used for separating the arguments. The default for this is the comma to give the usual mathematical function notation together with the default `bracket-style`=`math`, in the table above, we have substituted it with `separator=" "` for `bracket-style`=`lisp`.

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

`crossref-symbol` This attribute is also private to the `use` element. It specifies which parts of the symbol presentation elements to attach the cross-references to: In some formats like HTML, and recently also in LaTeX (thanks to the `hyperref.sty` package), there it may be good 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 the notation for (a, b) pairs, there 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 (the brackets, the separator, and the print-form of the symbol).

`precedence` is reserved to specify the operator precedence. This will be used in future versions of he presentation style sheets to eliminate brackets according to precedence. For details see future versions of this report or `http://www.mathweb.org/omdoc/xsl.html`.

```
<presenation for="forall" parent="OMBIND"
                          separator=".">
  <use format="TeX">\forall</use>
  <use format="html">&#8704;</use>
  </use>
</presentation>
```

```
<OMBIND>
  <OMS cd="quant1" name="forall"/>
  <OMBVAR><OMV name="X"/></OMBAR>
  <OMS cd="logic1" name="true"/>
</OMBIND>
```

using the left presentation element on the right OPENMATH expression yields

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

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

Figure 3.3: `presentation` with `parent`=`OMBIND`

Note that there can be more than one `use` element per `presentation` element allowing to reuse the attributes in the `presentation` element. This is just a notational convenience that is equivalent to copying the `presentation` element and using single `use` elements in them.

Chapter 4

Creating OMDoc representations from LaTeX

In this chapter we describe the LaTeX style `latex2omdoc.sty` that allows to create OpenMath objects and OMDoc representations from inside LaTeX documents. This is at the moment the only way to author OMDoc other than directly typing it.

This LaTeX-based approach is particularly useful when migrating existing mathematical texts from LaTeX representation, e.g. in papers to OMDoc, or as a maintenance tool for authors that are more efficient/used to generating LaTeX than Xml.

One of the primary advantages of using LaTeX to generate OMDoc documents over directly writing them in a text editor is that LaTeX allows to define macros (see 4.2). Thus one can reuse frequently recurring patterns in OpenMath objects and OMDoc structures instead of retyping them every time.

Moreover the `latex2omdoc.sty` style file can be used as a migration tool from LaTeX texts to OMDoc documents. For instance it is possible to transport much of the structure that is present in a well-structured mathematical LaTeX document (definition, theorem, proof) into OMDoc simply by redefining the LaTeX macros used structuring the original document (see section 4.4).

In the next section, we will present an abbreviation mode for OpenMath objects that allows to specify these by single characters. Then (section 4.3), we will use this to discuss the facilities for creating OMDoc documents. Finally, in appendix D, we will work an example text that shows most of facilities in action.

4.1 OpenMath Objects

In this section, we will first briefly review the possible types of OpenMath objects and show the two LaTeX representations provided by the `latex2omdoc` style (full LaTeX mode and abbreviation mode; see Figure 4.1).

Both modes write the generated OpenMath representations to an auxiliary file that can be specified by the user in the `omoutput` environment. `\begin{omoutput}[test.xml]` will prepare a file `test.xml` for output and direct all OpenMath output to it. The default for the optional argument is the name of the top-level LaTeX file with extension `xml`, `\end{omoutput}` will close the file, so that it can for instance directly be inserted into the LaTeX itself, say with a `\verbatiminput` statement from the `moreverb.sty` style (see also the source of this document in section 4.4). Note that the second call to an `omoutput` environment with the same file name will overwrite a first one.

The `omoutput` environment has a variant `OMoutput`, which wraps the output with an Xml declaration, and declaration of the document type definition so that the result can be used as a standalone Xml file that can e.g. be validated. For this it is necessary to specify the specify the URL of the Xml document type definition (DTD) with the `\dtdurl` command, the default setting

is `omdoc.dtd`, i.e. assuming that the dtd is in the same directory as the top-level LaTeX file. For instance `\dtdurl{http://www.mathweb.org/dtd/omdoc.dtd}` will use the current distribution version.

Note that `omoutput` and `OMoutput` environments can be nested (both individually and among each other up to 15 levels), so that it is possible to generate more than one file at a time. Since the DTD will probably not vary between these, it is probably good to specify `\dtdurl` in the document header.

The abbreviation mode can be turned on by the environment given by the bracketings `\<` and `\>` or `\(` and `\)`. The latter differs from the former in that it additionally wraps an `<OMDOC>` element bracketing in the output (this is often a useful abbreviation).

Figure 4.1 shows an example of a λ-abstraction $\lambda XY.X^{-Y}$ and Figure 4.1 an attribuition of the variable X with the type $(\iota \to o)$ and the color green.

category	XML representation	full LaTeX	abbreviation
symbol	`<OMS cd="CD" name="N"/>`	`\oms{CD}{N}`	`#CD:N:`
variable	`<OMV name="N"/>`	`\omv{N}`	`$N:`

application	`<OMA>`
	`  <OM*>...<OM*>`
	`</OMA>`
		`\begin{oma}`
		`  <OM*>...<OM*>`
		`\end{oma}`
			`(<OM*>`
			`  ...<OM*>)`

binding structure	`<OMBIND>`
	`  <OMS.../>`
	`  <OMBVAR>`
	`  <OMV*>...<OMV*>`
	`  </OMBVAR>`
	`  <OM*>`
	`</OMBIND>`
		`\begin{ombind}`
		`  \oms{a}{b}`
		`  \begin{ombvar}`
		`  \omv{x}...\omv{z}`
		`  \end{ombvar}`
		`  <OM*>`
		`\end{ombind}`
			`{#a:b:`
			`  $x:$z:`
			`.<OM*>}`

attribution	`<OMATTR>`
	`  <OMATP>`
	`  <OMS*><OM*>...`
	`  <OMS*><OM*>`
	`  </OMATP>`
	`  <OM*>`
	`</OMATTR>`
		`\begin{omattr}`
		`  \begin{omatp}`
		`  \oms{a}{b}<OM*>`
		`  \oms{c}{d}<OM*>`
		`  \end{omatp}`
		`  <OM*>`
		`\end{omattr}`
			`[#a:b:<OM*>,`
			`  #c:d:<OM*>`
			`|<OM*>]`

`<OM*>` stands for an arbitrary OpenMath object and `<OMV*>` stands for a variable or an attribuited variable.

Figure 4.1: LaTeX representations of OpenMath objects

Note that these macros do not check for validity of the XML output, this is left to an XML validator. Moreover, little work has gone into generating good error messages. In particular, some of the macros in the abbreviated mode make assumptions about the OpenMath-well-formedness; most notably, the start tag `{` for an OpenMath binding object only works, iff the next object is a valid OpenMath symbol, i.e. of the form `#cd:name:`.

