Michael Kohlhase
Computer Science
FAU Erlangen-Nürnbeg
michael.kohlhase@fau.de

An Open Markup Format for Mathematical Documents

OMDoc [Version 1.3]

November 7, 2019

This Document is the OMDoc 1.3 Specification.

Version Information	
Revision	(None)
Last Change	(None)
Author	(None)

Springer

To Andrea
— my wife, collaborator, and best friend —
for all her support

Abstract

The OMDoc (Open Mathematical Documents) format is a content markup scheme for (collections of) mathematical documents including articles, textbooks, interactive books, and courses. OMDoc also serves as the content language for agent communication of mathematical services on a mathematical software bus.

This document describes version 1.3 of the OMDoc format, the final and mature release of OMDoc1. The format features a modularized language design, OpenMath and MathML for representing mathematical objects, and has been employed and validated in various applications.

This book contains the rigorous specification of the OMDoc document format, an OMDoc primer with paradigmatic examples for many kinds of mathematical documents. Furthermore we discuss applications, projects and tool support for OMDoc.

Foreword

Computers are changing the way we think. Of course, nearly all desk-workers have access to computers and use them to email their colleagues, search the web for information and prepare documents. But I'm not referring to that. I mean that people have begun to think about what they do in computational terms and to exploit the power of computers to do things that would previously have been unimaginable.

This observation is especially true of mathematicians. Arithmetic computation is one of the roots of mathematics. Since Euclid's algorithm for finding greatest common divisors, many seminal mathematical contributions have consisted of new procedures. But powerful computer graphics have now enabled mathematicians to envisage the behaviour of these procedures and, thereby, gain new insights, make new conjectures and explore new avenues of research. Think of the explosive interest in fractals, for instance. This has been driven primarily by our new-found ability rapidly to visualise fractal shapes, such as the Mandelbrot set. Taking advantage of these new opportunities has required the learning of new skills, such as using computer algebra and graphics packages.

The argument is even stronger. It is not just that computational skills are a useful adjunct to a mathematician's arsenal, but that they are becoming essential. Mathematical knowledge is growing exponentially: following its own version of Moore's Law. Without computer-based information retrieval techniques it will be impossible to locate relevant theories and theorems, leading to a fragmentation and slowing down of the field as each research area rediscovers knowledge that is already well-known in other areas. Moreover, without the use of computers, there are potentially interesting theorems that will remain unproved. It is an immediate corollary of Gödel's Incompleteness Theorem that, however huge a proof you think of, there is a short theorem whose smallest proof *is* that huge. Without a computer to automate the discovery of the bulk of these huge proofs, then we have no hope of proving these simple-stated theorems. We have already seen early examples of this phenomenon in the Four-Colour Theorem and Kepler's Conjecture on sphere packing. Perhaps computers can also help us to navigate, abstract and, hence, understand these huge proofs.

Realising this dream of: computer access to a world repository of mathematical knowledge; visualising and understanding this knowledge; reusing and combining it to discover new knowledge, presents a major challenge to mathematicians and informaticians. The first part of this challenge arises because mathematical knowledge will be distributed across multiple sources and represented in diverse ways. We need a lingua franca that will enable this babel of mathematical languages to communicate with each other. This is why this book — proposing just such a lingua franca — is so important. It lays the foundations for realising the rest of the dream.

OMDoc is an open markup language for mathematical documents. The 'markup' aspect of OMDoc means that we can take existing knowledge and annotate it with the information required to retrieve and combine it automatically. The 'open' aspect of OMDoc means that it is extensible, so future-proofed against new developments in mathematics, which is essential in such a rapidly growing and complex field of knowledge. These are both essential features. Mathematical knowledge is growing too fast and is too distributed for any centrally controlled solution to its management. Control must be distributed to the mathematical communities that produce it. We must provide lightweight mechanisms under local control that will enable those communities to put the produce of their labours into the commonwealth with minimal effort. Standards are required to enable interaction between these diverse knowledge sources, but they must be flexible and simple to use. These requirements have informed OMDoc's development. This book will explain to the international mathematics community what they need to do to contribute to and to exploit this growing body of distributed mathematical knowledge. It will become essentially reading for all working mathematicians and mathematics students aspiring to take part in this new world of shared mathematical knowledge.

OMDoc is one of the first fruits of the Mathematical Knowledge Management (MKM) Network (`http://www.mkm-ig.org/`). This network combines researchers in mathematics, informatics and library science. It is attempting to realise the dream of creating a universal digital mathematics library of all mathematical knowledge accessible to all via the world-wide-web. Of course, this is one of those dreams that is never fully realised, but remains as a source of inspiration. Nevertheless, even its partial realisation would transform the way that mathematics is practised and learned. It would be a dynamic library, providing not just text, but allowing users to run computer software that would provide visualisations, calculate solutions, reveal counter-examples and prove theorems. It would not just be a passive source of knowledge but a partner in mathematical discovery. One major application of this library will be to teaching. Many of the participants in the MKM Network are building teaching aids that exploit the initial versions of the library. There will be a seamless transition between teaching aids and research assistants — as the library adjusts its contribution to match the mathematical user's current needs. The library will be freely available to all: all nations, all age groups and all ability levels.

I'm delighted to write this foreword to one of the first steps in realising this vision.

Alan Bundy, Edinburgh, 25. May 2006

Preface

Mathematics is one of the oldest areas of human knowledge[1]. It forms the basis most modern sciences, technology and engineering disciplines build upon it: Mathematics provides them with modeling tools like statistical analysis or differential equations. Inventions like public-key cryptography show that no part of mathematics is fundamentally inapplicable. Last, but not least, we teach mathematics to our students to develop abstract thinking and hone their reasoning skills.

However, mathematical knowledge is far too vast to be understood by one person, moreover, it has been estimated that the total amount of published mathematics doubles every ten–fifteen years [Odl95]. Thus the question of supporting the management and dissemination of mathematical knowledge is becoming ever more pressing but remains difficult: Even though mathematical knowledge can vary greatly in its presentation, level of formality and rigor, there is a level of deep semantic structure that is common to all forms of mathematics and that must be represented to capture the essence of the knowledge.

At the same time 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 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 connected by a commonly accepted distribution architecture, which makes the combined systems appear to the user as one homogeneous application. They will communicate with human users and amongst themselves by exchanging structured mathematical documents, whose document format makes the context of the communication and the meaning of the mathematical objects unambiguous.

Thus the inter-operation of mathematical services can be seen as a knowledge management task between software systems. On the other hand, mathematical knowledge management will almost certainly be web-based, distributed, modular, and integrated into the emerging math services architecture. So the two fields constrain and cross-fertilize each other at the same time. A shared fundamental task that has to be solved for the vision of a "web of mathematical knowledge" (MathWeb) to become reality is to define an open markup language for the mathematical objects and knowledge exchanged between mathematical services. The OMDoc format (Open Mathematical Documents) presented here is an answer to this challenge, it attempts to provide an infrastructure for the communication and storage of mathematical knowledge.

Mathematics – with its long tradition in the pursuit of conceptual clarity and representational rigor – is an interesting test case for general knowledge

[1] We find mathematical knowledge written down on Sumerian clay tablets, and even Euclid's *Elements*, an early rigorous development of a larger body of mathematics, is over 2000 years old.

management, since it abstracts from vagueness of other knowledge without limiting its inherent complexity. The concentration on mathematics in OMDoc and this book does not preclude applications in other areas. On the contrary, all the material directly extends to the STEM (science, technology, education, and mathematics) fields, once a certain level of conceptualization has been reached.

This book tries to be a one-stop information source about the OMDoc format, its applications, and best practices. It is intended for authors of mathematical documents and for application developers. The book is divided into four parts: an introduction to markup for mathematics (Part I), an OMDoc primer with paradigmatic examples for many kinds of mathematical documents (Part II), the rigorous specification of the OMDoc document format (Part III), and an XML document type definition and schema (Part IV).

The book can be read in multiple ways:

- for users that only need a casual exposure to the format, or authors that have a specific text category in mind, it may be best to look at the examples in the OMDoc primer (Part II of this book),
- for an in-depth account of the format and all the possibilities of modeling mathematical documents, the rigorous specification in Part III is indispensable. This is particularly true for application developers, who will also want to study the external resources, existing OMDoc applications and projects, in Part **??**.
- Application developers will also need to familiarize themselves with the OMDoc Schema in the Appendix.

Acknowledgments

Of course the OMDoc format has not been developed by one person alone. The original proposal was taken up by several research groups, most notably the Ωmega group at Saarland University, the Maya and ActiveMath projects at the German Research Center of Artificial Intelligence (DFKI), the MoWGLI EU Project, the RIACA group at the Technical University of Eindhoven, and the CourseCapsules project at Carnegie Mellon University. They discussed the initial proposals, represented their materials in OMDoc and in the process refined the format with numerous suggestions and discussions.

The author specifically would like to thank Serge Autexier, Bernd Krieg-Brückner, Olga Caprotti, David Carlisle, Claudio Sacerdoti Coen, Arjeh Cohen, Armin Fiedler, Andreas Franke, George Goguadze, Alberto González Palomo, Dieter Hutter, Andrea Kohlhase, Christoph Lange, Paul Libbrecht, Erica Melis, Till Mossakowski, Normen Müller, Immanuel Normann, Martijn Oostdijk, Martin Pollet, Julian Richardson, Manfred Riem, and Michel Vollebregt for their input, discussions, and feedback from implementations and applications.

Special thanks are due to Alan Bundy and Jörg Siekmann. The first triggered the work on OMDoc, has lent valuable insight over the years, and has graciously consented to write the foreword to this book. Jörg continually supported the OMDoc idea with his abundant and unwavering enthusiasm. In fact the very aim of the OMDoc format: openness, cooperation, and philosophic adequateness came from the spirit in his Ωmega group, which the author has had the privilege to belong to for more than 10 years.

The work presented in this book was supported by the "Deutsche Forschungsgemeinschaft" in the special research action "Resource-adaptive cognitive processes" (SFB 378), and a three-year Heisenberg Stipend to the author. Carnegie Mellon University, SRI International, and Jacobs University Bremen have supported the author while working on revisions for versions 1.1 to 1.3.

Contents

Part I

Setting the Stage for Open Mathematical Documents

In this part of the book we will look at the problem of marking up mathematical knowledge and mathematical documents in general, situate the OMDoc format, and compare it to other formats like OpenMath and MathML.

The OMDoc format is an open markup language for mathematical documents and the knowledge encapsulated in them. The representation in OMDoc makes the document content unambiguous and their context transparent.

OMDoc approaches this goal by embedding control codes into mathematical documents that identify the document structure, the meaning of text fragments, and their relation to other mathematical knowledge in a process called *document markup*. Document markup is a communication form that has existed for many years. Until the computerization of the printing industry, markup was primarily done by a copy editor writing instructions on a manuscript for a typesetter to follow. Over a period of time, a standard set of symbols was developed and used by copy editors to communicate with typesetters on the intended appearance of documents. As computers became widely available, authors began using word processing software to write and edit their documents. Each word processing program had its own method of markup to store and recall documents.

Ultimately, the goal of all markup is to help the recipient of the document better cope with the content by providing additional information e.g. by visual cues or explicit structuring elements. Mathematical texts are usually very carefully designed to give them a structure that supports understanding of the complex nature of the objects discussed and the argumentations about them. Such documents are usually structured according to the argument made and enhanced by specialized notation (mathematical formulae) for the particular objects.[2] In contrast, the structure of texts like novels or poems normally obey different (e.g. aesthetic) constraints.

In mathematical discourses, conventions about document form, numbering, typography, formula structure, choice of glyphs for concepts, etc. and the corresponding markup codes have evolved over a long scientific history and by now carry a lot of the information needed to understand a particular text. But since they pre-date the computer age, they were developed for the consumption by humans (mathematicians) and mainly with "ink-on-paper" representations (books, journals, letters) in mind, which turns out to be too limited in many ways.

In the age of Internet publication and mathematical software systems, the universal accessibility of the documents breaks an assumption implicit in the design of traditional mathematical documents: namely that the reader will come from the same (scientific) background as the author and will directly understand the notations and structural conventions used by the author. We can also rely less and less on the premise that mathematical documents are primarily for human consumption as mathematical software systems are more and more embedded into the process of doing mathematics. This, together with the fact that mathematical documents are primarily produced and stored on computers, places a much heavier burden on the markup format, since it has to make all of this implicit information explicit in the communication.

In the next two chapters we will set the stage for the OMDoc approach. We will first discuss general issues in markup formats (see Section 1.1), existing solutions (see Section 1.2), and the current XML-based framework for markup languages on the web (see Section 1.3). Then we will elaborate the special requirements for marking up the content of mathematics (see Chapter 2).

[2] Of course this holds not only for texts in pure mathematics, but for any argumentative text, including texts from the sciences and engineering disciplines. We will use the adjective "mathematical" in an inclusive way to make this distinction on text form, not strictly on the scientific labeling.

1

Document Markup for the Web

Document markup is the process of adding codes to a document to identify the structure of a document and to specify the format in which its fragments are to appear. We will discuss two conflicting aspects — structure and appearance — in document markup. As the Internet imposes special constraints imposed on markup formats, we will reflect its influence.

In the past few years the XML format has established itself as a general basis for markup languages. As OMDoc and all mathematical markup schemes discussed here are XML applications (instances of the XML framework), we will go more into the technical details to supply the technical prerequisites for understanding the specification. We will briefly mention XML validation and transformation tools, if the material reviewed in this section is not enough, we refer the reader to [Har01].

1.1 Structure vs. Appearance in Markup

Text processors and desktop publishing systems (think for example of Microsoft Word) are software systems aiming to produce "*ink-on-paper*" or "*pixel-on-screen*" representations of documents. They are very well-suited to execute typographic conventions for the appearance of documents. Their internal markup scheme mainly defines presentation traits like character position, font choice and characteristics, or page breaks. We will speak of **presentation markup** for such markup schemes. They are perfectly sufficient for producing high-quality presentations on paper or on screen, but for instance it does not support 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[1]) text processors. Here, the text form

[1] "What you see is what you get"; in the context of markup languages this means that the document markup codes are hidden from the user, who is presented with a presentation form of the text even during authoring.

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 call them **content markup** to distinguish them from presentation markup schemes, and discuss TEX/LATEX [Knu84; Lam94] as an example.

TEX is a typesetting markup language that uses explicit markup codes (strings beginning with a backslash) in a document, for instance, the markup `$\sqrt{\sin x}$` stands for the mathematical expression $\sqrt{\sin x}$ in TEX. To determine from this functional specification the visual form (e.g. the character placement and font information), we need a document formatting engine. This program will transform the document that contains the content markup (the "source" document) into a presentation markup scheme that specifies the appearance (the "target" document) like DVI [Knu84], POSTSCRIPT [Rei87], or PDF [**PDFReference**] that can directly be presented on paper or on screen. This two-stage approach allows the author to mark up the function of a text fragment and leave the conversion of this markup into presentation information to the formatter. The specific form of translation is either hard-wired into the formatter, or given externally in *style files* or *style sheets*.

LATEX [Lam94] is a comprehensive set of style files for the TEX formatter, the heading for a section with the title "The Joy of TEX" would be marked up as

```
\section[{\TeX}]{The Joy of {\TeX}\index{tex@\TeX}}\label{sec:TeX}
```

This piece of markup 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 glyph "TEX" is inserted into the index, where the word `tex` would have been, and the section number can be referred to using the label `sec:TeX`. Note that renumbering is not a problem in this approach, since the actual numbers are only inferred by the formatter at run-time. 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. Pure presentation markup schemes like DVI or POSTSCRIPT are normally only used for document delivery. On the other hand, many form-oriented markup schemes allow to "fine-tune" documents by directly controlling presentation. For instance, LATEX allows to specify traits such as font size information, or using

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

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

1.2 Markup for the World Wide Web

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

- the size, resolution, and color depth of a given screen are not known at the time the document is marked up,
- the structure of a text is no longer limited to a linear text with (e.g. numbered) cross-references as in a traditional book or article: Internet documents are usually hypertexts,
- the computational resources of the computer driving the screen are not known beforehand. Therefore the distribution of work (e.g. formatting steps) between the client and the server has to be determined at run-time. Finally, the related problem that
- the bandwidth of the Internet is ever-growing but always limited.

These issues impose somewhat conflicting demands on markup languages for the Web. The first two seem to favor content markup languages, since low-level presentational traits like glyph placement and font availability cannot be pre-meditated on the server. However, the amount of formatting that can be delegated to the client, and the availability of style files is limited by the latter two concerns.

In response the "Hypertext Markup Language" (HTML [RHJ98]) evolved as the original markup format for the World Wide Web. This is a markup scheme that 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. It ensures a high degree of reusability of documents on the Internet while conserving bandwidth, so that HTML carries most of the text markup on the Internet today.

The major innovation in HTML was the use of **uniform resource locators** (**URL**) to reference documents provided by web servers. URLs are strings in a special format that can be interpreted by browsers or other web agents to request documents from web servers, e.g. to be displayed to the user in the browser as a new node in the current hypertext document. Since URLs are global references, they are the means that make the Internet into a "*world-wide*" web (of references). Since uniform resource *locators* are closely tied to the physical location of a document on the Internet, which can change over time, they have since been generalized to **uniform resource identifier** (**URI**; see [BLFM98]). These are strings of similar structure, that only identify resources on the Internet, see [Har01], i.e. their structure need not be directly translatable to an Internet location (we call this act **de-referencing**). Indeed, URIs need not even correspond to a physical manifestation of a resource at all, they can identify a virtual resource, that is produced by a web service on demand.