`latex2omdoc.sty` also supports OMDoc's `id`/`xref` extension to OpenMath objects (see footnote 4). In short, this allows to re-use OpenMath objects by reference. For instance, the left hand side OMDoc element in Figure 4.1 can be realized by the expression below it and has the same meaning as the OpenMath object on the right hand side. `latex2omdoc.sty` provides two primitives for this: `~` for adding `id` attributes to the next element. This takes one token as argument, so if the attribute needs more than one letter, it has to be grouped by `<` and `>`. `^` for adding `xref` attributes to the next element, the same token conventions as above apply.

```
\begin{omobj}
  \begin{ombind}
    \omsref{lambda}
    \begin{ombvar}
      \omv{X}
      \omv{Y}
    \end{ombvar}
    \begin{oma}
      \omsref{exp}
      \omv{X}
      \begin{oma}
        \omsref{inv}
        \omv{Y}
      \end{oma}
    \end{oma}
  \end{ombind}
\end{omobj}
```

```
<OMOBJ>
  <OMBIND>
    <OMS cd="ecc" name="lambda"/>
    <OMBVAR>
      <OMV name="X"/>
      <OMV name="Y"/>
    </OMBVAR>
    <OMA>
      <OMS cd="arith1" name="exp"/>
      <OMV name="X"/>
      <OMA>
        <OMS cd="arith1" name="unary-minus"/>
        <OMV name="Y"/>
      </OMA>
    </OMA>
  </OMBIND>
</OMOBJ>
```

$\lambda XY.X^{-Y}$

```
\({#ecc:lambda:$X:(#arith1:unary-minus:$Y:)}\)
```

Figure 4.2: The representations of the functions $\lambda XY.X^{-Y}$

```
\begin{omattr}
  \begin{omatp}
    \oms{ecc}{type}
    \begin{oma}
      \oms{mltt}{funtype}
      \oms{typeind}
      \oms{typebool}
    \end{oma}
    \oms{colors}{colorattr}
    \oms{colors}{green}
  \end{omatp}
\end{omattr}
```

```
<OMATTR>
  <OMATP>
    <OMS cd="ecc" name="type"/>
    <OMA>
      <OMS cd="mltt" name="funtype"/>
      <OMS cd="types" name="individuals"/>
      <OMS cd="types" name="booleans"/>
    </OMA>
    <OMS cd="colors" name="colorattr"/>
    <OMS cd="colors" name="green"/>
  </OMATP>
  <OMV name="X"/>
</OMATTR>
```

$X_{\iota \to o}^{green}$

```
\<[#ecc:type:,
    (#mltt:funtype:#types:ind:#types:bool:);
   #colors:colorattr:,#colors:green:
   |$X:]\>
```

Figure 4.3: An OpenMath attribuition

```
<OMA>
  <OMV name="f"/>
  <OMA id="a">
    <OMV name="g"/>
    <OMV name="x" id="aa"/>
    <OMV xref="2"/>
  </OMA>
  <OMA xref="first"/>
</OMA>
```

```
<OMA>
  <OMV name="f"/>
  <OMA>
    <OMV name="g"/>
     <OMV name="x"/>
    <OMV name="x"/>
  </OMA>
  <OMA>
    <OMV name="g"/>
    <OMV name="x"/>
    <OMV name="x"/>
  </OMA>
</OMA>
```

```
\(($f:~a($g:~<aa>$x:^<aa>$:^a())\)
```

Figure 4.4: Reusing OpenMath objects by reference

4.2 Defining Macros

One of the primary advantages of using LaTeX to generate OMDoc documents over directly writing them in a text editor is that LaTeX allows to define macros. They can be simply be defined using LaTeX's `\(re)newcommand`, `\(re)newenvironment` facilities. Currently, `latex2omdoc.sty` only allows to use the full-form macros in definitions. For instance, we can use the definitions to simplify the attribuition example above to `\<\oftypeiogreen<$X:>\>`.

```
\newcommand{\typeio}%             the type of unary predicates on individuals
{\begin{oma}\oms{mltt}{funtype}\oms{types}{individuals}\oms{types}{booleans}\end{oma}}

\newcommand{\colgreen}%           the attribute value pair for the color green
{\oms{colors}{colorattr}\oms{colors}{green}}

\newcommand{oftypeiogreen}[1]%    attribuition stating that the content is of \typeio
{\begin{omattr}\begin{omatp}\oms{ecc}{type}\typio\colgreen\end{omatp}#1\end{omattr}}
```

4.3 OMDoc Elements

In Figure 4.3, we have given an overview over the macros that write OMDoc material for the Xml file. Since these generate OMDoc output, they must be enclosed in an `omdoc` environment. This is a variant of the `OMoutput` environment that also wraps the content into an `<omdoc>` element that is the Xml top-level element for OMDoc documents.

Note that as in the case of the OpenMath macros described in section 4.1, they do not check for validity of the Xml output, this is left to an Xml validator.

4.4 Useful Redefinitions of commonly used Macros

The `latex2omdoc.sty` style file can be used as a migration tool from LaTeX texts to OMDoc documents. In this section, we will present a set of useful macros that can be used transport much of the structure that is present in a well-structured mathematical LaTeX document (definition, theorem, proof) into OMDoc.

For instance there are the environments `theorem`, `lemma`, `corollay`, `conjecture`, `definition`, `remark` that allow to use the file `migrationex.tex`, in Figure 4.4 to generate the OMDoc file `migration.xml` in Figure 4.4, generating the DVI output in Figure 4.4 in the process. As we can see for migrating the a file with LaTeX markup like the first part (above the comment) of `migrationex.tex` in Figure 4.4, can be migrated by using the corresponding migration macros provided by `latex2omdoc.sty`, and adding the `omverb` environment for all parts that should go to the Xml file verbatim.

Of course we have to specify the language used in the texts in the document using a `\ommiglanguage` statement and the OMDoc theory by `\ommigtheory`. These two details go into the Xml file and cannot be deduced by `latex2omdoc.sty` itself.

4.5 Wishlist

This style file is far from finished, so please help it, if you can. The wishlist for `latex2omdoc.sty` includes the following items

1. Indentation in the Xml file, to make the output more readable.
2. a definition mechanism that uses the abbreviated form for OpenMath objects.
3. better error messages, and some validity warnings.
4. extend the abbreviated mode by parser tags for `<OMSTR>`, `<OMI>`, `<OMF>`,...

macro	Type	Arguments	Env
omdoc	root	label	+
omgroup	struc	label, type	+
omtext	text	label	+
omCMP	text	[lang], label	+
omrsrelation	text	type, for, from	+
omassertion	math	label, theory, type	+
omFMP	math		+
omassumption	math	label	+
omconclusion	math	label	+
omproof	math	label, for, theory	+
omexample	math	label, for	+
omhypothesis	proof	label	+
omderive	proof	label	+
omconclude	proof	label	+
ommethod	proof	theory, name	-
omparameter	proof	—	+
ompremise	proof	refersto	-
omextpremise	proof	href	-
omsymbol	theory	label	+
omstructure	theory	label	+
omfeature	theory	label	+
omcommonname	theory	[lang]	+
omdefinition	theory	label, for	+
omexercise	aux	label, for	+
omsolution	aux	label	+
omhint	aux	label	+
ommcgroup	aux	—	+
ommc	aux	label	+
omchoice	aux	—	+
omanswer	aux	verdict	+
omomlet	aux	href	+
omref	aux	tolabel	-

Figure 4.5: OMDoc elements

5. an Xsl style sheet to generate `latex2omdoc` input from OMDoc representations: a first version exists at `http://www.mathweb.org/omdoc/xsl/omdoc2latex.xsl`, but it is largely untried.

```
\newtheorem{thm}{Theorem}[section]
\newtheorem{defn}{Definition}[thm]
\begin{defn}[Monoid]
    A monoid is a semigroup with a unit element
\end{defn}
\begin{thm}[Hauptsatz]
    A monoid has at most one unit.
\end{thm}

%%%%%%% migrates to
%%%%%%% (we leave the stuff up there to see the old version in the DVI)

\begin{omdocout}[migrationex.xml]
\ommiglanguage{eng}
\begin{ommetadata}
\dctitle{Monoids migrated from LaTeX}
\end{ommetadata}

\begin{omtheory}{monoids}
\ommigtheory{monoids}
\begin{definition}[Monoid]
  \begin{omverb}
    A monoid is a semigroup with a unit element
  \end{omverb}
\end{definition}
\end{omtheory}

\begin{remark}[Hauptsatz]{monoids}
  \begin{omverb}
    A monoid has at most one unit.
  \end{omverb}
\end{remark}

\begin{omassertion}{sin}{trig}{theorem}
  \begin{omFMP}
    \mathematica{sin[1+1] == 2}
  \end{omFMP}
\end{omassertion}

\end{omdocout}
%%% Local Variables:
%%% mode: latex
%%% TeX-master: "omdoc"
%%% End:
```

Figure 4.6: The file `migrationex.tex`

Definition 4.4.0.1 (Monoid) *A monoid is a semigroup with a unit element*

Theorem 4.4.1 (Hauptsatz) *A monoid has at most one unit.*

OMDoc(1): monoids.
OMDoc(2): Monoid.sym.
OMDoc(3): 1-Monoid. 1-Monoid-cmp
OMDoc(4): 2-Hauptsatz. 2-Hauptsatz-cmp
OMDoc(5): sin. MATHEMATICAsin[1+1] == 2

Figure 4.7: The DVI output of `migrationex.tex`

```
<?xml version="1.0"?>
<!DOCTYPE omdoc SYSTEM "http://www.mathweb.org/omdoc/dtd/omdoc.dtd" []>


<!-- generated from omdoc.tex, do not edit -->


<omdoc id="top">

<metadata>
 <dc:Title xml:lang="en">
 Monoids migrated from LaTeX
 </dc:Title>
</metadata>

<theory id="monoids">
<symbol id="Monoid.sym">
<commonname xml:lang="eng">Monoid</commonname>
</symbol>

<definition id="1-{Monoid}" for="Monoid-{sym}">
<metadata>
 <dc:Title xml:lang="eng">
 Monoid
 </dc:Title>
</metadata>

<CMP xml:lang="eng" format="omtext">
    A monoid is a semigroup with a unit element

</CMP>
</definition>

</theory>

<omtext id="2-{Hauptsatz}">
<metadata>
 <dc:Title xml:lang="eng">
 Hauptsatz
 </dc:Title>
</metadata>

<rsrelation type="elaboration" for="monoids" />
<CMP xml:lang="eng" format="omtext">
    A monoid has at most one unit.

</CMP>
</omtext>

<assertion id="sin" theory="trig" type="theorem">
<FMP>
</FMP>

</assertion>

</omdoc>
```

Figure 4.8: The file `migrationex.xml` generated form `migrationex.tex`

Chapter 5

Conclusion

We have proposed an extension to the OpenMath standard that allows to represent the semantics and structure various kinds of mathematical documents, including articles, textbooks, interactive books, courses. We have motivated and described the language and presented an Xml document type definition for it.

We are currently testing this in the development of a user-adaptive interactive book including proof explanation based on IDA [CCS99] in close collaboration with the authors. This case study unites several of the application areas discussed in the introduction. The re-representation of IDA in the OMDoc format makes it possible to machine-understand the structure of the document, read it into the MBase [FK00, KF00] knowledge base system without loss of information, preserving the structure, and generate personalized sub-documents or linearizations of the structured data based on a simple user model. Furthermore, the OMDoc representation supports the formalization of (parts of) the mathematical knowledge in IDA and makes it accessible to the ΩMEGA mathematical assistant system [BCF+97], which can prove some of the problems either fully automatically (by proof planning) or in interaction with the authors. This newly developed formal data (it is not present in IDA now) will enable the reader to read and experiment with the proofs behind the mathematical theory, much as she can in the present version with the integrated computer algebra system GAP [S+95]. Finally, OMDoc will serve as the input format for the Lima system (see [Bau99]), an experimental natural language understanding system specialized to mathematical texts (this can be used to develop formalization in `FMP`s from the text in the respective `CMP`s).

In the context of this project, 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 [Koh00] 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. Michel Vollebregt has built a program that traverses an OMDoc and substitutes various representations for formulae (including the Mathematica, GAP, and Maple representations) with the corresponding OpenMath representations.

Acknowledgments

The work presented in this report was supported by the "Deutsche Forschungsgemeinschaft" in the special research action "Resource-adaptive cognitive processes" (SFB 378), Project ΩMEGA and Heisenberg Stipend.

The author would like to thank Armin Fiedler, Andreas Franke, Martin Pollet, and Julian Richardson for productive discussions, and the RIACA group (specifically Arjeh Cohen, Olga Caprotti, Michel Vollebregt, and Manfred Riem) for valuable input from the IDA case study.

Bibliography

[AHMS00] Serge Autexier, Dieter Hutter, Heiko Mantel, and Axel Schairer. Towards an evolutionary formal software-development using CASL. In C. Choppy and D. Bert, eds., *Proceedings Workshop on Algebraic Development Techniques, WADT-99*. Springer, LNCS 1827, 2000.

[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.

[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.

[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.

[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.

[CCS99] Arjeh Cohen, Hans Cuypers, and Hans Sterk. *Algebra Interactive!* Springer Verlag, 1999. Interactive Book on CD.

[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.

[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.

[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`.

[DMOT01] Steve DeRose, Eve Maler, David Orchard, and Ben Trafford. Xml linking language (XLink). W3c recommendation, W3C, 2001. Available at `http://www.w3.org/TR/xlink`.

[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.

[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.

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

[HF96] Xiaorong Huang and Armin Fiedler. Presenting machine-found proofs. In M.A. McRobbie and J.K. Slaney, eds., *Proceedings of the 13th Conference on Automated Deduction*, no.1104 in LNAI, pp.221–225, New Brunswick, NJ, USA, 1996. Springer Verlag.

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

[Hor98] Helmut Horacek. Generating inference-rich discourse through revisions of RST-trees. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence AAAI-98 and Tenth Conference on Innovative Application of Artificial Intelligence IAAI-98*, pp.814–820. MIT Press, 1998.

[Hut99] Dieter Hutter. Reasoning about theories. Tech. rep., Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), 1999.

[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.

[Koh00] Michael Kohlhase. Creating OMDOC representations from LaTeX. Internet Draft available at `http://www.mathweb.org/omdoc`, 2000.

[MT83] William Mann and Sandra Thompson. Rhethorical structure theory: A theory of text organization. Technical Report ISI/RR-83-115, University of Southern California, Information Science Institute, 1983.

[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`.

[S+95] Martin Schönert et al. *GAP – Groups, Algorithms, and Programming*. Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany, 1995.