The concrete syntax and architecture of HTML is derived from the "Simple Generalized Markup Language" SGML [Gol90], which is similar to TEX/LaTeX in spirit, but tries to give the markup scheme a more declarative semantics (as opposed to the purely procedural – and rather baroque – semantics of TEX) to make it simpler to reason about (and thus reuse) documents. In particular unlike TEX, SGML separates content markup codes from directives to the formatting engine. SGML has a separate style sheet language DSSSL [DuC97], which was not adopted by HTML, because of resource limitations in the client. Instead, HTML has been augmented with its own (limited) style sheet language CSS [Bos+98] that is executed by the browser.

1.3 XML, the eXtensible Markup Language

The need for content markup schemes for maintaining documents on the server, as well as for specialized presentation of certain text parts (e.g. for mathematical or chemical formulae), has led to a profusion of markup schemes for the Internet, most of which share the basic SGML syntax with HTML. To organize this zoo of markup languages, the World Wide Web Consortium (W3C [W3c], an international interest group of universities and web industry) has developed a language framework for Internet markup languages called XML (e<u>X</u>tensible <u>M</u>arkup <u>L</u>anguage) [BPSM97]. XML is a set of grammar rules that allows to interpret certain sequences of Unicode [Inc03] characters as document trees. These grammar rules are shared by all XML-based markup languages (called XML applications) and are very well-supported by a great variety of XML processors. The XML format is accompanied by a set of specialized vocabularies (most of them XML applications) that standardize various aspects of document management and web services. These are canonicalized by the W3C as "recommendations". We will briefly review the ones that are relevant for understanding the OMDoc format and make the book self-contained. For details see one of the many XML books, e.g. [Har01].

1.3.1 XML Document Trees

Conceptually speaking, XML views a document as a tree whose nodes consist of elements, attributes, text nodes, namespace declarations, XML comments, etc. (see Figure 1.1 for an example[2]). For communication this tree is serialized

[2] This tree representation glosses over namespace nodes in the tree, but the conceptual tree is sufficient for the application in this book.

```
<omtext xml:id="foo" xmlns="http://omdoc.org/ns"
        xmlns:om="http://www.openmath.org/OpenMath">
  <CMP xml:lang='en'><xhtml:p>
   The number
   <om:OMOBJ><om:OMS cd="nums1" name="pi"/><om:OMOBJ>
   is  irrational .
  </xhtml:p></CMP>
</omtext>
```

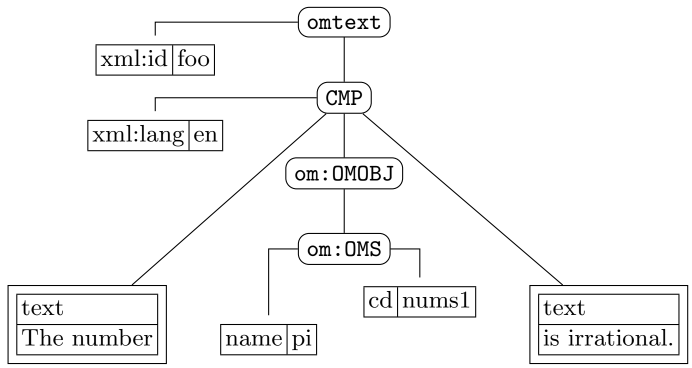


Fig. 1.1. An XML Document as a Tree

into a balanced bracketing structure (see the listing at the top of Figure 1.1), where an element el is represented by the brackets <el> (called the **opening tag**) and </el> (called the **closing tag**). The leaves of the tree are represented by **empty elements** (serialized as <el></el>, which can be abbreviated as <el/>), and text nodes (serialized as a sequence of UNICODE characters). An element node can be annotated by further information using **attribute nodes** — serialized as an **attribute** in its opening tag: for instance <el visible="no"> might add the information for a formatting engine to hide this element. As a document is a tree, the XML specification mandates that there must be a unique **document root**.

Let us now come to a feature that we have glossed over so far: XML **namespaces** [BHL99]. In many XML applications, we need to mix several XML vocabularies or languages. In our example in Figure 1.1 we have three: the OMDOC vocabulary with the elements omtext and CMP, the OPENMATH vocabulary with the elements om:OMOBJ and om:OMS, and the general XML vocabulary for the attributes xml:id and xml:lang.

To allow a safe mixing of independent XML vocabularies, XML can associate elements and attributes[3] with a **namespace**, which is simply a URI that uniquely identifies the intended vocabulary[4]. In XML syntax, namespace membership is represented by namespace declarations and qualified names.

[3] Traditionally most XML applications use attributes that are not namespaced.

[4] Note that it need not be a valid URL (uniform resource locator; i.e. a pointer to a document provided by a web server).

A **namespace declaration** is a pseudo-attribute with name `xmlns` whose value is a namespace URI ⟪*nsURI*⟫ (see e.g. the first line in Figure 1.1). In a nutshell, a namespace declaration specifies that this element and all its descendants are in the namespace ⟪*nsURI*⟫, unless they have a namespace declaration of their own or there is a namespace declaration in a closer ancestor that overwrites it.

Similarly, a **namespace abbreviation** can be declared on any element by a pseudo-attribute of the form `xmlns:`⟪*nsa*⟫`="`⟪*nsUR*⟫`"`, where ⟪*nsa*⟫ is an XML simple name, and ⟪*nsURI*⟫ is the namespace URI. In the scope of this declaration (in all descendants, where it is not overwritten) we can specify that an element or attribute is in the namespace ⟪*nsURI*⟫ by using a **qualified name**: a pair ⟪*nsa*⟫`:`⟪*el*⟫, where ⟪*nsa*⟫ is a namespace abbreviation and ⟪*el*⟫ is a simple name (i.e. one that does not contain a colon). In Figure 1.1, we have a namespace abbreviation in the second line, which is used for the OpenMath objects in line five. This rule has one exception: the namespace abbreviation `xml` is reserved for the XML namespace and does not have to be declared.

Since XML elements only encode trees, the distribution of whitespace (including line-feeds) in non-text elements has no meaning in XML, and can therefore be added and deleted without effecting the semantics. XML considers anything between `<!--` and `-->` in a document as a comment. They should be used with care, since they are not necessarily passed on by the XML parser, and therefore might not survive processing by XML applications.

Material that is relevant to the document, but not valid XML, e.g. binary data or data that contains angle brackets or elements that are unbalanced or not part of the XML application can be encoded by embedding it into **`CDATA` sections**. A `CDATA` section begins with the string `<[CDATA[` and suspends the XML parser until the string `]]>` is found. The result of parsing a `CDATA` section is equivalent to escaping the five XML-specific characters `<`, `>` `"`, `'`, and `&` to the XML entities `<`, `>`, `"`, `'`, and `&`. For instance, we have the following correspondence between a `CDATA` section and XML-escaped content:

```
<[CDATA[a<b<sup>3</sup>]]>    ≙    a&lt;b&lt;sup&gt;3&lt;/sup&gt;
```

As a consequence, an XML application is free to choose the form of its output and the particular form should not be relied upon.

1.3.2 Validating XML Documents

XML offers various mechanisms for specifying a subset of trees (or well-bracketed XML documents) as admissible in a given XML application: the most commonly used ones are **document type definition**s (**DTD** [BPSM97]), XML **schemata** [Xml], and RelaxNG schemata [Vli03]. All of these are context-free grammars for trees, that can be used by a **validating parser** to reject XML documents that do not conform. Note that DTDs and schemata cannot enforce all constraints that a particular XML application may want to impose on documents. Therefore validation is only a necessary condition for **validity** with respect to that application. Since the XML schema languages can express slightly stronger sets of constraints and are namespace-aware, they allow stronger document validation, and usually take normative precedence over the DTD if present.

Listing 1.1 shows part of an OMDoc document. The first line identifies the document as an XML document (version 1.0 of the XML specification). The second and third lines constitute the **document type declaration** which specifies the DTD and the document root element. In this case the `omdoc` element starting in line 4 is the root element and will be validated against the DTD identified by the **public Identifier**[5] in line two and which can be found at the URI in line three. See Chapter **??** for an in-depth discussion of the OMDoc DTD and validation.

Listing 1.1. The Structure of an XML Document with DTD

```
<?xml version="1.0"?>
<!DOCTYPE omdoc PUBLIC "−//OMDoc//DTD OMDoc V1.3//EN"
                       "http://omdoc.org/dtd/omdoc.dtd">
4 <omdoc xml:id="example−omdoc" xmlns="http://omdoc.org/ns">
...
</omdoc>
```

Note that it is not mandatory to have a document type declaration in an XML document, or that an XML parser even read it (we call an XML parser **validating** if it does). If no document type declaration is present, then a parser will just check for XML-well-formedness, and possibly rely on some schema for further validation[6]. Note that if a validating parser reads an XML document with a document type declaration, then it must process it and validate the document.

But a DTD not only contains information for validation, it also

declares XML entities XML entities are strings of the form `&`⟪*abbr*⟫`;`, which abbreviate sequences of Unicode characters and are expanded by the parser as it reads the document.

supplies default values for attributes which are added to the representation of the parsed document by the parser as it reads the document.

declares types of attributes This is is relevant for attribute types `ID` and `IDREF`. The former are required to be document-unique (as well as being XML simple names [BPSM97, section 2.3]) and the latter must point to an existing `ID`-type attribute in the same document.

---

[5] A string that allows to identify an XML resource, it can be mapped to a concrete URI via the XML catalog; see Section **??** for details.

[6] Note that RelaxNG schemata do not have a specified in-document means for associating a schema with elements. For the way to associate an XML schema with a document we refer to XML schema recommendation [Xml] or the XML literature.

`ID`-type attributes are commonly used to identify elements in XML documents (see the discussion in Subsection 1.3.3), which raises a subtle point with respect to DTDs. If an XML document is processed without a document type declaration or by a non-validating parser, the information which attributes are `ID`-type ones is lost, and referencing does not work as as expected. Fortunately, there is a recent W3C-solution to this problem: Following the XML ID recommendation [MVW05] XML parsers must recognize attributes of the form `xml:id` as `ID`-type attributes, even if no DTD is present.

However DTDs may still serve an important role, even if they are superseded by schema-based approaches for pure validation. For instance a format like Presentation-MathML (see Subsection 2.1.1) seems dependent on a DTD, since it needs to define a rich set of mnemonic entities for mathematical symbols in Unicode and uses `ID`-type attributes for cross-referencing. Formats like Content-MathML (Subsection 2.1.1), OpenMath (Subsection 2.1.2) or OMDoc proper can live without DTDs, since they do not.

1.3.3 XML Fragments and URI References

As documents are construed as trees in XML, the notion of a document fragment becomes definable simply as a sets of well-formed sub-trees. Building on this, URLs and URIs can be extended to references of document fragments. These **URI reference**s are traditionally considered to consist of two parts: A proper URI and a specific **fragment identifier** separated by the hash character `#`. The URI identifies an XML document on the web, whereas the fragment identifier identifies a specific fragment of that document.

XML provides the XPointer framework [Gro+03a] for fragment identifiers. It specifies multiple schemes for fragment identifiers. Fragment identifiers of the form `xpointer(`⟪*path*⟫`)` use an XPath [CD99] expression ⟪*path*⟫ to specify a path through the document tree leading to the desired element (see [DMJ03]). Fragment identifiers in the `element()` scheme [Gro+03b] use expressions of the form `element(`⟪*cpath*⟫`)`, where ⟪*cpath*⟫ is an `ID`-type identifier together with a simple child-path; e.g. `element(foo/3/7)` identifies the 7^{th} child of the 3^{rd} child of the (unique) element that has ID-type attribute with value `foo`.

URI references of the form ⟪*uri*⟫`#`⟪*id*⟫ as they are used in HTML to refer to named anchors (<a name="⟪*id*⟫"/>) are regained as a special case (the shorthand `xpointer`): If ⟪*uri*⟫ is a URI of an XML document D then ⟪*uri*⟫`#`⟪*id*⟫ refers to the unique element in D, that has an attribute of type `ID` with value ⟪*id*⟫.

1.3.4 Summary

In summary, XML provides a widely standardized infrastructure for defining Internet markup languages based on tree structures rather than on sequences of characters. XML processors like parsers, serializers, XML databases, and XSLT transformation engines are widely deployed and incorporated into many programming languages. Building XML applications on top of this infrastructure frees the implementers from dealing with low-level details of parsing, validation, and mass storage. It is no surprise that XML has become one of the most successful interoperability formats in information technology.

Note that the use of XML does not give any support for mathematics in itself, since the tree models are completely general. It is the role of specific XML applications like the ones we will present in the next two chapters to specialize the XML tree structures to representations that can be interpreted as mathematical objects and documents.

2

Markup for Mathematical Knowledge

Mathematicians make use of various kinds of documents (e.g. e-mails, letters, pre-prints, journal articles, and textbooks) for communicating mathematical knowledge. Such documents employ specialized notational conventions and visual representations to convey the mathematical knowledge reliably and efficiently. The respective representations are supported by pertinent markup systems like TEX/LATEX.

Even though mathematical documents can vary greatly in their level of presentation, formality and rigor, there is a level of deep semantic structure that is common to all forms of mathematics and that must be represented to capture the essence of the knowledge. As John R. Pierce has written in his book on communication theory [Pie80], mathematics and its notations should not be viewed as one and the same thing. Mathematical ideas exist independently of the notations that represent them. However, the relation between meaning and notation is subtle, and part of the power of mathematics to describe and analyze derives from its ability to represent and manipulate ideas in symbolic form. The challenge in putting mathematics on the World Wide Web is to capture both notation and content (that is, meaning) in such a way that documents can utilize the highly-evolved notational forms of written and printed mathematics, and the potential for interconnectivity in electronic media.

In this chapter, we present the state of the art for representing mathematical documents on the web and analyze what is missing to mark up mathematical knowledge. We posit that there are three levels of information in mathematical knowledge: formulae, mathematical statements, and the large-scale theory structure (constructing the context of mathematical knowledge). The first two are immediately visible in marked up mathematics, e.g. textbooks, the third is largely left to an implicit meta-level of mathematical communication, or the organization of mathematical libraries. We will discuss these three levels in the next sections.

2.1 Mathematical Objects and Formulae

A distinguishing feature of mathematical documents is the use of a complex and highly evolved system of two-dimensional symbolic notations, commonly called (mathematical) **formulae**. Formulae serve as representations of mathematical objects, such as functions, groups, or differential equations, and also of statements about them, like the "Fundamental Theorem of Algebra".

The two best-known open markup formats for representing mathematical formulae for the Web are MathML [Aus+03a] and OpenMath [Bus+04]. There are various other formats that are proprietary or based on specific mathematical software packages like Wolfram Research's Mathematica® [Wol02]. We will not concern ourselves with them, since we are only interested in open formats. Furthermore, we will only give a general overview for the open formats here to survey the state of the art, since content MathML and OpenMath are used for formula representation in the OMDoc format and thus the technical details of the two markup schemes are covered in more detail in the OMDoc specification in Chapter 13. Figure 2.1 gives an overview over the current state of the standardization activities.

language	MathML	OpenMath
by	W3C Math WG	OpenMath society
origin	math for HTML	integration of CAS
coverage	content + presentation; K-14	content; extensible
status	Version 2.2e (VI 2003)	Version 2 (VI 2004)
activity	maintenance	maintenance
Info	`http://w3c.org/Math/`	`http://www.openmath.org/`

Fig. 2.1. The Status of Markup Standardization for Mathematical Formulae

OpenMath was originally a development driven mainly by the Computer Algebra community in Europe trying to standardize the communication of mathematical objects between Computer Algebra Systems. The format has been discussed in a series of workshops and has been funded by a series of grants by the European Union. This process led to the OpenMath 1 standard in June 1999 and eventually to the incorporation of the OpenMath society as the institutional guardian of the OpenMath standard. MathML has developed out of the effort to include presentation primitives for mathematical notation (in TeX quality) into HTML, and was the first XML application to reach recommendation status[1] at the W3C [Bus+99].

[1] As such, MathML played a great role as technology driver in the development of XML. This role gives MathML a somewhat peculiar status at the W3C; it is the only "vertical" (application/domain-driven) XML application standardized

The competition and collaboration between these two approaches to representation of mathematical formulae and objects has led to a large overlap between the two developer communities. MathML deals principally with the *presentation* of mathematical objects, while OpenMath is solely concerned with their semantic meaning or *content*. While MathML does have some limited facilities for dealing with content, it also allows semantic information encoded in OpenMath to be embedded inside a MathML structure. Thus the two technologies may be seen as highly compatible[2] and complementary (in aim).

2.1.1 MathML

> MathML is an XML application for describing mathematical *notation* and capturing both its *structure* and *content*. The goal of MathML is to enable mathematics to be served, received, and processed on the World Wide Web, just as HTML has enabled this functionality for text.
>
> *from the MathML2 Recommendation* [Aus+03a]

To reach this goal, MathML offers two sub-languages: Presentation-MathML for marking up the two-dimensional, visual appearance of mathematical formulae, and Content-MathML as a markup infrastructure for the functional structure of mathematical formulae.

To mark up the visual appearance of formulae Presentation-MathML represents mathematical formulae as a tree of layout primitives. For instance the expression $\frac{3}{x+2}$ would be represented as the layout tree in Figure 2.2. The layout primitives arrange "inner boxes" (given in black) and provide an outer box (given in gray here) for the next level of layout. In Figure 2.2 we see the general layout schemata for numbers (`m:mn`), identifiers (`m:mi`), operators (`m:mo`), bracketed groups (`m:mfence`), and fractions (`m:mfrac`); others include horizontal grouping (`m:mrow`), roots (`m:mroot`), scripts (`m:msup`, `m:msub`, `m:msubsup`), bars and arrows (`m:munder`, `m:mover`, `m:munderover`), and scoped CSS styling (`m:mstyle`). Mathematical symbols are taken from Unicode and provided with special mnemonic entities by the MathML DTD, e.g. `∑` for Σ.