[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.

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

Appendix A

Quick-Reference Table to the OMDoc Elements

We give a quick-reference table for the OMDoc elements

Element	Type	Attributes		D	Content
		Required	Optional	C	
omdoc	general	id	type, catalogue	+	OMDoc element
catalogue	general			–	loc*
loc	general	theory	omdoc, cd	–	EMPTY
omgroup	general			+	OMDoc element
ref	general		xref, theory, name kind	–	ANY
omtext	general	id	rsrelation	+	CMP+,FMP?
CMP	general		type, xml:lang	–	ANY
FMP	math			–	assumption*,conclusion)|OMOBJ
assertion	math	id	type, theory	+	private*,symbol*,CMP*,FMP?
assumption	math	id		+	CMP*,OMOBJ?
concluson	math	id		+	CMP*,OMOBJ?
example	math	id	type, assertion, proof	+	CMP|OMOBJ
alternative-def	math	id, for, theory, entailed-by, entails, entailed-by-thm, entails-thm	type	+	CMP*,(FMP|requation*|OMOBJ)
proof	proof	id, for, theory		+	symbol*,CMP*, (hypothesis|derive|metacomment)*, conclude
proofobject	proof	id, for, theory		+	CMP*,OMOBJ
metacomment	proof	id		–	CMP*
hypothesis	proof	id		–	symbol*,CMP*,FMP?
derive	proof	id		–	CMP*,FMP?,method?,premise*, (proof|proofobject)?

conclude	proof	id		–	CMP*,method?,premise*, (proof|proofobject)?
method	proof			–	(ref|OMSTR),parameter*
parameter	proof			–	OMOBJ
premise	proof	xref		–	EMPTY
theory	theory	id		+	commonname*,CMP*,most below
symbol	theory	id	type, scope	+	CMP*,(commonname|type|selector)*
commonname	theory		xml:lang	–	ANY
signature	theory	id, for, system		–	
type	theory	system		–	OMOBJ
axiom	theory	id		+	private*,symbol*,CMP*,FMP?
definition	theory	id, for	type, just-by	+	CMP*,(FMP+|requation+|OMOBJ)?
requation	theory			–	pattern, value
pattern	theory			–	OMA|OMS
value	theory			–	OpenMath element
imports	theory	id, from	type, hiding	–	CMP*,morphism?
morphism	theory	id	base	–	requation*
inclusion	theory	for		–	
adt	theory	id	type	+	CMP*,commonname*,sortdef+
sortdef	theory	id	kind, scope	–	commonname*,(symbol|insort)*
constructor	theory	id	type, scope	+	commonname*,argument*
argument	theory	sort		+	selector?
insort	theory	for		–	
selector	theory	id	type, scope, kind	–	commonname*
theory-inclusion	theory	id, from, to, by	timestamp	+	(morphism,decomposition?)
axiom-inclusion	theory	id, from, to	timestamp	+	morphism?, (path-just|assertion-just))
assertion-just	theory	ids	timestamp	–	EMPTY
decomposition	theory	links	timestamp	–	EMPTY
path-just	theory	local, globals	timestamp	–	EMPTY
exercise	aux	id, for	rsinfo	+	private*,symbol*,CMP*,FMP?,hint?, (solution*|mc*)
hint	aux	id		+	private*,symbol*,CMP*,FMP?
solution	aux	id	for	+	(private*,symbol*,CMP*,FMP?)|proof
mc	aux	id		–	symbol*,choice,hint?,answer
choice	aux	id		+	private*,symbol*,CMP*,FMP?
answer	aux	id, verdict		+	private*,symbol*,CMP*,FMP?
omlet	aux	id	argstr, function	+	ANY

private	aux	id	for, , theory, type, pto, pto-version, format, requires, classid, codebase, width, height	+	CMP*,data
code	aux	id, theory	for, type, pto, pto-version, format, requires, classid, codebase, width, height	+	CMP*,input?,output?,effect?,data
input	aux			–	CMP*
output	aux			–	CMP*
effect	aux			–	CMP*
data	aux		href	–	<![CDATA[...]]>
ignore	aux		type, comment		ANY
presentation	aux	id, for	fixity, parent, lbrack, rbrack, separator, bracket-style	–	use*
use	aux	format	lbrack, rbrack, separator, crossref-symbol	–	ANY
metadata	meta			–	(element)*
Title	meta		xml:lang	–	ANY
Contributor	meta		role	–	ANY
Creator	meta		role	–	ANY
Subject	meta			–	ANY
Description	meta			–	ANY
Publisher	meta			–	ANY
Date	meta		action	–	ANY
Type	meta			–	ANY
Format	meta			–	ANY
Identifier	meta		scheme	–	ANY
Source	meta			–	ANY
Lanugae	meta			–	ANY
Relation	meta			–	ANY
Coverage	meta			–	ANY
Rights	meta			–	ANY
extradata	meta			–	ANY

and the OMDoc attributes.

Attribute	Description

`id`	associates a unique identifier to an element, which can thus be referenced by an `for` attribute.
`for`	can be used to reference an element by its unique identifier given in its `id` attribute.
`logic`	specifies the logical system used to encode the property in an `FMP`.
`system`	specifies the type system used in the `signature` element
`from, to`	only appear in the `linkage`[1] element, they specify the elements between whom the linkage text mediates.
`type`	the type of an element. The value of this attribute is largely unspecified, its interpretation depends on the application.
`xml:lang`	the language the text in the element is expressed in. This must be a 1766-compliant specification of the primary language of the content.
`xref`	a uniform resource locator (`URL`).
`verdict`	only appears in the `answer` element, it specifies the truth or falsity of the answer. This can be used e.g. by a grading application.

[1] This is not right anymore

Appendix B

Dublin Core Metadata

In the following we will describe individual Dublin Core metadata elements.

`Title` The title of the publication. Although an OMDoc document must include at least one instance of this element type, multiple instances are permitted. Any reading system that displays title metadata to the user should either use the first `Title` only, or all `Title` elements.

`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. Publication 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 by the reading system. This specification recommends that the content of the `Creator` elements hold the text for a single name as it would be presented to the user. The `Creator` elements has the optional `role` attribute, which can take the values defined in appendix B.1.

`Subject` Multiple instances of the `Subject` element are supported, each including an arbitrary phrase or keyword.

`Description` Plain text describing the publication's content.

`Publisher` The publisher as defined in RFC2413.

`Contributor` A party whose contribution to the publication is secondary to those named in `Creator` elements. Other than significance of contribution, the semantic of this element are identical to those of `Creator`.

`Date` Date of publication, in the format defined by "Date and Time Formats" at `http://www.w3.org/TR/NOTE-datetime` and by ISO 8601 on which it is based. In particular, dates without times are represented in the form YYYY[-MM[-DD]]: a mandatory 4-digit year, an optional 2-digit month, and if the month is given, an optional 2-digit day of month.

`Type` Dublin Core provides examples of resource types "such as home page, novel, poem, working paper, technical report, essay, dictionary." The Dublin Core is currently considering revisions to the usage of this field, and should be consulted for updated definitions.

`Format` An enumerated list of formats for this content is being developed by the Dublin Core.