Since the aim of MathML is to do most of the formatting inside the browser, where resource considerations play a large role, it restricts itself to a fixed set of mathematical concepts – the K-14 fragment (Kindergarten to 14^{th} grade; i.e. undergraduate college level) of mathematics. K-14 contains a large set of commonly used glyphs for mathematical symbols and very general and

by the W3C, which otherwise concentrates on "horizontal" (technology-driven) standards.

[2] e.g. MathML is the preferred presentation format for OpenMath objects and OpenMath content dictionaries are the primary specification language for MathML semantics.

```
<m:mfrac>
 <m:mn>3</m:mn>
 <m:mfenced>
   <m:mi>x</m:mi>
   <m:mo>+</m:mo>
   <m:mn>2</m:mn>
 </m:mfenced>
</m:mfrac>
```

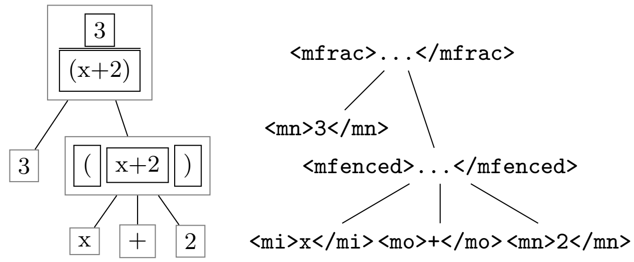


Fig. 2.2. The Layout Tree for the Formula $\frac{3}{x+2}$

powerful presentation primitives, similar to those that make up the lower level of TEX. However, it does not offer the programming language features of TEX[3] for the obvious computing resource considerations. Presentation-MathML is supported by current versions of the browsers Amaya [Vat], MS Internet Explorer [Cor] (via the MathPlayer plug-in [Mat]), and Mozilla [Org].

MathML also offers content markup for mathematical formulae, a sublanguage called **Content-MathML** to contrast it from the **Presentation-MathML** described above. Here, a mathematical formula is represented as a tree as well, but instead of marking up the visual appearance, we mark up the functional structure. For our example $\frac{3}{x+2}$ we obtain the tree in Figure 2.3, where we use @ as the function application operator (it interprets the first child as a function and applies it to the rest of the children as arguments).

```
<m:apply>
  <m:divides/>
  <m:cn>3</m:cn>
  <m:apply>
    <m:plus/>
    <m:ci>x</m:ci>
    <m:cn>2</m:cn>
  </m:apply>
</m:apply>
```

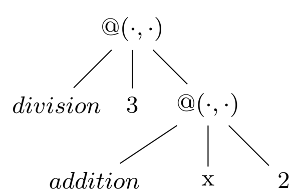


Fig. 2.3. The functional Structure of $\frac{3}{x+2}$

Content-MathML offers around 80 specialized elements for the most common K-14 functions and individuals. In Figure 2.3 we see function application (`m:apply`), content identifiers (`m:ci`), content numbers (`m:cn`) and the functions for division (`m:divide`) and addition (`m:plus`).

---

[3] 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 CSS and XSLT style languages it inherits from XML.

Finally, MathML offers a specialized `m:semantics` element that allows to annotate MathML formulae with alternative representations. This feature can be used to provide combined content- and presentation-MathML representations. Figure 2.4 shows an example of this for our expression $\frac{3}{x+2}$. The outermost `m:semantics` element is used for mixing presentation and content markup. The first child of the `m:semantics` element contains Presentation-MathML (this is used by the MathML-aware browser), the subsequent `m:annotation-xml` element contains Content-MathML markup for the same formula. Corresponding sub-expressions are co-referenced by cross-references: The presentation element carries an `id` attribute, which serves as the target for an `xlink:href` attribute in the content markup. This technique is called parallel markup, it allows to select logical sub-expressions by selecting layout sub-schemata in the browser, e.g. for copy and paste. Note that a `m:semantics` element can have more than one `m:annotation-xml` child, so that other content formats such as OpenMath can also be incorporated.

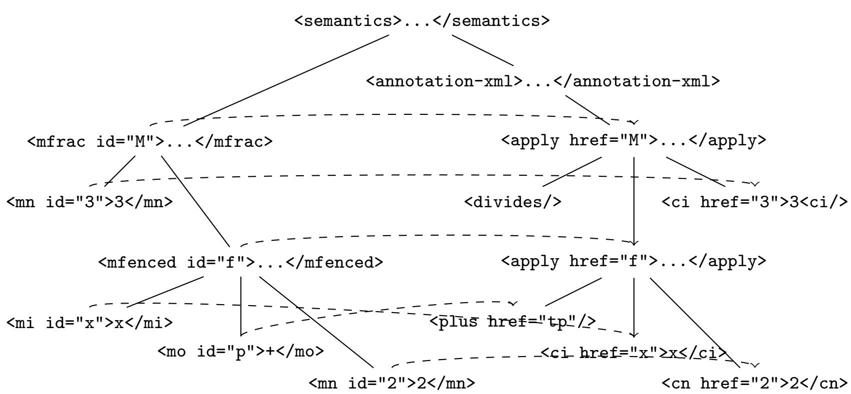


Fig. 2.4. Mixing Presentation and Content-MathML

2.1.2 OpenMath

> [...] OpenMath: a standard for the representation and communication of mathematical objects. [...]
> OpenMath allows the *meaning* of an object to be encoded rather than just a visual representation. It is designed to allow the free exchange of mathematical objects between software systems and human beings. On the worldwide web it is designed to allow mathematical expressions embedded in web pages to be manipulated and computed with in a meaningful and correct way. It is designed to be machine-generatable and machine-readable, rather than written by hand.
>
> *from the* OpenMath*2 Standard* [Bus+04]

Driven by the intention of representing the *meaning* of mathematical objects expressed in the quote above, the OpenMath format is not primarily an XML application. Rather, OpenMath defines an abstract (mathematical) object model for mathematical objects and specifies an XML encoding (and a binary[4] encoding) for that[5].

The central construct of OpenMath is that of an **OpenMath object** (realized by the element `om:OMOBJ` in the XML encoding), which has a tree-like representation made up of applications (`om:OMA`), binding structures (`om:OMBIND` using `om:OMBVAR` to specify the bound variables[6]), variables (`om:OMV`), and symbols (`om:OMS`).

The handling of symbols — which are used to represent the multitude of mathematical domain constants — is maybe the largest difference between OpenMath and Content-MathML. Instead of providing elements for all K-14 concepts, the OpenMath standard adds an extension mechanism for mathematical concepts, the **content dictionaries**. These are machine-readable documents that define the meaning of mathematical concepts expressed by OpenMath symbols. Just like the library mechanism of the C programming language, they allow OpenMath to externalize the definition of extended language concepts. As a consequence, K-14 need not be part of the OpenMath language, but can be defined in a set of content dictionaries (see [OMCD]).

The `om:OMS` element carries the attributes `cd` and `name`. The `name` attribute gives the name of the symbol, the `cd` attribute specifies the content dictionary.

4 The binary encoding allows to optimize encoding size and (more importantly) parsing time for large OpenMath objects. The binary encoding for OpenMath objects will not play a role for the OMDoc format, so we will not pursue this here.

5 The MathML specification is very vague on what the meaning of Content-MathML fragments might be; we have to assume that its XML document object model [Urlc] or the or its infoset [CT04] must be.

6 Binding structures are somewhat awkwardly realized via the `m:apply` element with an `m:bvar` child in Content-MathML.

As variables do not carry a meaning independent of their local content, `om:OMV` only carries a `name` attribute. See Listing 2.1 for an example that uses most of the elements.

Listing 2.1. OpenMath Representation of $\forall a, b.a + b = b + a$

```
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMBIND cdbase="http://www.openmath.org/cd">
    <OMS cd="quant1" name="forall"/>
    <OMBVAR><OMV name="a"/><OMV name="b"/></OMBVAR>
    <OMA><OMS cd="relation" name="eq"/>
      <OMA><OMS cd="arith1" name="plus"/>
        <OMV name="a"/>
        <OMV name="b"/>
      </OMA>
      <OMA><OMS cd="arith1" name="plus"/>
        <OMV name="b"/>
        <OMV name="a"/>
      </OMA>
    </OMA>
  </OMBIND>
</OMOBJ>
```

Listing 2.1 shows the XML encoding of the law of commutativity for addition (the formula $\forall a, b.a + b = b + a$) in OpenMath. Note that as we have discussed above, this representation is not self-contained but relies on the availability of content dictionaries `quant1`, `relation1`, and `arith1`. Note that in this example they can be accessed via the URL specified in the `cdbase` attribute, but in general, the content dictionaries are only used for *identification of symbol*s. In particular, in the classical OpenMath model, content dictionaries are only viewed as a resource for system developers, who use them as a reference decide which symbol to use in an export/import facility for a computer algebra system. In the communication between mathematical software systems, they are no longer needed: If two systems agree on a set of content dictionaries, then they agree on the meaning of all OpenMath objects that can be constructed using their symbols (the meaning of applications and bindings is known from the folklore).

The content dictionary architecture is the greatest strength of the OpenMath format. It establishes an object model and XML encoding based on what we call "semantics by pointing". Two OpenMath objects have the same meaning in this model, iff they have the same structure and all symbols point to the same content dictionaries[7].

In the standard encoding of OpenMath content dictionary, the meaning of a symbol is specified by a set of

"formal mathematical properties" The `omcd:FMP` element contains an OpenMath object that expresses the desired property.

[7] Note that we can interpret the Content-MathML model as a "semantics by pointing" model as well. Only that here the K-14 elements do not point to machine-readable content dictionaries, but at the (human-readable) MathML specification, which specifies their meaning.

“commented mathematical properties” The omcd:CMP element contains a natural language description of a desired property.

For instance, the specification in Listing 2.2 is part of the standard OpenMath content dictionary arith1.ocd [OMCD] for the elementary arithmetic operations.[8]

Listing 2.2. Part of the OpenMath Content Dictionary arith1.

```
<CDDefinition>
  <Name>plus</Name>
  <CDDescription>
    The symbol representing an n−ary commutative function plus.
  </CDDescription>
  <CMP><xhtml:p>for all a,b | a + b = b + a</xhtml:p></CMP>
  <FMP>∀a, b.a + b = b + a</FMP>
</CDDefinition>
```

On the other hand, the content dictionary encoding defined in the OpenMath standard (and the particular content dictionaries blessed by the OpenMath society) are the greatest weakness of OpenMath. The represent the knowledge in a very unstructured way — to name just a few problems:

- in the omcd:CMP, we can only make use of ASCII representation of formulae.
- The relation between a particular omcd:CMP and omcd:FMP elements is unclear.
- For properties like the distributivity of addition over multiplication it is unclear, whether we should express this in the definition of the symbol plus or the symbol times.
- Are all properties constitutive for the meaning of the symbol? Should they be verified for an implementation of a content dictionary?
- What is the relationship between content dictionaries? Are they translation-equivalent? Does one entail the other?

The OpenMath2 standards acknowledges these problems and explicitly opens up the content dictionary format allowing other representations that meet certain minimal criteria relegating the standard encoding above to a reference implementation of the minimal model.

We will analyze the questions raised above from a general standpoint when discussing the remaining two levels of mathematical knowledge. This analysis constitutes the basic intuitions for the OMDoc format.

[8] The content of the omcd:FMP element is actually the OpenMath object in the representation in Listing 2.1, we have abbreviated it here in the usual mathematical notation, and we will keep doing this in the remaining document: wherever an XML element in a figure contains mathematical notation, it stands for the corresponding OpenMath element.

2.2 Mathematical Texts and Statements

The mathematical markup languages OpenMath and MathML we have discussed in the last section have dealt with mathematical objects and formulae. The formats either specify the semantics of the mathematical object involved in the standards document itself (MathML) or in a fixed set of generally agreed-upon documents (OpenMath content dictionaries). In both cases, the mathematical knowledge involved is relatively fixed. Even in the case of OpenMath, which has an extensible library mechanism, the content dictionaries are not in themselves objects of communication (they are mainly background reference for the implementation of OpenMath interfaces).

For the communication among mathematicians (rather than computation systems) this level of support is insufficient, because the mathematical knowledge expressed in definitions, theorems (stating properties of defined objects), their proofs, and even whole mathematical theories is the primary focus of mathematical communication. For content markup of mathematical knowledge, we have to turn implicit or presentational structuring devices in mathematical documents into explicit ones. For instance, **mathematical statements** like the ones in the document fragment in Figure 2.5 are delimited by keywords (e.g. **Definition**, **Lemma** and $\Box$) or by changes in text font.

Definition 3.2.5 (Monoid)
A monoid is a semigroup $S = (G, \circ)$ with an element $e \in G$, such that $e \circ x = x$ for all $x \in G$. e is called a left unit of S.

Lemma 3.2.6
A monoid has at most one left unit.
Proof: We assume that there is another left unit f ...
This contradicts our assumption, so we have proven the claim. $\Box$

Fig. 2.5. A Fragment of a Traditional Mathematical Document

Of course, the content of a mathematical statement, e.g. the statement of an assertion that "addition is commutative" can be expressed by a Content-MathML or OpenMath formula like the one in Listing 2.1, but the information that this formula is a theorem that has a proof, cannot be directly expressed without extending the formalism. Even formalizations of mathematics like Russell and Whitehead's famous "Principia Mathematica" [WR10] treat this information on the meta-level. If we are willing to extend the mathematical formalism to include primitives for such information, we arrive at formalisms called **logical frameworks** (see [Pfe01] for an overview), where they are treated as the primary objects of study. The most prevalent approach here uses the "formulae as types" idea that delegates mathematical formulae to the status of types. Logical frameworks capture mathematical statements in formulae and as such can be expressed in Content-MathML or OpenMath. However, this approach relies on full formalization of the mathematical content, and cannot be directly used to capture mathematical practice. In particular, the gap between formal mathematics and informal (but rigorous) treatments of mathematics that rely on natural language as we find them in textbooks and journal articles is wide. The formalization process is so tedious, that it is seldom executed in practice (the "Principia Mathematica" and the Mizar mathematical library [Miz] are solitary examples).

2.3 Large-Scale Structure and Context in Mathematics

The large-scale structure of mathematical knowledge is much less apparent than that for formulae and even statements. Experienced mathematicians are nonetheless aware of it, and use it for navigating the vast space of mathematical knowledge and to anchor their communication.

Much of this structure can be found in networks of **mathematical theories**: groups of mathematical statements, e.g. those in a monograph "Introduction to Group Theory" or a chapter or section in a textbook. The relations among such theories are described in the text, sometimes supported by mathematical statements called representation theorems. We can observe that mathematical texts can only be understood with respect to a particular mathematical context given by a theory which the reader can usually infer from the document. The context can be stated explicitly (e.g. by the title of a book) or implicitly (e.g. by the fact that the e-mail comes from a person that we know works on finite groups, and that she is talking about math).

If we make the structure of the context as explicit as the structure of the mathematical objects (we will speak of **context markup**), then mathematical software systems will be able to provide novel services that rely on this structure. We contend that without an explicit representation of context structure, tasks like semantics-based searching and navigation or object classification can only be performed by human mathematicians that can understand the implicitly given structure.

Mathematical theories have been studied by mathematicians and logicians in the search of a rigorous foundation for mathematical practice. They have been formalized as collections of symbol declarations — giving names to mathematical objects that are particular to the theory — and logical formulae, which state the laws governing the properties of the theory. A key research question was to determine conditions for the consistency of mathematical theories. In inconsistent theories all statements are vacuously valid[9], and therefore only consistent theories make interesting statements about mathematical objects.

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

It is one of the critical observations of meta-mathematics that theories can be extended without endangering consistency, if the added formulae can be proven from the formulae already in the theory (such formulae are called theorems). As a consequence, consistency of a theory can be determined by examining the **axiom**s (formulae without a proof) alone. Thus the role of proofs is twofold, they allow to push back the assumptions about the world to simpler and simpler axioms, and they allow to test the model by deriving consequences of these basic assumptions that can be tested against the data.

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

Even though all of this has theoretically been known to (meta)-mathematicians for almost a century, it has only been an explicit object of formal study and exploited by mathematical software systems in the last decades. Much of the meta-mathematics has been formally studied in the context of proof development systems like AutoMath [Bru80] NuPrL [Con+86], Hol [GM93], Mizar [Rud92] and Ωmega [Ben+97] which utilize strong logical systems that allow to express both mathematical statements and proofs as mathematical objects. Some systems like Isabelle [PN90] and Twelf [Pfe91] even allow the specification of the logic language itself, in which the reasoning takes place. Such semi-automated theorem proving systems have been used to formalize substantial parts of mathematics and mechanically verify many theorems in the respective areas. These systems usually come with a library system that manages and structures the body of mathematical knowledge formalized in the system so far.

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

Even though there have been standardization efforts, the most recent one being the Casl standard (Common Algebraic Specification Language; see [Mos04]) there have been no efforts of developing this into a general markup language for mathematics with attention to web communication and standards. The OMDoc format attempts to provide a content-oriented markup scheme that supports all the aspects and structure of mathematical knowledge we have discussed in this section. Before we define the language in the next chapter, we will briefly go over the consequences of adopting a markup language like OMDoc as a standard for web-based mathematics.

3

OMDoc: Open Mathematical Documents

Based on the analysis of the structure inherent in mathematical knowledge and existing content markup systems for mathematics we will now briefly introduce basic design assumptions and the development history of the OMDoc format, situate it, and discuss possible applications.

3.1 A Brief History of the OMDoc Format

OMDoc initially developed from the quest for a solution of the problem of representing knowledge on the one hand and integrating external mathematical reasoning systems in the ΩMEGA project at Saarland University on the other. ΩMEGA [Sie+02] is a large-scale proof development environment that integrates various reasoning engines (automated theorem provers, decision procedures, computer algebra systems) via knowledge-based proof planning with the aim of creating a mathematical assistant system.

3.1.1 The Design Problem

One of the hard practical problems of building such systems is to represent, provision, and manage the relevant (factual, tactic, and intuitive) knowledge human mathematicians use in developing mathematical theories and proofs: Knowledge-based reasoning systems use explicit representations of this knowledge to automate the search for a proof, and before a system can be applied to a mathematical domain it must be formalized, the proof tactics of this domain must be identified, and the intuitions of when to use which tactic must be coaxed from practitioners. Ideally, as a valuable and expensive resource, this knowledge would be shared between mathematical assistant systems to be able to compare the relative strength of the systems and to enhance practical coverage. This poses the problem that the knowledge must be represented at a level that would accommodate the different systems' representational quirks and bridge between them.

Developing an agent-oriented framework for distributed reasoning via remote procedure calls to achieve system scalability (MathWeb-SB [FK99; ZK02]; see Chapter 9 for an OMDoc-based reformulation) revealed that the underlying problem in integrating mathematical systems is a semantic one: all the reasoning systems make differing ontological assumptions that have to be reconciled to achieve a correct (i.e. meaning-preserving) integration. This integration problem is quite similar to the one at the knowledge level: if the knowledge ingrained in the system design could be explicitly described, then it would be possible to find applicable systems and deploy the necessary (syntactic) and (semantic) bridges automatically.

The approaches and solutions offered by the automated reasoning communities at that time were insular at best: They standardized character-level syntax standardizing on first-order logic [SSY94; HKW96], or explored bilateral system integrations overcoming deep ontological discrepancies between the systems [FH97].

At the same time, (ca 1998) the Computer Algebra Community was grappling with similar integration problems. The OpenMath standard that was emerging shad solved the web-scalability problem in representing mathematical formulae by adopting the emerging XML framework as a syntactical basis and providing structural markup with explicit context references as a syntax-independent representation approach. First attempts by the author to influence OpenMath standardization so that the format would allow mathematical knowledge representation (i.e. the statements and context level) were unsuccessful. The OpenMath community had intensively discussed similar issues under the heading of "content dictionary inheritance" and "conformance specification", and had decided that they were too controversial for standardization.

3.1.2 Design Principles

The start of the development of OMDoc as a content-based representation format for mathematical knowledge was triggered by an e-mail by Alan Bundy to the author in 1998, where he lamented the fact that one of the great hindrances of knowledge-based reasoning is the fact that formalizing mathematical knowledge is very time-consuming and that it is very hard for young researchers to gain recognition for formalization work. This led to the idea of developing a global repository of formalized mathematics, which would eventually allow peer-reviewed publication of formalized mathematical knowledge, thus generating academic recognition for formalization work and eventually lead to the much enlarged corpus of formalized mathematics that is necessary for knowledge-based formal mathematical reasoning. Young researchers would contribute formalizations of mathematical knowledge in the form of mathematical documents that would be both formal and thus machine-readable, as well as human-readable, so that humans could find and understand them[1].

This idea brought the final ingredient to the design principles: in a nutshell, the OMDoc format was to

1. be *Ontologically uncommitted* (like the OpenMath format), so that it could serve as a *integration format* for mathematical software systems.
2. provide a representation format for *mathematical documents* that combined *formal* and *informal* views of all the *mathematical knowledge* contained in them.
3. be based on *sound logic/representational principles* (as not to embarrass the author in front of his colleagues from automated reasoning)
4. be based on *structural/content markup* to guarantee both 1.) and 2.).

3.1.3 Development History

Version 1.0 of the OMDoc format was released on November 1^{st} 2000 to give users a stable interface to base their documents and systems on. It was adopted by various projects in automated deduction, algebraic specification, and computer-supported education. The experience from these projects uncovered a multitude of small deficiencies and extension possibilities of the format, that have been subsequently discussed in the OMDoc community.

OMDoc 1.1 was released on December 29^{th} 2001 as an attempt to roll the uncontroversial and non-disruptive part of the extensions and corrections into a consistent language format. The changes to version 1.0 were largely conservative, adding optional attributes or child elements. Nevertheless, some non-conservative changes were introduced, but only to less used parts of the format or in order to remedy design flaws and inconsistencies of version 1.0.

OMDoc 1.2 is the mature version in the OMDoc 1 series of specifications. It contains almost no large-scale changes to the document format, except that Content-MathML is now allowed as a representation for mathematical objects. But many of the representational features have been fine-tuned and brought up to date with the maturing XML technology (e.g. `ID` attributes now follow the XML ID specification [MVW05], and the Dublin Core elements follow the official syntax [DUB03a]). The main development is that the OMDoc specification, the DTD, and schema are split into a system of interdependent modules that support independent development of certain language aspects and simpler specification and deployment of sub-languages. Version

[1] Here the strong influence of the Mizar project under Andrzej Trybulec must be acknowledged, at that time, the project had already realized these two goals. They had even established the "Journal of Formalized Mathematics", where LaTeX articles were generated from the automatically verified Mizar source. However, the Mizar mathematical language [Urld] used a human-oriented syntax that defied outside parsing and web-integration, had a tightly integrated largely undocumented sort system, and made very strong ontological commitments.

1.2 of OMDoc freezes the development so that version 2 can be started off on the modules.

3.2 Three Levels of Markup

To achieve content and context markup for mathematical knowledge, OMDoc uses three levels of modeling corresponding to the concerns raised previously. We have visualized this architecture in Figure 3.1.

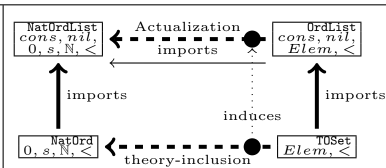
Level of Representation	OMDoc Example
Theory Level: Development Graph • Inheritance via symbol-mapping • Theory inclusion via proof-obligations • Local (one-step) vs. global links	
Statement Level: • Axiom, definition, theorem, proof, example,... • Structure explicit in statement forms and references	<definition for="plus" type="recursive"> <CMP><xhtml:p>Addition is defined by recursion on the second argument.</xhtml:p> </CMP> <FMP>$X + 0 = 0$</FMP> <FMP>$X + s(Y) = s(X + Y)$</FMP> </definition>
Object Level: OpenMath/MathML • Objects as logical formulae • Semantics by pointing to theory level	<OMA> <OMS cd="arith1" name="plus"/> <OMV name="X"/> <OMS cd="nat" name="zero"/> </OMA>

Fig. 3.1. OMDoc in a Nutshell (the Three Levels of Modeling)

Building on the discussion in Chapter 2 we distinguish three levels of representation in OMDoc

Mathematical Theories (see Section 2.1) At this level, OMDoc supplies original markup for clustering sets of statements into theories, and for specifying relations between theories by morphisms. By using this scheme, mathematical knowledge can be structured into reusable chunks. Theories also serve as the primary notion of context in OMDoc, they are the natural target for the context aspect of formula and statement markup.

Mathematical Statements (see Section 2.2) OMDoc provides original markup infrastructure for making the structure of mathematical statements explicit. Again, we have content and context markup aspects. For instance the definition in the right hand side of the second row of Figure 3.1 contains an informal description of the definition as a first child and a formal description in the two recursive equations in the second and third children supported by the `type` attribute, which states that this is a recursive definition. The context markup in this example is simple: it states that this piece of markup pertains to a symbol declaration for the symbol `plus` in the current theory (presumably the theory `arith1`).

Mathematical Formulae (see Section 2.3) At the level of mathematical formulae, OMDoc uses the established standards OpenMath [Bus+04] and Content-MathML [Aus+03a]. These provide content markup for the structure of mathematical formulae and context markup in the form of URI references in the symbol representations (see Chapter 13 for an introduction).

All levels are augmented by markup for various auxiliary information that is present in mathematical documents, e.g. notation declarations, exercises, experimental data, program code, etc.

3.3 Situating the OMDoc Format

The space of representation languages for mathematical knowledge reaches from the input languages of computer algebra systems (CAS) to presentation markup languages for mathematical vernacular like TEX/LATEX. We have organized some of the paradigmatic examples in a diagram mapping coverage (which kinds of mathematical knowledge can be expressed) against machine support (which services the respective software system can offer) in Figure 3.2.

On the left hand side we see CAS like Mathematica® [Wol02] or Maple™ [Cha+92] that are relatively restricted in the mathematical objects — they can deal with polynomials, group representations, differential equations only, but in this domain they can offer sophisticated services like equation solving, factorization, etc. More to the right we see systems like automated theorem provers, whose language — usually first-order logic — covers much more of mathematics, but that cannot perform computational services[2] like the CAS do.

In the lower right hand corner, we find languages like "mathematical vernacular", which is just the everyday mathematical language. Here coverage is essentially universal: we can use this language to write international treaties, math books, and love letters; but machine support is minimal, except for typesetting systems for mathematical formulae like TEX, or keyword search in the natural language part.

The distribution of the systems clusters around the diagonal stretching from low-coverage, high-support systems like CAS to wide-coverage, low-support natural language systems. This suggests that there is a trade-off

[2] Of course in principle, the systems could, since computation and theorem proving are inter-reducible, but in practice theorem provers get lost in the search spaces induced by computational tasks.

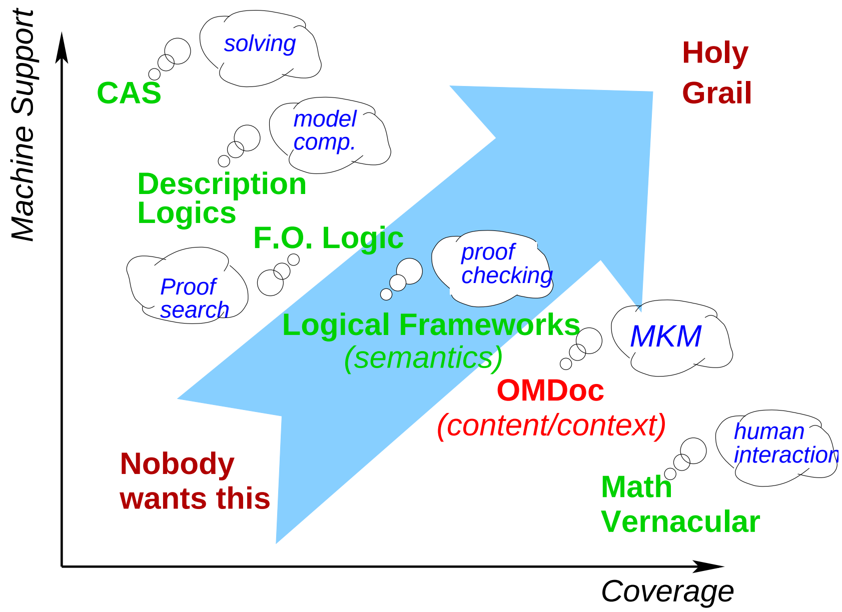


Fig. 3.2. Situating Content Markup: Math. Knowledge Management

between coverage and machine support. All of the representation languages occupy legitimate places in the space of representation languages, trying to find sweet-spots along this coverage/support trade-off. OMDoc tries to occupy the "content markup" position. To understand this position better, let us contrast it to the "semantic markup" position immediately to the left of and above it. This is an important distinction, since it marks the border between formal and informal mathematics.

We define a **semantic markup format** (aka **formal system**) as a representation system that has a way of specifying when a formula is a consequence of another. Many semantic markup formats express the consequence relation by means of a formal calculus, which allows the mechanization of proof checking or proof verification. It is a widely held belief in mathematics, that all mathematical knowledge can in principle be expressed in a formal system, and various systems have been proposed and applied to specific areas of mathematics. The advantage of having a well-defined consequence relation (and proof-checking) has to be paid for by committing to a particular logical system.

Content markup does not commit to a particular consequence relation, and concentrates on providing services based on the marked up structure of the content and the context. Consider for instance the logical formula in Listing 2.1, where the OpenMath representation does not specify the full consequence relation (or the formal system) for the formula. It does something less but still useful, which is what we could call *semantics by pointing*: The symbols used in the representation are identified by a pointer (the URI jointly specified in the `cd` and `name` attributes) to a defining document (in this case an OpenMath content dictionary). Note that URI equality is a sufficient condition for two symbols to be equal, but not a necessary condition: Two symbols can be semantically equal without pointing to the same document, e.g. if the two defining documents are semantically marked up and the definitions are semantic consequences of each other.

In this sense, content markup offers a more generic markup service (for all formal systems; we do not have to commit ourselves) at the cost of being less precise (we for instance miss out on some symbol equalities). Thus, content markup is placed to the lower right of semantic markup in Figure 3.2. Note however, that content markup can easily be turned into semantic markup by adding a consequence relation, e.g. by pointing to defining documents that are marked up semantically. Unlike OpenMath and Content-MathML, the OMDoc format straddles the content/semantics border by closing the loop and providing a content markup format for both formulae and the defining documents. In particular, *an* OMDoc *document is semantic if all the documents it references are*.

As a consequence, OMDoc can serve as a migration format from formal to informal mathematics (and thus from representations that for human consumption to such that can be supported by machines). A document collection can be marked for content and context structure, making the structures and context references explicit in a first pass. Note that this pass may involve creating additional documents or identifying existing documents that serve as targets for the context references so that the document collection is self-contained. In a second (and possible semi-automatic) step, we can turn this self-contained document collection into a formal representation (semantic markup) by committing on consequence relations and adding the necessary detail to the referenced documents.

3.4 The Future: An Active Web of (Mathematical) Knowledge

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

- Cut and paste on the level of computation (take the output from a web search engine and paste it into a computer algebra system),
- Automatically proof checking published proofs,
- Math explanation (e.g. specializing a proof to an example that simplifies the proof in this special case),

- Semantic search for mathematical concepts (rather than keywords),
- Data mining for representation theorems (are there unnoticed groups out there?),
- Classification: Given a concrete mathematical structure, is there a general theory for it?

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

This promise of activating a web of knowledge is not limited to mathematics: the task of transforming the current presentation-oriented world-wide web into a "Semantic Web" [BL98] has been identified as one of the main challenges by the world W3C. With the OMDoc format we pursue an alternative vision of a 'Semantic Web' for Mathematics. Like Tim Berners-Lee's vision we aim to make the Web (here mathematical knowledge) machine-understandable instead of merely machine-readable. However, instead of a top-down metadata-driven approach, which tries to approximate the content of documents by linking them to web ontologies (expressed in terminologic logics), we explore a bottom-up approach and focus on making explicit the intrinsic structure of the underlying scientific knowledge. A connection of documents to web ontologies is still possible, but a secondary effect.

The direct applications of OMDoc (apart from the general effect towards a Semantic Web) are not confined to mathematics proper either. The MathML working group in the W3C has led the way in many web technologies (presenting mathematics on the web taxes the current web technology to its limits); the endorsement of the MathML standard by the W3 Committee is an explicit testimony to this. We expect that the effort of creating an infrastructure for digital mathematical libraries will play a similar role, since mathematical knowledge is the most rigorous and condensed form of knowledge and will therefore pinpoint the problems and possibilities of the semantic web.

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

Part II

An OMDoc Primer

This part of the book provides an easily approachable description of the OMDoc format by way of paradigmatic examples of OMDoc documents. The primer should be used alongside the formal descriptions of the language contained in Part III.

The intended audience for the primer are users who only need a casual exposure to the format, or authors that have a specific text category in mind. The examples presented here also serve as specifications of "best practice", to give the readers an intuition for how to encode various kinds of mathematical knowledge.

Each chapter of the OMDoc primer deals with a different category of mathematical document and introduces new features of the OMDoc format in the context of concrete examples.

Chapter 4: Mathematical Textbooks and Articles

discusses the markup process for an informal but rigorous mathematical texts. We will use a fragment of Bourbaki's "Algebra" as an example. The development marks up the content in four steps, from the document structure to a full formalization of the content that could be used by automated theorem provers. The first page of Bourbaki's "Algebra" serves as an example of the treatment of a rigorous presentation of pure mathematics, as it can be found in textbooks and articles.

Chapter 5 OpenMath Content Dictionaries

transforms an OpenMath content dictionary into an OMDoc document. OpenMath content dictionaries are semi-formal documents that serve as references for mathematical symbols in OpenMath encoded formulae. As of OpenMath2, OMDoc is an admissible OpenMath content dictionary format. They are a good example for mathematical glossaries, and background references, both formal and informal.

Chapter 6 Structured and Parametrized Theories

shows the power of theory markup in OMDoc for theory reuse and modular specification. The example builds a theory of ordered lists of natural numbers from a generic theory of ordered lists and the theory of natural numbers which acts as a parameter in the actualization process.

Chapter 7 A Development Graph for Elementary Algebra

extends the range of theory-level structure by specifying the elementary algebraic hierarchy. The rich fabric of relations between these theories is made explicit in the form of theory morphisms, and put to use for proof reuse.

Chapter 8 Courseware and the Narrative/Content Distinction

covers markup for a fragment of a computer science course in the OMDoc format, dwelling on the difference between the narrative structure of the course and the background knowledge. Course materials like slides or writings on blackboards are usually much more informal than textbook presentations of mathematics. They also openly structure materials by didactic criteria and leave out important parts of the rigorous development, which the student is required to pick up from background materials like textbooks or the teacher's recitation.

Chapter 9 Communication with and between Mathematical Software Systems

uses an OMDoc fragment as content for communication protocols between mathematical software systems on the Internet. Since the communicating parties in this situation are machines, OMDoc fragments are embedded into other XML markup that serves as a protocol for the distribution layer.