`Identifier` A string or number used to uniquely identify the resource. At least one `Identifier` must have an `id` attribute specified.

`Source` Information regarding a prior resource from which the publication was derived (see RFC 2413).

`Language` If specified, the content must be an RFC 1766-compliant specification of the primary language of the content. (See `http://www.ietf.org/rfc/rfc1766.txt`) US English (en-us) is the default.

`Coverage` The place or time which the publication's content addresses.

`Rights` A statement about rights, or a link to one. In this specification, the copyright notice and any further rights description should appear directly.

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 specification and use optional `role` attributes for this purpose. (see section B.1)

B.1 Roles in Dublin Core Metadata

The normative list of values used for the `role` attribute is defined by the USMARC relator code list (`http://www.loc.gov/marc/relators/re9802r1.html`). The overview given here is only for convenience.

Currently, OMDoc only supports the following roles:

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

Author in quotations or text extracts `aqt` Use for a person whose work is largely quoted or extracted in a works to which he or she did not contribute directly. Such quotations are found particularly in exhibition catalogs, collections of photographs, etc.

Author of afterword, colophon, etc. `aft` Use for a person or corporate body responsible for an afterword, postface, colophon, etc. but who is not the chief author of a work.

Author of introduction, etc. `aui` Use for a person or corporate body responsible for an introduction, preface, foreword, afterword, or other critical matter, but who is not the chief author. The author of a specific part of the book may be expressed using a metadata element in that part.

Bibliographic antecedent `ant` Use for the author responsible for a work upon which the work represented by the catalog record is based. This may be appropriate for adaptations, sequels, continuations, indexes, etc.

Collaborator `clb` 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.

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

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

Transcriber `trc` Use for a person who prepares a handwritten or typewritten copy from original material, including from dictated or orally recorded material.

Translator `trl` 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.

Appendix C

The OMDoc Document Type Definition

In this section, 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`. It 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 OMDoc format (OpenMath documents)
   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/omdoc.dtd (current experimental)
   URL: http://www.mathweb.org/omdoc/dtd/omdoc*.dtd (old)
   This DTD is still experimental, it is intended as a basis for discussion.
   Comments are welcome! (send mail to kohlhase@mathweb.org)
   See the documentation and examples at http://www.mathweb.org/omdoc
   (c) 1999, 2000 Michael Kohlhase, released under the GNU Public License
-->

<!-- first we define a couple of useful abbreviations  -->

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


<!ENTITY % idmatter "id CDATA #REQUIRED %midmatter;">
<!ENTITY % idimatter "id CDATA #IMPLIED %midmatter;">

<!ENTITY % idrefmatter "%idmatter; for CDATA #REQUIRED">
<!ENTITY % idirefmatter "%idimatter; for CDATA #REQUIRED">
<!-- attribute for is an URIref -->

<!ENTITY % timestamp "timestamp NMTOKEN #REQUIRED">

<!--    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 "(en|de|fr|la|it|nl|ru|pl|es|tr|zh|ja|ko)">

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

<!ENTITY % frommatter    "from NMTOKEN #REQUIRED">
<!ENTITY % fromtomatter  "%frommatter; to CDATA #REQUIRED">
<!-- attribute to is an URIref -->

<!ENTITY % fromtobymatter "%fromtomatter; by CDATA #REQUIRED">
```

```
<!-- attribute by is an URIref -->

<!ENTITY % linkmatter "%midmatter; id CDATA #IMPLIED xref CDATA #IMPLIED">
<!-- attribute xref is an URIref -->

<!ENTITY % rstype "abstract|introduction|conclusion|thesis|antithesis|
                          elaboration|motivation|evidence|linkage|narrative|
                          sequence|alternative|general">

<!ENTITY % rsmatter "type NMTOKEN #IMPLIED
                     for CDATA #IMPLIED
                     from CDATA #IMPLIED">
<!-- attribute for is a URIref, from a URIrefs -->
<!-- best use one of the %rstype; for type -->


<!--  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 linkmatter
      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 % omns "xmlns CDATA #FIXED 'http://www.openmath.org/OpenMath'">
<!-- things which can be variables -->

<!ENTITY % omvar "OMV | OMATTR" >

<!-- symbol, original OM, links make no sense -->
<!ELEMENT OMS EMPTY>
<!ATTLIST OMS name CDATA #IMPLIED
              cd CDATA #IMPLIED
              %omns;>

<!-- variable original OM, links make no sense -->
<!ELEMENT OMV EMPTY>
<!ATTLIST OMV name CDATA #IMPLIED  %omns;>

<!-- integer; links make sense, since integers can be big -->
<!ELEMENT OMI (#PCDATA)>
<!ATTLIST OMI %linkmatter;  %omns;>

<!-- byte array; links make sense, since byte arrays can be big -->
<!ELEMENT OMB (#PCDATA) >
<!ATTLIST OMB %linkmatter;  %omns;>

<!-- string; links make sense, since strings can be big -->
<!ELEMENT OMSTR (#PCDATA) >
<!ATTLIST OMSTR %linkmatter;  %omns;>

<!-- floating point; links make sense, since Integers can be big -->
<!ELEMENT OMF EMPTY>
<!ATTLIST OMF dec CDATA #IMPLIED
              hex CDATA #IMPLIED
              %linkmatter; %omns;>

<!-- apply constructor; links make sense, no copied substructure -->
<!ELEMENT OMA (%omel;)*>
<!ATTLIST OMA %linkmatter; %omns;>

<!-- binding constructor & variable; links make sense, no copied substructure -->
<!ELEMENT OMBIND ((%omel;), OMBVAR, (%omel;))? >
<!ATTLIST OMBIND %linkmatter; %omns;>

<!-- bound variables, original OM, links make no sense -->
<!ELEMENT OMBVAR (%omvar;)+ >
<!ATTLIST OMBVAR  %omns;>

<!-- error; original OM, links make no sense -->
```

```
<!ELEMENT OME (OMS, (%omel;)* ) >
<!ATTLIST OME %omns;>

<!-- attribution constructor & attribute pair constructor -->
<!ELEMENT OMATTR (OMATP, (%omel;))? >
<!ATTLIST OMATTR %linkmatter; %omns;>

<!ELEMENT OMATP (OMS, (%omel;))+ >
<!ATTLIST OMATP %omns;>

<!-- OM object constructor; links make sense to avoid copying substructure -->
<!ELEMENT OMOBJ (%omel;)? >
<!ATTLIST OMOBJ %omns; %linkmatter;>


<!-- 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 % dcdata "Contributor | Creator | Translator
                 | Subject | Title
                 | Description | Publisher | Date | Type
                 | Format | Identifier | Source | Language
                 | Relation | Coverage | Rights">

<!ENTITY % dcns "xmlns CDATA #FIXED 'http://purl.org/DC'">
<!ENTITY % rolematter "role (aut|aqt|aft|aui|ant|clb|edt|ths|trc|trl) 'aut'">

<!ELEMENT metadata ((%dcdata;)*,extradata?)>

<!-- first the Dublin Core Metadata model of the
     Dublin Metadata initiative (http://purl.org/dc) -->