Together these examples cover many of the mathematical documents involved in communicating mathematics. As the first two chapters build upon each other and introduce features of the OMDoc format, they should be read in succession. The remaining three chapters build on these, but are largely independent.

To keep the presentation of the examples readable, we will only present salient parts of the OMDoc representations in the discussion. The full text of the examples can be accessed at `https://svn.omdoc/repos/omdoc/doc/spec/examples/spec`.

4

Mathematical Textbooks and Articles

In this chapter we will work an example of a stepwise formalization of mathematical knowledge. This is the task of e.g. an editor of a mathematical textbook preparing it for web-based publication. We will use an informal, but rigorous text: a fragment of Bourbaki's Algebra [Bou74], which we show in Figure 4.1. We will mark it up in four stages, discussing the relevant OMDoc elements and the design decisions in the OMDoc format as we go along. Even though the text was actually written prior to the availability of the TeX/LaTeX system, we will take a LaTeX representation as the starting point of our markup experiment, since this is the prevalent source markup format in mathematics nowadays.

Section 4.1 discusses the minimal markup that is needed to turn an arbitrary document into a valid OMDoc document — albeit one, where the markup is worthless of course. It discusses the necessary XML infrastructure and adds some meta-data to be used e.g. for document retrieval or archiving purposes.

In Section 4.2 we mark up the top-level structure of the text and classify the paragraphs by their category as mathematical statements. This level of markup already allows us to annotate and extract some meta-data and would allow applications to slice the text into individual units, store it in databases like MBase (see Section **??**), or the In2Math knowledge base [Dah01; BB01], or assemble the text slices into individualized books e.g. covering only a subtopic of the original work. However, all of the text itself, still contains the LaTeX markup for formulae, which is readable only by experienced humans, and is fixed in notation. Based on the segmentation and meta-data, suitable systems like the ActiveMath system described in Section **??** can re-assemble the text in different orders.

In Section 4.3, we will map all mathematical objects in the text into OpenMath or Content-MathML objects. To do this, we have to decide which symbols we want to use for marking up the formulae, and how to structure the theories involved. This will not only give us the ability to generate specialized and user-adaptive notation for them (see Chapter **??**), but also to

1. LAWS OF COMPOSITION

DEFINITION 1. *Let E be a set. A mapping of $E \times E$ is called a law of composition on E. The value $f(x, y)$ of f for an ordered pair $(x, y) \in E \times E$ is called the composition of x and y under this law. A set with a law of composition is called a magma.*

The composition of x and y is usually denoted by writing x and y in a definite order and separating them by a characteristic symbol of the law in question (a symbol which it may be agreed to omit). Among the symbols most often used are $+$ and $\cdot$, the usual convention being to omit the latter if desired; with these symbols the composition of x and y is written respectively as $x + y$, $x.y$ or xy. A law denoted by the symbol $+$ is usually called *addition* (the composition $x + y$ being called the *sum* of x and y) and we say that it is *written additively*; a law denoted by the symbol . is usually called *multiplication* (the composition $x.y = xy$ being called the *product* for x and y) and we say that it is *written multiplicatively*.
In the general arguments of paragraphs 1 to 3 of this chapter we shall generally use the symbols $\top$ and $\perp$ to denote arbitrary laws of composition.
By an abuse of language, a mapping of a *subset* of $E \times E$ into E is sometimes called a law of composition *not everywhere defined* on E.

Examples. (1) The mappings $(X, Y) \mapsto X \cup Y$ and $(X, Y) \mapsto X \cap Y$ are laws of composition on the set of subsets of a set E.

(2) On the set $\mathbf{N}$ of natural numbers addition, multiplication, and exponentiation are laws of composition (the compositions of $x \in \mathbf{N}$ and $y \in \mathbf{N}$ under these laws being denoted respectively by $x + y$, xy, or $x.y$ and x^y) (*Set Theory*, III, §3, no. 4).

(3) Let E be a set; the mapping $(X, Y) \mapsto X \circ Y$ is a law of composition on the set of subsets of $E \times E$ (*Set Theory*, II, §3, no. 3, Definition 6); the mapping $(f, g) \mapsto f \circ g$ is a law of composition on the set of mappings from E into E (*Set Theory*, II, §5, no. 2).

Fig. 4.1. A fragment from Bourbaki's algebra [Bou74]

copy and paste them to symbolic math software systems. Furthermore, an assembly into texts can now be guided by the semantic theory structure, not only by the mathematical text categories or meta-data.

Finally, in Section 4.4 we will fully formalize the mathematical knowledge. This involves a transformation of the mathematical vernacular in the statements into some logical formalism. The main benefit of this is that we can verify the mathematical contents in theorem proving environments like NuPrL [Con+86], Hol [GM93], Mizar [Rud92] and OMEGA [Ben+97].

4.1 Minimal OMDoc Markup

It actually takes very little change to an existing document to make it a valid OMDoc document. We only need to wrap the text into the appropriate XML document tags. In Listing 4.1, we have done this and also added meta-data. Actually, since the `metadata` and the document type declaration are optional in OMDoc, just wrapping the original text with lines 1, 4, 7, 31, 32, and 36 to 38 is the simplest way to create an OMDoc document.

Listing 4.1. The outer part of the document

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <!DOCTYPE omdoc PUBLIC "-//OMDoc//DTD OMDoc Basic V1.3//EN"
3                  "http://omdoc.org/dtd/omdoc-basic.dtd" []>
4  
5  <omdoc xml:id="algebra1.omdoc" version="1.3" modules="@basic"
6         xmlns:dc="http://purl.org/dc/elements/1.1/"
7         xmlns:cc="http://creativecommons.org/ns"
8         xmlns="http://omdoc.org/ns">
9    <metadata>
10     <dc:title>Laws of Composition</dc:title>
11     <dc:creator role="trl">Michael Kohlhase</dc:creator>
12     <dc:date action="created">2002-01-03T07:03:00</dc:date>
13     <dc:date action="updated">2002-11-23T18:17:00</dc:date>
14     <dc:description>
15       A first migration step for a fragment of Bourbaki's Algebra
16     </dc:description>
17     <dc:source>
18       Nicolas Bourbaki, Algebra, Springer Verlag 1989, ISBN 0-387-19373-1
19     </dc:source>
20     <dc:type>Text</dc:type>
21     <dc:format>application/omdoc+xml</dc:format>
22     <dc:rights>Copyright (c) 2005 Michael Kohlhase</dc:rights>
23     <cc:license>
24       <cc:permissions reproduction="permitted" distribution="permitted"
25                       derivative_works="permitted"/>
26       <cc:prohibitions commercial_use="permitted"/>
27       <cc:requirements notice="required" copyleft="required" attribution="required"/>
28     </cc:license>
29   </metadata>
30 
31   <omtext xml:id="all">
32     <CMP xml:lang="en">
33       <xhtml:p>
34         {\sc Definition 1.} Let E be a set. A mapping E × E is called a law of
35         ...
36         mappings from E into E ({\emph{Set Theory}}, II, §5, no. 2).
37       </xhtml:p>
38     </CMP>
39   </omtext>
40 </omdoc>
```

We will now explain the general features of the OMDoc representation in detail by line numbers. The references point to the relevant sections in the OMDoc specification; details and normative rules for using the elements in questions can be found there.

We will now explain the general features of the OMDoc representation in detail by line numbers. The references point to the relevant sections in the

OMDoc specification; details and normative rules for using the elements in questions can be found there.

line	Description	ref.
1	This document is an XML 1.0 file that is encoded in the UTF-8 encoding.	
2,3	The parser is told to use a document type definition for validation. The string `omdoc` specifies the name of the root element, the identifier `PUBLIC` specifies that the DTD (we use the "OMDoc basic" DTD; see Subsection 22.3.1), which can be identified by the public identifier in the first string and looked up in an XML catalog or (if that fails) can be found at the URL specified in the second string. A DTD declaration is not strictly needed for an OMDoc document, but is recommended, since the DTD supplies default values for some attributes.	?? p. ??
4	In general, XML files can contain as much whitespace as they want between elements, here we have used it for structuring the document.	
5	Start tag of the root element of the document. It declares the version (OMDoc 1.3) via the `version`, and an identifier of the document using the `xml:id` attribute. The optional `modules` specifies the sub-language used in this document. This is used when no DTD is present (see Subsection 22.3.1).	11.1 p. 98
6,7	the namespace prefix declarations for the Dublin Core, Creative Commons, and OpenMath namespaces. They declare the prefixes `dc:`, `cc:`, and `om:`, and bind them to the specified URIs. We will need the OpenMath namespace only in the third markup step described in Section 4.3, but spurious namespace prefix declarations are not a problem in the XML world.	10 p. 89
8	the namespace declaration for the document; if not prefixed, all elements live in the OMDoc namespace.	10.2 p. 89
9–29	The metadata for the whole document in Dublin Core format	11.3 p. 100
10	The title of the document	12.2 p. 113
11	The document creator, here in the role of a translator	12.3 p. 116
12	The date and time of first creation of the document in ISO 8601 norm format.	12.2 p. 114
13	The date and time of the last update to the document in ISO 8601 norm format.	12.2 p. 114
14–16	A short description of the contents of the document	12.2 p. 114
17–19	Here we acknowledge that the OMDoc document is just a translation from an earlier work.	12.2 p. 115
20	The type of the document, this can be `Dataset` (un-ordered mathematical knowledge) or `Text` (arranged for human consumption).	12.2 p. 115

21	The format/MIME type [FB96] of the document, for OMDoc, this is `application/omdoc+xml`.	12.2 p. 115
22	The copyright resides with the creator of the OMDoc document	12.2 p. 115
23–28	The creator licenses the document to the world under certain conditions as specified in the Creative Commons license specified in this element.	12.4 p. 118
24,25	The `cc:permissions` element gives the world the permission to reproduce and distribute it freely. Furthermore the license grants the public the right to make derivative works under certain conditions.	12.4 p. 119
26	The `cc:prohibitions` can be used to prohibit certain uses of the document, but this one is unencumbered.	12.4 p. 119
27	The `cc:requirements` states conditions under which the license is granted. In our case the licensee is required to keep the copyright notice and license notices intact during distribution, to give credit to the copyright holder, and that any derivative works derived from this document must be licensed under the same terms as this document (the copyleft clause).	12.4 p. 119
31-37	The `omtext` element is used to mark up text fragments. Here, we have simply used a single `omtext` to classify the whole text in the fragment as unspecific “text”.	14.4 p. 145
32-36	The `CMP` element holds the actual text in a multilingual group. Its `xml:lang` specifies the language. If the document is used with a DTD or an XML schema (as we are) this attribute is redundant, since the default value given by the DTD or schema is `en`. More keywords in other languages can be given by adding more `CMP` elements.	14.1.1 p. 138
33–35	The text of the LaTeX fragment we are migrating. For simplicity we do not change the text, and leave that to later stages of the migration.	
38	The closing tag of the root `omdoc` element. There may not be text after this in the file.	11.1 p. 98

4.2 Marking up the text structure and statements

In the next step, we analyze and mark up the structure of the text of the further, and embed the paragraphs into markup for mathematical statements or text segments. Instead of lines 32–36 in Listing 4.1, we will now have the representation in Listing 4.2.

Listing 4.2. The segmented text

```
<omtext xml:id="magma.def" type="definition">
  <CMP>
    <xhtml:p>Let <legacy format="TeX">E</legacy> be a set . . . called a magma.</xhtml:p>
  </CMP>
</omtext>
```

```
<omtext xml:id="t1">
  <CMP>
    <xhtml:p>The composition of <legacy format="TeX">x</legacy> ...
10    multiplicatively .<xhtml:p>
  </CMP>
</omtext>
<omtext xml:id="t2">
  <CMP><xhtml:p>In the general ... composition.<xhtml:p></CMP>
15 </omtext>
<omtext xml:id="t3">
  <CMP><xhtml:p>By an abuse ... on <legacy format="TeX">E.</legacy></xhtml:p></CMP>
</omtext>

20 <omgroup xml:id="magma−ex" type="enumeration">
   <metadata><dc:title>Examples</dc:title></metadata>

   <omtext type="example" xml:id="e1.magma">
     <CMP>
25     <xhtml:p>
         The mappings <legacy format="TeX">(X, Y)</legacy>
        ... subsets of a set <legacy format="TeX">E</legacy>.
       </xhtml:p>
     </CMP>
30 </omtext>
   <omtext type="example" xml:id="e2.magma">
     <CMP>
       <xhtml:p>
        On the set <legacy format="TeX">N</legacy> ... III, §3, no. 4).
35     </xhtml:p>
     </CMP>
   </omtext>
   <omtext type="example" xml:id="e3.magma">
     <CMP>
40     <xhtml:p>Let <legacy format="TeX">E</legacy> be a set; ... II, §5, no. 2).</xhtml:p>
     </CMP>
   </omtext>
</omgroup>
```

In summary, we have sliced the text into `omtext` fragments and individually classified them by their mathematical role. The formulae inside have been encapsulated into `legacy` elements that specify their format for further processing. The higher-level structure has been captured in OMDoc grouping elements and the document as well as some of the slices have been annotated by metadata.

line	Description	ref.
1	The `omtext` element classifies the text fragment as a `definition`, other types for mathematical statements include `axiom`, `example`, `theorem`, and `lemma`. Note that the numbering of the original text is lost, but can be re-created in the text presentation process. The optional `xml:id` attribute specifies a document-unique identifier that can be used for reference later.	14.4 p. 145
2	A multilingual group of `CMP` elements that hold the text (in our case, there is only the English default). Here the TeX formulae have been marked up with `legacy` elements characterizing them as such. This might simplify a later automatic transformation to OpenMath or Content-MathML.	13.5 p. 134

4–13	We have classified every paragraph in the original as a separate `omtext` element, which does not carry a `type` since it does not fit any other mathematical category at the moment.	14.4 p. 145
15	The three examples in the original in Figure 4.1 are grouped into an enumeration. We use the OMDoc `omgroup` element for this. The optional attribute `xml:id` can be used for referencing later. We have chosen `enumeration` for the `type` attribute to specify the numbering of the examples in the original.	15.6 p. 166
16	We can use the `metadata` of the `omgroup` element to accommodate the title "Examples" in the original. We could enter more metadata at this level.	12.2 p. 113
18	The `type` attribute of this `omtext` element classifies this text fragment as an example.	14.4 p. 145

4.3 Marking up the Formulae

After we have marked up the top-level structure of the text to expose the content, the next step will be to mark up the formulae in the text to content mathematical form. Up to now, the formulae were still in TEX notation, which can be read by TEX/LATEX for presentation to the human user, but not used by symbolic mathematics software. For this purpose, we will re-represent the formulae as OpenMath objects or Content-MathML, making their functional structure explicit.

So let us start turning the TEX formulae in the text into OpenMath objects. Here we use the hypothetical `mbc.mathweb.org` as repository for theory collections.

Listing 4.3. The definition of a magma with OpenMath objects

```
<!DOCTYPE omdoc PUBLIC "-//OMDoc//DTD OMDoc CD V1.3//EN"
                      "http://omdoc.org/dtd/omdoc-cd.dtd"
          [<!ENTITY % om.prefixed "INCLUDE">]>

<theory xml:id="magmas">
  <imports from="background.omdoc#products"/>
  <imports from="http://mbc.mathweb.org/omstd/relation1.omdoc#relation1"/>

  <symbol name="magma">
    <metadata><dc:description>Magma</dc:description></metadata>
  </symbol>
  <symbol name="law_of_composition"/>

  <definition xml:id="magma.def" for="magma law_of_composition">
    <CMP>
     <xhtml:p>
      Let <om:OMOBJ><om:OMV name="E"/></om:OMOBJ> be a set. A mapping of
      <om:OMOBJ>
        <om:OMA><om:OMS cd="products" name="Cartesian-product"/>
          <om:OMV name="E"/><om:OMV name="E"/>
        </om:OMA>
      </om:OMOBJ> is called a
```

```
   <term cd=”magmas” name=”magma” role=”definiendum”>law of composition</term>
   on <om:OMOBJ><om:OMV name=”E”/></om:OMOBJ>. The value
   <om:OMOBJ>
     <om:OMA><om:OMV name=”f”/>
27     <om:OMV name=”x”/><om:OMV name=”y”/>
     </om:OMA>
   </om:OMOBJ>
   of <om:OMOBJ><om:OMV name=”f”/></om:OMOBJ> for an ordered pair
   <om:OMOBJ>
32   <om:OMA><om:OMS cd=”sets” name=”in”/>
       <om:OMA><om:OMS cd=”products” name=”pair”/>
         <om:OMV name=”x”/><om:OMV name=”y”/>
       </om:OMA>
       <om:OMA><om:OMS cd=”products” name=”Cartesian−product”/>
37       <om:OMV name=”E”/><om:OMV name=”E”/>
       </om:OMA>
     </om:OMA>
   </om:OMOBJ> is called the
   <term cd=”magmas” name=”law_of_composition”
42                  role=”definiendum−applied”>composition</term>
    of <om:OMOBJ><om:OMV name=”x”/></om:OMOBJ> and
   <om:OMOBJ><om:OMV name=”y”/></om:OMOBJ> under this law.
   A set with a law of composition is called a
   <term cd=”magmas” name=”magma” role=”definiendum”>magma</term>.
47 </xhtml:p>
  </CMP>
 </definition>
...
</theory>
52 ...
```

Of course all the other mathematical statements in the documents have to be treated in the same way.

line	Description	ref.
1–4	The `omdoc-basic` document type definition is no longer sufficient for our purposes, since we introduce new symbols that can be used in other documents. The DTD for OMDoc content dictionaries (see Chapter 5), which allows this. Correspondingly, we would specify the value `cd` for the attribute `module`. The part in line 4 is the internal subset of the DTD, which sets a parameter entity for the modularized DTD to instruct it to accept OpenMath elements in their namespace prefixed form. Of course a suitable namespace prefix declaration is needed as well.	22.3.2 p. 218
5	The start tag of a theory. We need this, since symbols and definitions can only appear inside `theory` elements.	15.6 p. 165

6,7	We need to import the theory `products` to be able to use symbols from it in the definition below. The value of the `from` is a relative URI reference to a `theory` element much like the one in line 5. The other `imports` element imports the theory `relation1` from the OpenMath standard content dictionaries[1]. Note that we do not need to import the theory `sets` here, since this is already imported by the theory `products`.	15.6.1 p. 166
9–11	A symbol declaration: For every definition, OMDoc requires the declaration of one or more `symbol` elements for the concept that is to be defined. The `name` attribute is used to identify it. The `dc:description` element allows to supply a multilingual (via the `xml:lang` attribute) group of keywords for the declared symbol	15.2.1 p. 152
12	Upon closer inspection it turns out that the definition in Listing 4.3 actually defines three concepts: “law of composition”, “composition”, and “magma”. Note that “composition” is just another name for the value under the law of composition, therefore we do not need to declare a symbol for this. Thus we only declare one for “law of composition”.	15.2.1 p. 152
14	A definition: the `definition` element carries a `name` attribute for reference within the theory. We need to reference the two symbols defined here in the `for` attribute of the `definition` element; it takes a whitespace-separated list of `name` attributes of `symbol` elements in the same theory as values.	15.2.4 p.155
16	We use an OpenMath object for the set E. It is an `om:OMOBJ` element with an `om:OMV` daughter, whose `name` attribute specifies the object to be a variable with name E. We have chosen to represent the set E as a variable instead of a constant (via an `om:OMS` element) in the theory, since it seems to be local to the definition. We will discuss this further in the next section, where we talk about formalization.	13.1.1 p. 122
17–21	This `om:OMOBJ` represents the Cartesian product $E\times E$ of the set E with itself. It is an application (via an `om:OMA` element) of the symbol for the binary Cartesian product relation to E.	13.1.1 p. 122
18	The symbol for the Cartesian product constructor is represented as an `om:OMS` element. The `cd` attribute specifies the theory that defines the symbol, and the `name` points to the `symbol` element in it that declares this symbol. The value of the `cd` attribute is a theory identifier. Note that this theory has to be imported into the current theory, to be legally used.	13.1.1 p. 122
22	We use the `term` element to characterize the defined terms in the text of the definition. Its `role` attribute can used to mark the text fragment as a `definiens`, i.e. a concept that is under definition.	14.2.6 p. 143

[1] The originals are available at `http://www.openmath.org/cd`; see Chapter 5 for a discussion of the differences of the original OpenMath format and the OMDoc format used here.

24–28	This object stands for $f(x, y)$	
30–39	This object represents $(x, y) \in E \times E$. Note that we make use of the symbol for the elementhood relation from the OpenMath core content dictionary `set1` and of the pairconstructor from the theory of products from the Bourbaki collection there.	

The rest of the representation in Listing 4.3 is analogous. Thus we have treated the first definition in Figure 4.1. The next two paragraphs contain notation conventions that help the human reader to understand the text. They are annotated as `omtext` elements. The third paragraph is really a definition (even if the wording is a bit bashful), so we mark it up as one in the style of Listing 4.3 above.

Finally, we come to the examples at the end of our fragment. In the markup shown in Listing 4.4 we have decided to construct a new theory for these examples since the examples use concepts and symbols that are independent of the theory of magmas. Otherwise, we would have to add the `imports` element to the theory in Listing 4.3, which would have mis-represented the actual dependencies. Note that the new theory has to import the theory `magmas` together with the theories from which examples are taken, so their symbols can be used in the examples.

Listing 4.4. Examples for magmas with OpenMath objects

```
<theory xml:id="magmas-examples">
  <metadata><dc:title>Examples</dc:title></metadata>

   <imports from="http://mbc.mathweb.org/omstd/fns1.omdoc##fns1"/>
   <imports from="background.omdoc#nat"/>
   <imports from="background.omdoc#functions"/>
   <imports from="#magmas"/>

  <omgroup xml:id="magma-ex" type="enumeration">
    <metadata><dc:title>Examples</dc:title></metadata>

    <example xml:id="e1.magma" for="#law_of_composition" type="for">
      <CMP>The mappings
        <xhtml:p>
        <om:OMOBJ>
          <om:OMBIND><om:OMS cd="fns1" name="lambda"/>
            <om:OMBVAR>
              <om:OMV name="X"/><om:OMV name="Y"/>
            </om:OMBVAR>
            <om:OMA><om:OMS cd="functions" name="pattern-defined"/>
              <om:OMA><om:OMS cd="products" name="pair"/>
                <om:OMV name="X"/>
                <om:OMV name="Y"/>
              </om:OMA>
              <om:OMA><om:OMS cd="sets" name="union"/>
                <om:OMV name="X"/>
                <om:OMV name="Y"/>
              </om:OMA>
            </om:OMA>
          </om:OMBIND>
        </om:OMOBJ> and
        <om:OMOBJ>
```

```
33          <om:OMBIND><om:OMS cd="fns1" name="lambda"/>
              <om:OMBVAR>
                <om:OMV name="X"/><om:OMV name="Y"/>
              </om:OMBVAR>
              <om:OMA><om:OMS cd="functions" name="pattern−defined"/>
38              <om:OMA><om:OMS cd="products" name="pair"/>
                  <om:OMV name="X"/>
                  <om:OMV name="Y"/>
                </om:OMA>
                <om:OMA><om:OMS cd="sets" name="intersection"/>
43                <om:OMV name="X"/>
                  <om:OMV name="Y"/>
                </om:OMA>
              </om:OMA>
            </om:OMBIND>
48        </om:OMOBJ>
          are <term cd="magmas" name="law_of_composition>laws of composition</term>
          on the set of subsets of a set
          <om:OMOBJ><om:OMS cd="magmas" name="E"/></om:OMOBJ>.
          </xhtml:p>
53      </CMP>
      </example>

      <example xml:id="e2.magma" for="#law_of_composition" type="for">
        <CMP>
58       <xhtml:p>
          On the set <om:OMOBJ><om:OMS cd="nat" name="Nat"/></om:OMOBJ>
          of <term cd="nats" name="nats">natural numbers</term>,
          <term cd="nats" name="plus">addition</term>,
          <term cd="nats" name="times">multiplication</term>, and
63        <term cd="nats" name="power">exponentiation</term> are ...
         </xhtml:p>
        </CMP>
      </example>
    </omgroup>
68 </theory>
```

The `example` element in line 13 is used for mathematical examples of a special form in OMDoc: objects that have or fail to have a specific property. In our case, the two given mappings have the property of being a law of composition. This structural property is made explicit by the `for` attribute that points to the concept that these examples illustrate, in this case, the symbol `law_of_composition`. The `type` attribute has the values `for` and `against`. In our case `for` applies, `against` would for counterexamples. The content of an `example` is a multilingual `CMP` group. For examples of other kinds — e.g. usage examples, OMDoc does not supply specific markup, so we have to fall back to using an `omtext` element with type `example` as above.

In our text fragment, where the examples are at the end of the section that deals with magmas, creating an independent theory for the examples (or even multiple theories, if examples from different fields are involved) seems appropriate. In other cases, where examples are integrated into the text, we can equivalently embed theories into other theories. Then we would have the following structure:

Listing 4.5. Examples embedded into a theory

```
<theory xml:id="magmas">
  <imports xml:id="imp3" from="background.omdoc#products"/>
  <imports from="http://mbc.mathweb.org/omstd/relation1.omdoc#relation1"/>
```

```
  ...
  <theory xml:id="magmas−examples"
    <imports xml:id="imp4"
       from="http://omdoc.org/examples/omstd/fns1.omdoc#fns1"/>
    <imports xml:id="imp5" from="background.omdoc#nat"/>
    <imports xml:id="imp6" from="background.omdoc#functions"/>
    ...
  </theory>
  ...
</theory>
```

Note that the embedded theory (`magmas-examples`) has access to all the symbols in the embedding theory (`magmas`), so it does not have to import it. However, the symbols imported into the embedded theory are only visible in it, and do not get imported into the embedding theory.

4.4 Full Formalization

The final step in the migration of the text fragment involves a transformation of the mathematical vernacular in the statements into some logical formalism. The main benefit of this is that we can verify the mathematical contents in theorem proving environments. We will start out by dividing the first definition into two parts. The first one defines the symbol `law_of_composition` (see Listing 4.6), and the second one `magma` (see Listing 4.7).

Listing 4.6. The formal definition of a law of composition

```
<symbol name="law_of_composition">
  <metadata><dc:description>A law of composition on a set.</dc:description></metadata>
</symbol>
<definition xml:id="magma.def" for="law_of_composition" type="simple">
  <CMP>
    <xhtml:p>
      Let <om:OMOBJ><om:OMV name="E"/></om:OMOBJ> be a set. A mapping of
     <om:OMOBJ><om:OMR href="#comp.1"/></om:OMOBJ>
     is called a <term cd="magmas" name="law_of_composition"
                          role="definiens">law of composition</term>
     on <om:OMOBJ><om:OMV name="E"/></om:OMOBJ>.
   </xhtml:p>
  </CMP>
  <om:OMOBJ>
    <om:OMBIND>
      <om:OMS cd="fns1" name="lambda"/>
      <om:OMBVAR>
        <om:OMV name="E"/><om:OMV name="F"/>
      </om:OMBVAR>
      <om:OMA><om:OMS cd="pl0" name="and"/>
       <om:OMA><om:OMS cd="sets" name="set"/>
          <om:OMV name="E"/>
       </om:OMA>
       <om:OMA>
         <om:OMS cd="functions" name="function"/>
         <om:OMA id="comp.1">
           <om:OMS cd="products" name="Cartesian−product"/>
           <om:OMV name="E"/>
           <om:OMV name="E"/>
         </om:OMA>
         <om:OMV name="E"/>
       </om:OMA>
```

```
    </om:OMA>
  </om:OMBIND>
 </om:OMOBJ>
</definition>
```

The main difference of this definition to the one in the section above is the `om:OMOBJ` element, which now accompanies the `CMP` element. It contains a formal definition of the property of being a law of composition in the form of a λ-term $\lambda E, F.set(E) \wedge F : E \times E \rightarrow E$[2]. The value `simple` of the `type` attribute in the `definition` element signifies that the content of the `om:OMOBJ` element can be substituted for the symbol `law_of_composition`, wherever it occurs. So if we have `law_of_composition`(A, B) somewhere this can be reduced to $(\lambda E, F.set(E) \wedge F : E \times E \rightarrow E)(A, B)$ which in turn reduces[3] to $set(A) \wedge B : A \times A \rightarrow A$ or in other words `law_of_composition`(A, B) is true, iff A is a set and B is a function from $A \times A$ to A. This definition is directly used in the second formal definition, which we depict in Listing 4.7.

Listing 4.7. The formal definition of a magma

```
<definition xml:id="magma.def" for="magma" type="implicit">
 <CMP>
   <xhtml:p>
     A set with a law of composition is called a
     <term cd="magmas" name="magma" role="definiendum">magma</term>.
   </xhtml:p>
 </CMP>
 <FMP>
   <om:OMOBJ>
     <om:OMBIND><om:OMS cd="pl1" name="forall"/>
       <om:OMBVAR><om:OMV name="M"/></om:OMBVAR>
       <om:OMA><om:OMS cd="pl0" name="iff"/>
         <om:OMA><om:OMS cd="magmas" name="magma"/>
           <om:OMV name="M"/>
         </om:OMA>
         <om:OMBIND>
           <om:OMS cd="pl1" name="exists"/>
           <om:OMBVAR>
             <om:OMV name="E"/><om:OMV name="C"/>
           </om:OMBVAR>
           <om:OMA><om:OMS cd="pl0" name="and"/>
             <om:OMA><om:OMS cd="relation1" name="eq"/>
               <om:OMV name="M"/>
               <om:OMA><om:OMS cd="products" name="Cartesian-product"/>
                 <om:OMV name="E"/>
                 <om:OMV name="C"/>
               </om:OMA>
             </om:OMA>
             <om:OMA><om:OMS cd="magmas" name="law_of_composition"/>
               <om:OMV name="E"/>
```

[2] We actually need to import the theories `pl1` for first-order logic (it imports the theory `pl0`) to legally use the logical symbols here. Since we did not show the theory element, we assume it to contain the relevant `imports` elements.

[3] We use the λ-calculus as a formalization framework here: If we apply a λ-term of the form $\lambda X.A$ to an argument B, then the result is obtained by binding all the formal parameters X to the actual parameter B, i.e. the result is the value of A, where all the occurrences of X have been replaced by B. See [Bar80; And02] for an introduction.

```
            <om:OMV name=”F”/>
          </om:OMA>
        </om:OMA>
      </om:OMBIND>
    </om:OMA>
   </om:OMBIND>
  </om:OMOBJ>
 </FMP>
</definition>
```

Here, the `type` attribute on the `definition` element has the value `implicit`, which signifies that the content of the `FMP` element should be understood as a logical formula that is made true by exactly one object: the property of being a magma. This formula can be written as

$$\forall M.magma(M) \Leftrightarrow \exists E, F.M = (E, F) \wedge law\_of\_composition(E, F)$$

in other words: M is a magma, iff it is a pair (E, F), where F is a law of composition over E.

Finally, the examples get a formal part as well. This mainly consists of formally representing the object that serves as the example, and making the way it does explicit. The first is done simply by adding the object to the example as a sibling node to the CMP. Note that we are making use of the OpenMath reference mechanism here that allows to copy subformulae by linking them with an `om:OMR` element that stands for a copy of the object pointed to by the `href` attribute (see Section 13.1), which makes this very simple. Also note that we had to split the example into two, since OMDoc only allows one example per `example` element. However, the `example` contains two `om:OMOBJ` elements, since the property of being a law of composition is binary.

The way this object is an example is made explicit by adding an assertion that makes the claim of the example formal (in our case that for every set E, the function $(X, Y) \mapsto X \cup Y$ is a law of composition on the set of subsets of E). The assertion is referenced by the `assertion` attribute in the `example` element.

Listing 4.8. A formalized magma example

```
<example xml:id=”e11.magma” for=”#law_of_composition”
         type=”for” assertion=”e11.magma.ass”>
 <CMP>
  <xhtml:p>
   The mapping <om:OMOBJ><om:OMR href=”#e11.magma.1”/></om:OMOBJ> is
   a law of composition on the set of subsets of a set
   <om:OMOBJ><om:OMS cd=”magmas” name=”E”/></om:OMOBJ>.
  </xhtml:p>
 </CMP>
 <om:OMOBJ>
  <om:OMA id=”e11.magma.2”><om:OMS cd=”sets” name=”subset”/>
   <om:OMV name=”E”/>
  </om:OMA>
 </om:OMOBJ>
 <om:OMOBJ>
  <om:OMBIND id=”e11.magma.1”>
   <om:OMS cd=”fns1” name=”lambda”/>
```

```
      <om:OMBVAR><om:OMV name="X"/><om:OMV name="Y"/></om:OMBVAR>
      <om:OMA>
        <om:OMS cd="functions" name="pattern-defined"/>
        <om:OMA><om:OMS cd="products" name="pair"/>
          <om:OMV name="X"/>
          <om:OMV name="Y"/>
        </om:OMA>
        <om:OMA><om:OMS cd="sets" name="union"/>
          <om:OMV name="X"/>
          <om:OMV name="Y"/>
        </om:OMA>
      </om:OMA>
    </om:OMBIND>
  </om:OMOBJ>
</example>

<assertion xml:id="e11.magma.ass">
  <FMP>
    <om:OMOBJ>
      <om:OMBIND>
        <om:OMS cd="pl1" name="forall"/>
        <om:OMBVAR><om:OMV name="E"/></om:OMBVAR>
        <om:OMA>
          <om:OMS cd="magmas" name="law_of_composition"/>
          <om:OMR href="#e11.magma.2"/>
          <om:OMR href="#e11.magma.1"/>
        </om:OMA>
      </om:OMBIND>
    </om:OMOBJ>
  </FMP>
</assertion>
```

n

5

OpenMath Content Dictionaries

Content Dictionaries are structured documents used by the OpenMath standard [Bus+04] to codify knowledge about mathematical symbols and concepts used in the representation of mathematical formulae. They differ from the mathematical documents discussed in the last chapter in that they are less geared towards introduction of a particular domain, but act as a reference/-glossary document for implementing and specifying mathematical software systems. Content Dictionaries are important for the OMDoc format, since the OMDoc architecture, and in particular the integration of OpenMath builds on the equivalence of OpenMath content dictionaries and OMDoc theories.

Concretely, we will look at the content dictionary `arith1.ocd` which defines the OpenMath symbols `abs`, `divide`, `gcd`, `lcm`, `minus`, `plus`, `power`, `product`, `root`, `sum`, `times`, `unary_minus` (see [OMCD] for the original). We will discuss the transformation of the parts listed below into OMDoc and see from this process that the OpenMath content dictionary format is (isomorphic to) a subset of the OMDoc format. In fact, the OpenMath2 standard only presents the content dictionary format used here as one of many encodings and specifies abstract conditions on content dictionaries that the OMDoc encoding below also meets. Thus OMDoc is a valid content dictionary encoding.

Listing 5.1. Part of the OpenMath content dictionary `arith1.ocd`

```
<CD>
  <CDName> arith1 </CDName>
  <CDURL> http://www.openmath.org/cd/arith1.ocd </CDURL>
  <CDReviewDate> 2003-04-01 </CDReviewDate>
  <CDStatus> official </CDStatus>
  <CDDate> 2001-03-12 </CDDate>
  <CDVersion> 2 </CDVersion>
  <CDRevision> 0 </CDRevision>
  <dc:description>
    This CD defines symbols for common arithmetic functions.
  </dc:description>