<!ELEMENT Contributor ANY><!ATTLIST Contributor %dcns; %rolematter;>
<!ELEMENT Title ANY><!ATTLIST Title %langmatter; %dcns;>
<!ELEMENT Creator ANY><!ATTLIST Creator %dcns; %rolematter;>
<!ELEMENT Translator ANY><!ATTLIST Translator %dcns; %langmatter;>
<!ELEMENT Subject ANY><!ATTLIST Subject %dcns;>
<!ELEMENT Description ANY><!ATTLIST Description %dcns;>
<!ELEMENT Publisher ANY><!ATTLIST Publisher %dcns;>
<!ELEMENT Date ANY><!ATTLIST Date action NMTOKEN #IMPLIED %dcns;>
<!ELEMENT Type ANY><!ATTLIST Type %dcns;>
<!ELEMENT Format ANY><!ATTLIST Format %dcns;>
<!ELEMENT Identifier ANY><!ATTLIST Identifier %dcns; scheme NMTOKEN "ISBN">
<!ELEMENT Source ANY><!ATTLIST Source %dcns;>
<!ELEMENT Language ANY><!ATTLIST Language %dcns;>
<!ELEMENT Relation ANY><!ATTLIST Relation %dcns;>
<!ELEMENT Coverage ANY><!ATTLIST Coverage %dcns;>
<!ELEMENT Rights ANY><!ATTLIST Rights %dcns;>

<!-- 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 ANY>

<!-- ========= OMDoc Math elements ============ -->

<!ENTITY % mathitem "assertion|alternative-def|example|
                     theory-inclusion|axiom-inclusion|
                     proof|proofobject">

<!ENTITY % cf "symbol*,CMP*,FMP?">
<!ENTITY % cfm   "(metadata?,private*,%cf;)">

<!ELEMENT FMP ((assumption*,conclusion)|OMOBJ)>
<!ATTLIST FMP logic NMTOKEN #IMPLIED
              %midmatter;>

<!ELEMENT assertion %cfm;>
<!ATTLIST assertion theory NMTOKEN #IMPLIED
                    type (theorem|lemma|corollary|conjecture) "conjecture"
                    %idmatter;>

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

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

```
<!ELEMENT alternative-def (metadata?,CMP*,(FMP|requation*|OMOBJ))>
<!ATTLIST alternative-def theory NMTOKEN #REQUIRED
                          type  (simple|inductive|implicit|obj) "simple"
                          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 URIres
     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 -->

<!-- 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 NMTOKEN #REQUIRED
                %idrefmatter;>

<!ELEMENT proofobject (metadata?,CMP*,OMOBJ)>
<!ATTLIST proofobject theory NMTOKEN #REQUIRED
                %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 (%cf;)>
<!ATTLIST hypothesis %idmatter;>

<!ELEMENT method ((ref|OMSTR),parameter*)>

<!ELEMENT parameter (OMOBJ)>

<!ELEMENT premise EMPTY>
<!ATTLIST premise href CDATA #REQUIRED>

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

<!ELEMENT axiom-inclusion (metadata?,morphism?,(path-just|assertion-just))>
<!ATTLIST axiom-inclusion %timestamp;
                          %fromtomatter;
                          %idmatter;>

<!ELEMENT theory-inclusion (metadata?,morphism,decomposition?)>
<!ATTLIST theory-inclusion %timestamp;
                           %idmatter;
                           %fromtobymatter;>

<!ELEMENT path-just EMPTY>
<!ATTLIST path-just %timestamp;
                    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 assertion-just EMPTY>
<!ATTLIST assertion-just %timestamp;
                         ids CDATA #REQUIRED
                         %midmatter;>
<!-- attribute ids is an URIrefs, points to  a list of assertions -->

<!ELEMENT decomposition EMPTY>
```

```
<!ATTLIST decomposition %timestamp;
                        links CDATA #REQUIRED
                        %midmatter;>
<!-- attribute links is an URIrefs, points to a list of axiom-inlcusions -->

<!-- OMDoc theory elements -->

<!ENTITY % onlyintheoryitem "symbol|axiom|definition|adt|imports|inclusion">
<!ENTITY % alsointheoryitem "alternative-def|proof|signature|assertion|exercise|presentation|example|omtext|omgroup|private|code|ignore">
<!ENTITY % intheoryitem "%onlyintheoryitem;|%alsointheoryitem;">

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

<!ELEMENT theory (metadata?,commonname*,CMP*,(%intheoryitem;)*)>
<!ATTLIST theory %idmatter;>

<!ELEMENT symbol (metadata?, CMP*,(commonname|type|selector)*)>
<!ATTLIST symbol %insymbolmatter;>

<!ELEMENT commonname ANY>
<!ATTLIST commonname %langmatter;
                     %midmatter;>

<!ELEMENT signature EMPTY>
<!ATTLIST signature %idrefmatter;
                    system NMTOKEN #REQUIRED>

<!ELEMENT type (OMOBJ)>
<!ATTLIST type system NMTOKEN #REQUIRED
               %midmatter;>

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

<!-- 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+|OMOBJ)?)>
<!ATTLIST definition just-by CDATA #IMPLIED
                     type  (simple|inductive|implicit|obj) "simple"
                     %idrefmatter;>
 <!-- attribute just-by is an URIref points to an assertion -->

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

<!ELEMENT pattern (OMA|OMS)>

<!ELEMENT value (%omel;)>

<!-- 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?,CMP*,commonname*,sortdef+)>
<!ATTLIST adt type (loose|generated|free) "loose"
              %idmatter;>

<!ELEMENT sortdef (commonname*,(constructor|insort)*)>
<!ATTLIST sortdef %idimatter;
                  kind NMTOKEN #FIXED "sort"
                  scope (global|local) "global">

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

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

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

<!ELEMENT selector (commonname)*>
<!ATTLIST selector %insymbolmatter;
                   type (total|partial) "partial">
```

```
<!-- Now comes the support for theory inheritance, i.e. for building complex -->
<!-- theories. -->

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

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

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

<!-- Auxiliary elements -->

<!ENTITY % auxitem "exercise|solution|omlet|private|code|presentation">

<!ELEMENT exercise (%cfm;,hint?,(solution*|mc*))>
<!ATTLIST exercise %idmatter;
                   %rsmatter;>

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

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

<!-- support for multiple choice interactions -->

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

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

<!ELEMENT answer %cfm;>
<!ATTLIST answer verdict CDATA #REQUIRED
                 %idimatter;>

<!ELEMENT omlet ANY>
<!ATTLIST omlet %idimatter;
                argstr CDATA #IMPLIED
                function CDATA #IMPLIED>
<!-- attribute function is an URIref, points to a code element
     attribute argstr is a string of arguments supplied to the function -->

<!ENTITY % privmatter "%idimatter;
                   theory CDATA #IMPLIED
                   pto NMTOKENS #IMPLIED
                   pto-version NMTOKENS #IMPLIED
                   format NMTOKEN #IMPLIED
                   requires CDATA #IMPLIED
                   type NMTOKEN #IMPLIED
                   classid CDATA #IMPLIED
                   codebase CDATA #IMPLIED
                   width CDATA #IMPLIED
                   height CDATA #IMPLIED">

<!ELEMENT private (metadata?,CMP*,data)>
<!ATTLIST private %privmatter;>

<!ELEMENT code (metadata?,CMP*,input?,output?,effect?,data)>
<!ATTLIST code %privmatter;>

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

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

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

<!ELEMENT data ANY>
<!ATTLIST data %midmatter;
               href 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>


<!ENTITY % crossrefmatter "crossref-symbol (no|yes|brackets|separator|lbrack|rbrack|all) 'yes'">

<!ELEMENT presentation (use*)>
<!ATTLIST presentation fixity NMTOKEN "prefix"
                       parent (OMA|OMBIND|OMATTR) #IMPLIED
                       lbrack CDATA "("
                       rbrack CDATA ")"
                       separator CDATA ","
                       bracket-style (lisp|math) "math"
                       mode CDATA #IMPLIED
                       precedence NMTOKEN #IMPLIED
                       %crossrefmatter;
                       %idirefmatter;>
<!-- currently recognized fixities are prefix|infix|postfix|assoc -->

<!ELEMENT use ANY>
<!ATTLIST use format NMTOKEN #REQUIRED
              %langmatter;
              requires CDATA #IMPLIED
              larg-group CDATA #IMPLIED
              rarg-group CDATA #IMPLIED
              lbrack CDATA #IMPLIED
              rbrack CDATA #IMPLIED
              separator CDATA #IMPLIED
              precedence NMTOKEN #IMPLIED
              crossref-symbol (no|yes|brackets|separator|lbrack|rbrack|all) "yes">

<!-- the attributes in the <use> element overwrite those in the
     <presentation> element, therefore, they do not have defaults -->

<!ENTITY % omdocitem "omtext|%mathitem;|%auxitem;|theory|omgroup|ref|ignore">
<!ELEMENT omdoc (metadata,catalogue?,(%omdocitem;)*)>
<!ATTLIST omdoc %idmatter;
                type NMTOKEN "document"
                catalogue CDATA #IMPLIED
                version CDATA #IMPLIED>

<!ELEMENT catalogue (loc)*>

<!ELEMENT loc EMPTY>
<!ATTLIST loc theory NMTOKEN #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 -->


<!-- Now, we come to the text elements, they are very simple, since they can include
     arbitrary text and  markup -->


<!ELEMENT omtext (metadata?,CMP+,FMP?)>
<!ATTLIST omtext %idmatter;
                 %rsmatter;>
<!-- for/from point to some %omdocitem; -->

<!ELEMENT CMP ANY>
<!ATTLIST CMP format NMTOKEN "omtext"
              %langmatter;>

<!-- grouping defines the structure of a document-->

<!ELEMENT omgroup (metadata?,(%omdocitem;)*)>
<!ATTLIST omgroup type NMTOKEN "sequence"
                  %idmatter;>
<!-- best use one of the %rstype; there -->
```

```
<!-- co-referencing  allows to use elements with an
     'id' attribute multiple times -->
<!ELEMENT ref ANY>
<!ATTLIST ref xref CDATA #IMPLIED
              theory NMTOKEN #IMPLIED
              name NMTOKEN #IMPLIED
              mid NMTOKEN #IMPLIED
              kind NMTOKEN #IMPLIED>

<!-- =============== omdoc.dtd ends here  ==================== -->
```