  <CDDefinition>
```

```
14  <Name> lcm </Name>
15  <Description>
16    The symbol to represent the n−ary function to return the least common
17    multiple of its arguments.
18  </Description>
19
20  <CMP> lcm(a,b) = a∗b/gcd(a,b) </CMP>
21  <FMP>... </FMP>
22
23  <CMP>
24    for all integers a,b |
25    There does not exist a c>0 such that c/a is an Integer and c/b is an
26    Integer and lcm(a,b) > c.
27  </CMP>
28  <FMP>...</FMP>
29  ...
30  </CD>
```

Generally, OpenMath content dictionaries are represented as mathematical theories in OMDoc. These act as containers for sets of symbol declarations and knowledge about them, and are marked by `theory` elements. The result of the transformation of the content dictionary in Listing 5.1 is the OMDoc document in Listing 5.2.

The first 25 lines in Listing 5.1 contain administrative information and metadata of the content dictionary, which is mostly incorporated into the metadata of the `theory` element. The translation adds further metadata to the `omdoc` element that were left implicit in the original, or are external to the document itself. These data comprise information about the translation process, the creator, and the terms of usage, and the source, from which this document is derived (the content of the `omcd:CDURL` element is recycled in Dublin Core metadata element `dc:source` in line 12.

The remaining administrative data is specific to the content dictionary per se, and therefore belongs to the `theory` element. In particular, the `omcd:CDName` goes to the `xml:id` attribute on the `theory` element in line 36. The `dc:description` element is directly used in the `metadata` in line 38. The remaining information is encapsulated into the `cd*` attributes.

Note that we have used the OMDoc sub-language "OMDoc Content Dictionaries" described in Subsection 22.3.2 since it suffices in this case, this is indicated by the `modules` attribute on the `omdoc` element.

Listing 5.2. The OpenMath content dictionary `arith1` in OMDoc form

```
1   <?xml version="1.0" encoding="utf−8"?>
2   <omdoc xml:id="arith1.omdoc" modules="@cd"
3          xmlns:dc="http://purl.org/dc/elements/1.1/">
4
5   <metadata>
6     <dc:title>The OpenMath Content Dictionary arith1.ocd in OMDoc Form</dc:title>
7     <dc:creator role="trl">Michael Kohlhase</dc:creator>
8     <dc:creator role="ant">The OpenMath Society</dc:creator>
9     <dc:date action="updated"> 2004−01−17T09:04:03Z </dc:date>
10    <dc:source>
11      Derived from the OpenMath CD http://www.openmath.org/cd/arith1.ocd.
12    </dc:source>
13    <dc:type>Text</dc:type>
14    <dc:format>application/omdoc+xml</dc:format>
```

```
  <dc:rights>Copyright (c) 2000 Michael Kohlhase;
    This OMDoc content dictionary is released under the OpenMath license:
    http://www.openmath.org/cdfiles/license.html
  </dc:rights>
</metadata>

<theory xml:id="arith1"
        cdstatus="official" cdreviewdate="2003-04-01" cdversion="2" cdrevision="0">
  <metadata>
    <dc:title>Common Arithmetic Functions</dc:title>
    <dc:description>This CD defines symbols for common arithmetic functions.</dc:description>
    <dc:date action="updated"> 2001-03-12 </dc:date>
  </metadata>
  <imports from="#sts"/>

  <symbol name="lcm">
    <metadata>
      <dc:description>The symbol to represent the n-ary function to return the least common
        multiple of its arguments.
      </dc:description>
      <dc:description xml:lang="de">
        Das Symbol für das kleinste gemeinsame Vielfache (als n-äre Funktion).
      </dc:description>
      <dc:subject>lcm, least common mean</dc:subject>
      <dc:subject xml:lang="de">kgV, kleinstes gemeinsames Vielfaches</dc:subject>
    </metadata>
    <type system="sts">
      <OMOBJ>
        <OMA><OMS name="mapsto" cd="sts"/>
          <OMA><OMS name="nassoc" cd="sts"/><OMV name="SemiGroup"/></OMA>
          <OMV name="SemiGroup"/>
        </OMA>
      </OMOBJ>
    </type>
  </symbol>

  <presentation xml:id="pr_lcm" for="#lcm">
    <use format="default">lcm</use>
    <use format="default" xml:lang="de">kgV</use>
    <use format="cmml" element="lcm"/>
  </presentation>

  <definition xml:id="lcm-def" for="lcm" type="pattern">
    <CMP><xhtml:p>We define <OMOBJ><OMR href="#lcm-def.O"/></OMOBJ>
      as <OMOBJ><OMR href="#lcm-def.1"/></OMOBJ><xhtml:p></CMP>
    <CMP xml:lang="de">
      <xhtml:p>Wir definieren <OMOBJ><OMR href="#lcm-def.O"/></OMOBJ>
      als <OMOBJ><OMR href="#lcm-def.1"/></OMOBJ><xhtml:p></CMP>
    <requation>
      <OMOBJ>
        <OMA id="lcm-def.O">
          <OMS cd="arith1" name="lcm"/>
          <OMV name="a"/><OMV name="b"/>
        </OMA>
      </OMOBJ>
      <OMOBJ>
        <OMA id="lcm-def.1">
          <OMS cd="arith1" name="divide"/>
          <OMA><OMS cd="arith1" name="times"/>
            <OMV name="a"/>
            <OMV name="b"/>
          </OMA>
          <OMA><OMS cd="arith1" name="gcd"/>
            <OMV name="a"/>
            <OMV name="b"/>
          </OMA>
        </OMA>
```

```
      </OMOBJ>
    </requation>
  </definition>

  <theory>
    <imports from="#relation1"/>
    <imports from="#quant1"/>
    <imports from="#logic1"/>

    <assertion xml:id="lcm-prop-3" type="lemma">
      <CMP><xhtml:p>For all integers <OMOBJ><OMV name="a"/></OMOBJ>,
        <OMOBJ><OMV name="b"/></OMOBJ> there is no
        <OMOBJ><OMR href="#lcm-prop-3.1"/></OMOBJ> such that
        <OMOBJ><OMR href="#lcm-prop-3.2"/></OMOBJ> and
        <OMOBJ><OMR href="#lcm-prop-3.3"/></OMOBJ> and
        <OMOBJ><OMR href="#lcm-prop-3.4"/></OMOBJ>.</xhtml:p>
      </CMP>
      <CMP xml:lang="de"><xhtml:p>Für alle ganzen Zahlen
        <OMOBJ><OMV name="a"/></OMOBJ>,
        <OMOBJ><OMV name="b"/></OMOBJ>
        gibt es kein <OMOBJ><OMR href="#lcm-prop-3.1"/></OMOBJ> mit
        <OMOBJ><OMR href="#lcm-prop-3.2"/></OMOBJ> und
        <OMOBJ><OMR href="#lcm-prop-3.3"/></OMOBJ> und
        <OMOBJ><OMR href="#lcm-prop-3.4"/></OMOBJ>.</xhtml:p>
      </CMP>
      <FMP>
        <OMOBJ><OMBIND><OMS cd="quant1" name="forall"/>
            <OMBVAR><OMV name="a"/><OMV name="b"/></OMBVAR>
            <OMA><OMS cd="logic1" name="implies"/>
              <OMA>...</OMA>
              <OMA><OMS cd="logic1" name="not"/>
                <OMBIND><OMS cd="quant1" name="exists"/>
                  <OMBVAR><OMV name="c"/></OMBVAR>
                  <OMA><OMS cd="logic1" name="and"/>
                    <OMA id="lcm-prop-3.1">...</OMA>
                    <OMA id="lcm-prop-3.2">...</OMA>
                    <OMA id="lcm-prop-3.3">...</OMA>
                    <OMA id="lcm-prop-3.4">...</OMA>
                  </OMA>
                </OMBIND>
              </OMA>
            </OMA>
          </OMBIND>
        </OMOBJ>
      </FMP>
    </assertion>
    ...
  </theory>
  ...
</theory>
```

One important difference between the original and the OMDoc version of the OpenMath content dictionary is that the latter is intended for machine manipulation, and we can transform it into other formats. For instance, the human-oriented presentation of the OMDoc version might look something like the following[1]:

[1] These presentation was produced by the style sheets discussed in Section **??**.

The OpenMath Content Dictionary arith1.ocd in OMDoc Form
Michael Kohlhase, The OpenMath Society
January 17. 2004
This CD defines symbols for common arithmetic functions.

Concept 1. `lcm` (lcm, least common mean)
Type (sts): $SemiGroup^* \to SemiGroup$
The symbol to represent the n-ary function to return the least common multiple of its arguments.

Definition 2.(lcm-def)
We define $lcm(a,b)$ as $\frac{a \cdot b}{gcd(a,b)}$

Lemma 3. *For all integers a, b there is no $c > 0$ such that $(a|c)$ and $(b|c)$ and $c < lcm(a,b)$.*

Fig. 5.1. A human-oriented presentation of the OMDoc CD

The OpenMath Content Dictionary arith1.ocd in OMDoc form
Michael Kohlhase, The OpenMath Society
17. Januar 2004
This CD defines symbols for common arithmetic functions.

Konzept 1. `lcm` (kgV, kleinstes gemeinsames Vielfaches)
Typ (sts): $SemiGroup^* \to SemiGroup$
Das Symbol für das kleinste gemeinsame Vielfache (als n-äre Funktion).

Definition 2.(lcm-def)
Wir definieren $kgV(a,b)$ als $\frac{a \cdot b}{ggT(a,b)}$

Lemma 3. *Für alle ganzen Zahlen a, b gibt es kein $c > 0$ mit $(a|c)$ und $(b|c)$ und $c < kgV(a,b)$.*

Fig. 5.2. A human-oriented presentation in German

6

Structured and Parametrized Theories

In Chapter 5 we have seen a simple use of theories in OpenMath content dictionaries. There, theories have been used to reference OpenMath symbols and to govern their visibility. In this chapter we will cover an extended example showing the structured definition of multiple mathematical theories, modularizing and re-using parts of specifications and theories. Concretely, we will consider a structured specification of lists of natural numbers. This example has been used as a paradigmatic example for many specification formats ranging from Casl (Common Abstract Specification Language [Mos04]) standard to the Pvs theorem prover [ORS92], since it uses most language elements without becoming too unwieldy to present.

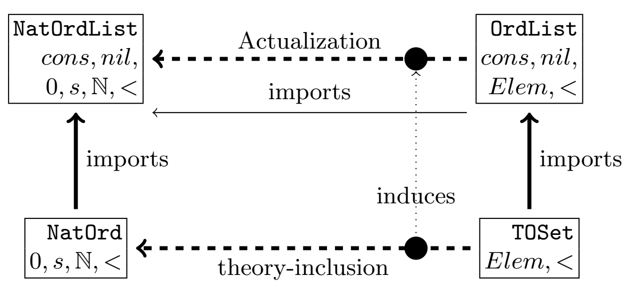


Fig. 6.1. A Structured Specification of Lists (of Natural Numbers)

In this example, we specify a theory `OrdList` of lists that is generic in the elements (which is assumed to be a totally ordered set, since we want to talk about ordered lists). Then we will instantiate `OrdList` by applying it to the theory `NatOrd` of natural numbers to obtain the intended theory `NatOrdList` of lists of natural numbers. The advantage of this approach is that we can re-use the generic theory `OrdList` to apply it to other element theories like that of “characters” to obtain a theory of lists of characters. In algebraic specification languages, we speak of **parametric theories**. Here, the theory `OrdList` has a formal parameter (the theory `TOSet`) that can be instantiated later with concrete values to get a **theory instance** (in our example the theory `NatOrdList`). We call this process theory **actualization**.

We begin the extended example with the theories in the lower half of Figure 6.1. The first is a (mock up of a) theory of totally ordered sets. Then we build up the theory of natural numbers as an **abstract data type** (see Chapter 16 for an introduction to abstract data types in OMDoc and a more elaborate definition of $\mathbb{N}$). The `sortdef` element posits that the set of natural numbers is given as the **sort** `NatOrd`, with the constructors `zero` and `succ`. Intuitively, a sort represents an inductively defined set, i.e. it contains exactly those objects that can be represented by the constructors only, for instance the number three is represented as $s(s(s(0)))$, where s stands for the successor function (given as the constructor `succ`) and 0 for the number zero (represented by the constructor `zero`). Note that the theory `nat` does not have any explicitly represented axioms. They are implicitly given by the abstract data type structure, in our case, they correspond to the five Peano Axioms (see Figure 15.1). Finally, the `argument` elements also introduce one partial inverse to the constructor functions per argument; in our case the predecessor function.

```
<theory xml:id="TOSet">
  <symbol name="set"/>
  <symbol name="ord"/>
  <axiom xml:id="toset"><CMP><xhtml:p>ord is a total order on set.</xhtml:p></CMP></axiom>
</theory>

<theory xml:id="nat">
  <adt>
    <sortdef name="Nat">
      <constructor name="zero"/>
      <constructor name="succ">
        <argument>
          <type><OMOBJ><OMS name="Nat" cd="nat"/></OMOBJ></type>
          <selector name="pred"/>
        </argument>
      </constructor>
    </sortdef>
  </adt>
</theory>

<theory xml:id="NatOrd">
  <imports from="#nat"/>
  <imports from="#TOSet"/>
  <symbol name="leq"/>
  <definition xml:id="leq.def" for="leq" type="implicit"
              existence="#leq.ex" uniqueness="#leq.uniq">
    <FMP>∀x.0 ≤ x ∧ ∀x, y.x ≤ y ⇒ s(x) ≤ s(y)</FMP>
  </definition>
  <assertion xml:id="leq.ex"><CMP><xhtml:p>≤ exists.</xhtml:p></CMP></assertion>
  <assertion xml:id="leq.unique"><CMP><xhtml:p>≤ is unique.</xhtml:p></CMP></assertion>
  <assertion xml:id="leq.TO"><CMP><xhtml:p>≤ is a total order on Nat.<xhtml:p></CMP></assertion>
</theory>
```

Finally we have extended the natural numbers by an ordering function $\leq$ (symbol `leq`) which we show to be a total ordering function in assertion `leq.TO`. Note that to state the assertion, we had to import the notion of a total ordering from theory `TOSet`. We can directly use this result to establish a **theory inclusion** between `TOSet` as the **source theory** and `NatOrd` as the **target theory**. A theory inclusion is a formula mapping between two theories, such that the translations of all axioms in the source theory are provable in the target theory. In our case, the mapping is given by the recursive function given in the `morphism` element in Listing 6 that maps the respective base sets and the ordering relations to each other. The `obligation` element just states that translation of the only theory-constitutive (see Subsection 15.2.4) element of the source theory (the axiom `toset`) has been proven in the target theory, as witnessed by the assertion `leq.TO`[1].

```
<theory−inclusion xml:id=”elem−nat−incl” to=”#NatOrd” from=”#TOSet”>
  <morphism xml:id=”elem−nat” type=”pattern”>
    <requation>
      <OMOBJ><OMS cd=”TOSet” name=”set”/></OMOBJ>
      <OMOBJ><OMS cd=”NatOrd” name=”Nat”/></OMOBJ>
    </requation>
    <requation>
      <OMOBJ><OMS cd=”TOSet” name=”ord”/></OMOBJ>
      <OMOBJ><OMS cd=”NatOrd” name=”leq”/></OMOBJ>
    </requation>
  </morphism>
  <obligation induced−by=”#toset” assertion=”#leq.TO”/>
</theory−inclusion>
```

We continue our example by building a generic theory `OrdList` of ordered lists. This is given as the abstract data type generated by the symbols `cons` (construct a list from an element and a rest list) and `nil` (the empty list) together with a defined symbol `ordered`: a predicate for ordered lists. Note that this symbol cannot be given in the abstract data type, since it is not a constructor symbol. Note that `OrdList` imports theory `TOSet` for the base set of the lists and the ordering relation $\leq$.

```
<theory xml:id=”OrdList”>
  <imports from=”#TOSet”/>
  <adt xml:id=”list−adt”>
    <sortdef name=”lists”>
      <constructor name=”cons”>
        <argument>
          <type><OMOBJ><OMS name=”set” cd=”TOSet”/></OMOBJ></type>
          <selector name=”head”/>
        </argument>
        <argument>
          <type><OMOBJ><OMS name=”lists” cd=”OrdList”/></OMOBJ></type>
          <selector name=”rest”/>
        </argument>
      </constructor>
      <constructor name=”nil”/>
```

[1] Note that as always, OMDoc only cares about the structural aspects of this: The OMDoc model only insists that there is the statement of an assertion, whether the author chooses to prove it or indeed whether the statement is true at all is left to other levels of modeling.

```
    </sortdef>
  </adt>

  <symbol name="ordered"/>
  <definition xml:id="ordered-def" for="ordered" type="informal">
    <CMP><xhtml:p>A list l is called ordered, iff head(l) ≤ z for all elements z ∈ rest(l) and
    rest(l) is ordered.</xhtml:p></CMP>
  </definition>
</theory>
```

The theory `NatOrdList` of lists of natural numbers is built up by importing from the theories `NatOrd` and `OrdList`. Note that the attribute `type` of the `imports` element `nat-list.im-elt` is set to `local`, since we only want to import the local axioms of the theory `OrdList` and not the whole theory `OrdList` (which would include the axioms from `TOSet`; see Section 18.3 for a discussion). In particular the symbols `set` and `ord` are not imported into theory `NatOrdList`: the theory `TOSet` is considered as a **formal parameter theory**, which is actualized to the **actual parameter theory** with this construction. The effect of the actualization comes from the morphism `elem-nat` in the import of `OrdList` that renames the symbol `set` (from theory `TOSet`) with `Nat` (from theory `NatOrd`). The actualization from `OrdList` to `NatOrdList` only makes sense, if the parameter theory `NatOrd` also has a suitable ordering function. This can be ensured using the OMDoc `inclusion` element.