Appendix D

A latex2omdoc Example

In this section, we will look at an example of an OMDoc file generated by `latex2omdoc.sty`

D.1 The LaTeX source file

```
\begin{omdocout}

\begin{ommetadata}
  \dccontributor{Contributor 1}
  \dccreator[edt]{Creator as the editor}
  \dctitle{The title of the document}
 here we define the metadata of the document
\end{ommetadata}

\begin{omomtext}{intro}\omrsrelation{introduction}{}{}
  \begin{omCMPverb}{omtext}
      We will start out with an introductory text
  \end{omCMPverb}
\end{omomtext}

\begin{omtheory}{testtheory}
\omcommonname[de]{Monoid Theorie}

\begin{omsymbol}{monoid}
  \begin{omCMPverb}{omtext}
    The monoids that we all know and love
  \end{omCMPverb}
  \begin{omCMPverb}[de]{omtext}
    Die Monoide, wie wir sie alle kennen und lieben
  \end{omCMPverb}
  \omcommonname[de]{plus}\omcommonname{plus}
  \begin{omtypeverb}{POST}
    <OMOBJ><OMA><OMS cd="mltt" name="funtype"/><OMV name="alpha"/></OMA></OMOBJ>
  \end{omtypeverb}
  here we define monoids
\end{omsymbol}


\begin{omomtext}{testremark}
  \omrsrelation{introduction}{}{}
  \begin{omCMP}{omtext}
    \begin{omverb}then we will use an omnote to\end{omverb}
    \begin{omomletverb}{appl1}{js}{sdlfkj}{call-mint}
       explain
    \end{omomletverb}
    \begin{omverb}the symbol\end{omverb}
  \end{omCMP}
\end{omomtext}

\begin{omdefinition}{monoiddef}{monoid}
  \begin{omCMPverb}{omtext}
    Here comes the definition of  monoids
  \end{omCMPverb}
  \begin{omFMP}\((#test:sym:$X:$Y:)\)\end{omFMP}
  sdfsdfsdf
\end{omdefinition}
```

```
\begin{omomtext}{testlinkage}
  \omrsrelation{linkage}{monoiddef}{recursivemonoid}
  \begin{omCMPverb}{omtext}
    then a omlinkage to link the previous definition to the next
  \end{omCMPverb}
\end{omomtext}

\begin{omdefinition}{recursivemonoid}{monoid}
  \begin{omCMPverb}{omtext}
    Here comes a recursive definition
  \end{omCMPverb}
  \begin{omrequation}
    \<(#arith1:plus:$X:#arith1:zero:)\>\goesto\($X:\)
  \end{omrequation}
  sdfsdfsdf
\end{omdefinition}

\end{omtheory}

\begin{omassertion}{testtheorem}{testtheory}{theorem}
  \begin{omCMPverb}{omtext}A theorem\end{omCMPverb}
  \begin{omFMP}
    \({#quant1:forall:$X:$Y:.(#relation:eq:$X:$Y:)}\)
  \end{omFMP}
\end{omassertion}

\begin{omassertion}{testlemma}{testtheory}{lemma}
  \begin{omCMPverb}{omtext}
    Transitivity of equality
  \end{omCMPverb}
  \begin{omassumption}{A1}
    \begin{omCMPverb}{omtext}\($X:\) and  \($Y:\) are equal\end{omCMPverb}
    \begin{omFMP}
    \((#relation:eq:$X:$Y:)\)
    \end{omFMP}
  \end{omassumption}
  \begin{omassumption}{A2}
    \begin{omCMPverb}{omtext}\($Y:\) and  \($Z:\) are equal\end{omCMPverb}
    \begin{omFMP}\((#relation:eq:$Y:$Z:)\)\end{omFMP}
  \end{omassumption}
  \begin{omconclusion}{C1}
    \begin{omCMPverb}{omtext}All objects are equal\end{omCMPverb}
    \begin{omFMP}
      \({#quant1:forall:$X:$Y:.(#relation:eq:$X:$Y:)}\)
    \end{omFMP}
  \end{omconclusion}
\end{omassertion}


\begin{omproof}{P1}{testlemma}{testtheory}
  \begin{omCMPverb}{omtext}
    A proof with two proof steps  for the lemma above
  \end{omCMPverb}
  \begin{omderive}{D1}
    \begin{omCMPverb}{omtext}The first proof step consists\end{omCMPverb}
    \begin{omassumption}{D1.A}
      \begin{omCMPverb}
        This proof step makes a local hypothesis
      \end{omCMPverb}
    \end{omassumption}
    \begin{omconclusion}{D1.C}
    \begin{omCMPverb}{omtext}
      The assertion of the first proof step
    \end{omCMPverb}
    \begin{omFMP}\(#test:formula:\)\end{omFMP}
  \end{omconclusion}
\end{omderive}
  \begin{omderive}{D2}
    \begin{omCMPverb}{omtext}The second step\end{omCMPverb}
    \begin{omconclusion}{D2.c}
    \begin{omFMP}\(#test:formula:\)\end{omFMP}
  \end{omconclusion}
\end{omderive}
  \begin{omconclude}{Conc}
    \begin{omCMPverb}{omtext}The last proof step\end{omCMPverb}
\end{omconclude}
\end{omproof}
\end{omdocout}
```

```
%%% Local Variables:
%%% mode: latex
%%% TeX-master: "omdoc"
%%% End:
```

D.2 The log information in the DVI file

here we define the metadata of the document
OMDoc(6): intro.
OMDoc(7): testtheory.
OMDoc(8): monoid. here we define monoids
OMDoc(9): testremark.
OMDoc(10): monoiddef. sdfsdfsdf
OMDoc(11): testlinkage.
OMDoc(12): recursivemonoid. sdfsdfsdf
OMDoc(13): testtheorem.
OMDoc(14): testlemma.
OMDoc(15): P1.

D.3 The XML document generated by this file

```
<?xml version="1.0"?>
<!DOCTYPE omdoc SYSTEM "http://www.mathweb.org/omdoc/dtd/omdoc.dtd" []>

<!-- generated from omdoc.tex, do not edit -->


<omdoc id="top">

<metadata>
 <dc:Contributor role="aut">Contributor 1</dc:Contributor>
 <dc:Creator role="edt">Creator as the editor</dc:Creator>
 <dc:Title xml:lang="en">The title of the document</dc:Title>
</metadata>

<omtext id="intro">
<rsrelation type="introduction" />
<CMP xml:lang="en" format="omtext">
      We will start out with an introductory text
</CMP>
</omtext>

<theory id="testtheory">
<commonname xml:lang="deu">Monoid Theorie</commonname>
<symbol id="monoid">
<CMP xml:lang="en" format="omtext">
    The monoids that we all know and love
</CMP>
<CMP xml:lang="deu" format="omtext">
    Die Monoide, wie wir sie alle kennen und lieben
</CMP>
<commonname xml:lang="deu">plus</commonname>
<commonname xml:lang="en">plus</commonname>
<type system="POST">
    <OMOBJ><OMA><OMS cd="mltt" name="funtype"/><OMV name="alpha"/></OMA></OMOBJ>
</type>
</symbol>

<omtext id="testremark">
<rsrelation type="introduction"/>
<CMP xml:lang="en" format="omtext">
then we will use an omnote to
<omlet id="appl1" type="js" argstr="sdlfkj" function="call-mint">explain</omlet>
the symbol</CMP>
</omtext>

<definition id="monoiddef" item="monoid">
<CMP xml:lang="en" format="omtext">
    Here comes the definition of  monoids
```

```
</CMP>
<FMP><OMOBJ><OMA><OMS cd="test" name="sym"/><OMV name="X"/><OMV name="Y"/></OMA></OMOBJ></FMP>
</definition>

<omtext id="testlinkage">
<rsrelation type="linkage" for="monoiddef" from="recursivemonoid"/>
<CMP xml:lang="en" format="omtext">
    then a omlinkage to link the previous definition to the next
</CMP>
</omtext>

<definition id="recursivemonoid" item="monoid">
<CMP xml:lang="en" format="omtext">
    Here comes a recursive definition
</CMP>
<requation>
 <pattern>
  <OMA><OMS cd="arith1" name="plus"/><OMV name="X"/><OMS cd="arith1" name="zero"/></OMA>
 </pattern>
 <value><OMOBJ><OMV name="X"/></OMOBJ></value>
</requation>
</definition>
</theory>

<assertion id="testtheorem" theory="testtheory" type="theorem">
<CMP xml:lang="en" format="omtext">A theorem</CMP>
<FMP><OMOBJ>
<OMBIND><OMS cd="quant1" name="forall"/>
 <OMBVAR><OMV name="X"/><OMV name="Y"/></OMBVAR>
 <OMA><OMS cd="relation" name="eq"/><OMV name="X"/><OMV name="Y"/></OMA>
</OMBIND>
</OMOBJ></FMP>
</assertion>

<assertion id="testlemma" theory="testtheory" type="lemma">
<CMP xml:lang="en" format="omtext">Transitivity of equality</CMP>
<assumption id="A1">
<CMP xml:lang="en" format="omtext">\($X:\) and  \($Y:\) are equal</CMP>
<FMP><OMOBJ>
<OMA><OMS cd="relation" name="eq"/><OMV name="X"/><OMV name="Y"/></OMA>
</OMOBJ></FMP>
</assumption>
<assumption id="A2">
<CMP xml:lang="en" format="omtext">\($Y:\) and  \($Z:\) are equal</CMP>
<FMP><OMOBJ>
<OMA><OMS cd="relation" name="eq"/><OMV name="Y"/><OMV name="Z"/></OMA>
</OMOBJ></FMP>
</assumption>
<conclusion id="C1">
<CMP xml:lang="en" format="omtext">All objects are equal</CMP>
<FMP><OMOBJ>
 <OMBIND><OMS cd="quant1" name="forall"/>
  <OMBVAR><OMV name="X"/><OMV name="Y"/></OMBVAR>
  <OMA><OMS cd="relation" name="eq"/><OMV name="X"/><OMV name="Y"/></OMA>
 </OMBIND>
</OMOBJ></FMP>
</conclusion>
</assertion>

<proof id="P1" item="testlemma" theory="testtheory">
 <CMP xml:lang="en" format="omtext">
    A proof with two proof steps  for the lemma above
 </CMP>
 <derive id="D1">
  <CMP xml:lang="en" format="omtext">The first proof step consis</CMP>
  <assumption id="D1.A">
   <CMP xml:lang="en" format="T">his proof step makes a local hypothesis</CMP>
  </assumption>
 <conclusion id="D1.C">
  <CMP xml:lang="en" format="omtext">
      The assertion of the first proof step
  </CMP>
  <FMP><OMOBJ><OMS cd="test" name="formula"/></OMOBJ></FMP>
</conclusion>
</derive>

<derive id="D2">
 <CMP xml:lang="en" format="omtext">The second step</CMP>
 <conclusion id="D2.c">
  <FMP><OMOBJ><OMS cd="test" name="formula"/></OMOBJ></FMP>
```

```
 </conclusion>
</derive>

<conclude id="Conc">
 <CMP xml:lang="en" format="omtext">The last proof step</CMP>
</conclude>
</proof>
</omdoc>
```