```
<theory xml:id="NatOrdList">
  <imports xml:id="natordlist.im-natord" from="#NatOrd"/>
  <imports xml:id="natordlist.im-elt" from="#OrdList" type="local">
    <morphism base="#elem-nat"/>
  </imports>
  <inclusion via="elem-nat-incl"/>
</theory>
```

The benefit of this `inclusion` requirement is twofold: If the theory inclusion from `TOSet` to `NatOrd` cannot be verified, then the theory `NatOrdList` is considered to be undefined, and we can use the development graph techniques presented in Section **??** to obtain a theory inclusion from `OrdList` to `NatOrdList`: We first establish an axiom inclusion from theory `TOSet` to `NatOrdList` by observing that this is induced by composing the theory inclusion from `TOSet` to `NatOrd` with the theory inclusion given by the `imports` from `NatOrd` to `NatOrdList`. This gives us a **decomposition** situation: every theory that the source theory `OrdList` inherits from has an axiom inclusion to the target theory `NatOrdList`, so the local axioms of those theories are provable in the target theory. Since we have covered all of the inherited ones, we actually have a theory inclusion from the source- to the target theory.

```
<axiom-inclusion xml:id="toset-natordlist-incl" from="#TOSet" to="#NatOrdList">
  <morphism base="#elem-nat"/>
  <path-just local="#elem-nat-incl" globals="#natordlist.im-natord"/>
</axiom-inclusion>

<theory-inclusion from="#OrdList" to="#NatOrdList">
  <morphism base="#elem-nat"/>
  <decomposition links="#toset-natordlist-incl #elem-nat-incl"/>
```

</theory−inclusion>

This concludes our example, since we have seen that the theory `OrdList` is indeed included in `NatOrdList` via renaming.

Note that with this construction we could simply extend the graph by actualizations for other theories, e.g. to get lists of characters, as long as we can prove theory inclusions from `TOSet` to them.

7

A Development Graph for Elementary Algebra

We will now use the technique presented in the last chapter for the elementary algebraic hierarchy. Figure 7.1 gives an overview of the situation. We will build up theories for semigroups, monoids, groups, and rings and a set of theory inclusions from these theories to themselves given by the converse of the operation.

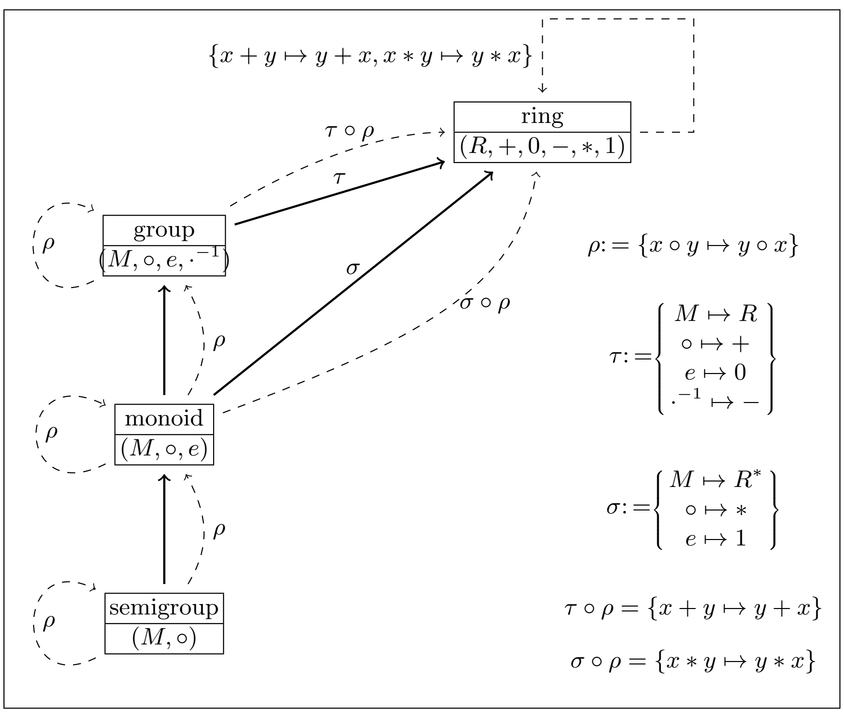


Fig. 7.1. A Development Graph for Elementary Algebra

We start off with the theory for *semigroups*. It introduces two symbols, the base set M and the operation $\circ$ on M together with two axioms that state that M is closed under $\circ$ and that $\circ$ is associative on M. We have a structural theory inclusion from this theory to itself that uses the fact that M together with the converse $\sigma(\circ)$ of $\circ$ is also a semigroup: the obligation for the axioms can be justified by themselves (for the closure axiom we have $\sigma(\forall x, y \in M.x \circ y \in M) = \forall y, x \in M.x \circ y \in M$, which is logically equivalent to the axiom.)

```
<theory xml:id="semigroup">
  <symbol name="base−set"/>
  <presentation for="#base−set"><use format="default">M</use></presentation>
  <symbol name="op"/>
  <presentation for="#op"><use format="default">∘</use></presentation>
  <axiom xml:id="closed.ax"><FMP>∀x, y ∈ M.x ∘ y ∈ M</FMP></axiom>
  <axiom xml:id="assoc.ax">
    <FMP>∀x, y, z ∈ M.(x ∘ y) ∘ z = x ∘ (y ∘ z)</FMP>
  </axiom>
</theory>

<theory−inclusion xml:id="sg−conv−sg" from="#semigroup" to="#semigroup">
  <morphism xml:id="sg−conv−sg.morphism">
    <requation>X ∘ Y ⇝ Y ∘ X</requation>
  </morphism>
  <obligation assertion="conv.closed" induced−by="#closed.ax"/>
  <obligation assertion="#assoc.ax" induced−by="#assoc.ax"/>
</theory−inclusion>
```

The theory of *monoids* is constructed as an extension of the theory of semigroups with the additional unit axiom, which states that there is an element that acts as a (right) unit for $\circ$. As always, we state that there is a unique such unit, which allows us to define a new symbol e using the definite description operator $\tau x.$: If there is a unique x, such that $\mathbf{A}$ is true, then the construction $\tau x.\mathbf{A}$ evaluates to x, and is undefined otherwise. We also prove that this e also acts as a left unit for $\circ$.

```
<theory xml:id="monoid">
  <imports xml:id="sg2mon" from="#semigroup"/>
  <axiom xml:id="unit.ax"><FMP>∃x ∈ M.∀y ∈ M.y ∘ x = y</FMP></axiom>
  <assertion xml:id="unit.unique"><FMP>∃¹x ∈ M.∀y ∈ M.y ∘ x = y</FMP></assertion>
  <symbol name="unit" xml:id=''unit''/>
  <presentation for="#unit"><use format="default">e</use></presentation>
  <definition xml:id="unit.def" for="unit" type="simple" existence="#unit.unique">
    τx ∈ M.∀y ∈ M.y ∘ x = y
  </definition>
  <assertion xml:id="left.unit"><FMP>∀x ∈ M.e ∘ x = x</FMP></assertion>
  <symbol name="setstar" xml:id=''setstar''/>
  <presentation for="#setstar" fixity="postfix">
    <use format="default">*</use>
  </presentation>
  <definition xml:id="ss.def" for="setstar" type="implicit">
    ∀S ⊆ M.S* = S\{e}
  </definition>
</theory>
```

Building on this, we first establish an axiom-selfinclusion from the theory of monoids to itself. We can make this into a theory selfinclusion using the theory-selfinclusion for semigroups as the local part of a path justification (recall that theory inclusions are axiom inclusions by construction) and the definitional theory inclusion induced by the import from semigroups to monoids as the global path.

```
<axiom-inclusion xml:id="mon-conv-mon.local" from="#monoid" to="#monoid">
  <morphism base="#sg-conv-sg.morphism"/>
  <obligation assertion="#left.unit" induced-by="#unit.ax"/>
</axiom-inclusion>

<axiom-inclusion xml:id="sg-conv-mon" from="#semigroup" to="#monoid">
  <morphism base="#sg-conv-sg.morphism"/>
  <path-just local="#sg-conv-sg" globals="#sg2mon"/>
</axiom-inclusion>
<theory-inclusion xml:id="mon-conv-mon.global" from="#monoid" to="#monoid">
  <morphism base="#sg-conv-sg.morphism"/>
  <decomposition links="#sg-conv-sg #sg-conv-mon"/>
</theory-inclusion>
```

Note that all of these axiom inclusions have the same morphism (denoted by ρ in Figure 7.1), in OMDoc we can share this structure using the `base` on the `morphism` element. This normally points to a morphism that is the base for extension, but if the `morphism` element is empty, then this just means that the morphisms are identical.

For groups, the situation is very similar: We first build a theory of groups by adding an axiom claiming the existence of inverses and constructing a new function $\cdot^{-1}$ from that via a definite description.

```
<theory xml:id="group">
  <imports xml:id="mon2grp" from="#monoid"/>
  <axiom xml:id="inv.ax"><FMP>∀x ∈ M.∃y ∈ M.x ∘ y = e</FMP></axiom>
  <symbol name="inv" xml:id=''inv''/>
  <presentation for="#inv" role="applied">
    <use format="default" lbrack="" rbrack="" fixity="postfix">⁻¹</use>
  </presentation>
  <definition xml:id="inv.def" for="inv" type="pattern">
    <requation>x⁻¹ ⇝ τy.x ∘ y = e</value></requation>
  </definition>
  <assertion xml:id="conv.inv"><FMP>∀x ∈ M.∃y ∈ M.y ∘ x = e</FMP></assertion>
</theory>
```

Again, we have to establish a couple of axiom inclusions to justify the theory inclusion of interest. Note that we have one more than in the case for monoids, since we are one level higher in the inheritance structure, also, the local chains are one element longer.

```
<axiom-inclusion xml:id="grp-conv-grp.local" from="#group" to="#group">
  <morphism base="#sg-conv-sg.morphism"/>
  <obligation assertion="conv.inv" induced-by="#inv.ax"/>
</axiom-inclusion>
<axiom-inclusion xml:id="sg-conv-grp" from="#semigroup" to="#group">
  <morphism base="#sg-conv-sg.morphism"/>
  <path-just local="#sg-conv-sg" globals="#mon2grp #sg2mon"/>
</axiom-inclusion>
<axiom-inclusion xml:id="mon-conv-grp" from="#monoid" to="#group">
  <morphism base="#sg-conv-sg.morphism"/>
  <path-just local="#mon-conv-mon.local" globals="#mon2grp"/>
</axiom-inclusion>
<theory-inclusion xml:id="grp-conv-grp" from="#group" to="#group">
  <morphism base="#sg-conv-sg.morphism"/>
  <decomposition links="#sg-conv-grp #mon-conv-grp #grp-conv-grp.local"/>
```

```
</theory−inclusion>
```

Finally, we extend the whole setup to a theory of rings. Note that we have a dual import from `group` and `monoid` with different morphisms (they are represented by σ and τ in Figure 7.1). These rename all of the imported symbols apart (interpreting them as additive and multiplicative) except of the punctuated set constructor $\cdot^*$, which is imported from the additive group structure only. We avoid a name clash with the operator that would have been imported from the multiplicative structure by specifying that this is not imported using the `hiding` on the `morphism` in the respective `imports` element[1].

```
<theory xml:id="ring">
  <symbol name="R" xml:id="R"/>
  <presentation for="#R"><use format="default">R</use></presentation>
  <symbol name="zero"/>
  <presentation for="#zero"><use format="default">0</use></presentation>
 <symbol name="plus"/>
 <presentation for="#plus" role="applied">
    <use format="default">+</use>
  </presentation>
 <symbol name="negative"/>
 <presentation for="#negative" role="applied">
    <use format="default">−</use>
  </presentation>
 <symbol name="times"/>
  <presentation for="#times" role="applied">
    <use format="default">∗</use>
  </presentation>
  <symbol name="one"/>
  <presentation for="#one"><use format="default">1</use></presentation>
  <imports xml:id="add.import" from="#group">
    <morphism>M ↦ R, x ∘ y ↦ x ∗ y, e ↦ 1, ·⁻¹ ↦ −</morphism>
  </imports>
  <imports xml:id="mult.import" from="#monoid">
    <morphism hiding="setstar">M ↦ M*, x ∘ y ↦ x ∗ y, e ↦ 1</morphism>
  </imports>
  <axiom xml:id="dist.ax"><FMP>x ∗ (y + z) = (x ∗ y) + (x ∗ z)</FMP></axiom>
  <assertion xml:id="dist.conv"><FMP>(z + y) ∗ x = (z ∗ x) + (y ∗ x)</FMP></assertion>
</theory>
```

Again, we have to establish some axiom inclusions to justify the theory selfinclusion we are after in the example. Note that in the rings case, things are more complicated, since we have a dual import in the theory of `rings`. Let us first establish the additive part.

```
<axiom−inclusion xml:id="sg−conv−rg.add" from="#semigroup" to="#ring">
  <morphism base="#sg−conv−sg.morphism #add.import"/>
  <path−just local="#sg−conv−sg" globals="#sg2mon #mon2grp #add.import "/>
</axiom−inclusion>
<axiom−inclusion xml:id="mon−conv−rg.add" from="#monoid" to="#group">
  <morphism base="#sg−conv−sg.morphism #add.import"/>
  <path−just local="#mon−conv−mon.local" globals="#mon2grp #add.import"/>
</axiom−inclusion>
```

[1] An alternative (probably better) to this would have been to explicitly include the operators in the morphisms, creating new operators for them in the theory of `rings`. But the present construction allows us to exemplify the `hiding`, which has not been covered in an example otherwise.

```
<axiom-inclusion xml:id="grp-conv-rg.add" from="#group" to="#group">
  <morphism base="#sg-conv-sg.morphism #add.import"/>
  <path-just local="#grp-conv-grp.local" globals="#add.import"/>
</axiom-inclusion>
```

The multiplicative part is totally analogous, we will elide it to conserve space. Using both parts, we can finally get to the local axiom self-inclusion and extend it to the intended theory inclusion justified by the axiom inclusions established above.

```
<axiom-inclusion xml:id="rg-conv-rg.local" from="#ring" to="#ring">
  <morphism xml:id="rg-conv-rg.morphism">x + y ↦ y + x, x ∗ y ↦ y ∗ x</morphism>
  <obligation assertion="#dist.conv" induced-by="#dist.ax"/>
</axiom-inclusion>
<theory-inclusion xml:id="rg-conv-rg" from="#ring" to="#ring">
  <morphism base="#rg-conv-rg.morphism"/>
  <decomposition links="#rg-conv-rg.local
                        #sg-conv-rg.add #mon-conv-rg.add #grp-conv-rg.add
                        #sg-conv-rg.mult #mon-conv-rg.mult #grp-conv-rg.mult"/>
</theory-inclusion>
```

This concludes our example. It could be extended to higher constructs in algebra like fields, magmas, or vector spaces easily enough using the same methods, but we have seen the key features already.

8

Courseware and the Narrative/Content Distinction

In this chapter we will look at another type of mathematical document: courseware; in this particular case a piece from an introductory course "Fundamentals of Computer Science" (Course 15-211 at Carnegie Mellon University). The OMDoc documents produced from such courseware can be used as input documents for ActiveMath (see Section **??**) and can be produced e.g. by **C**Point (see Section **??**).

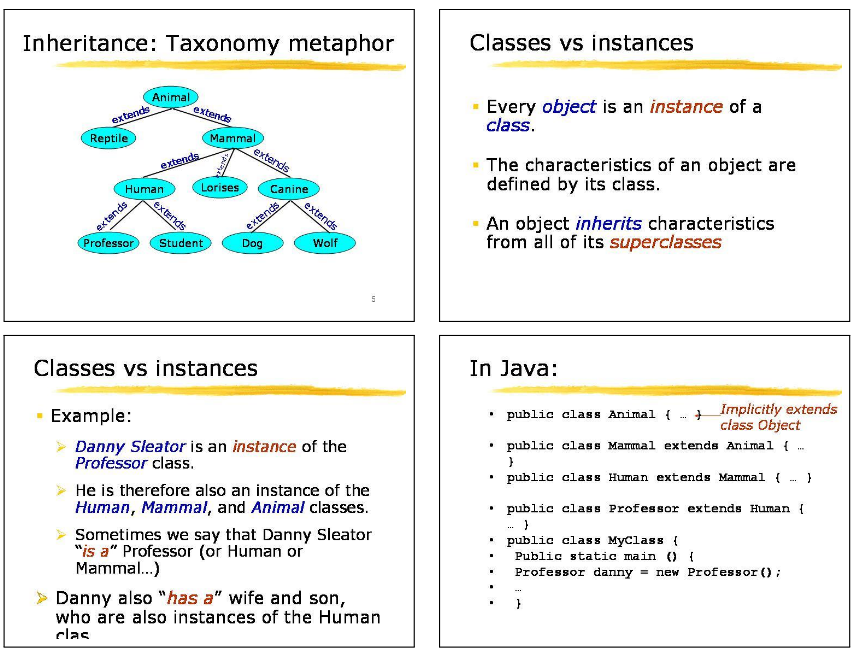


Fig. 8.1. Three slides from 15-211

We have chosen a fragment that is relatively far from conventional mathematical texts to present the possibility of semantic markup in OMDoc even under such circumstances. We will highlight the use of OMDoc theories for such an application. Furthermore, we will take seriously the difference between marking up the knowledge (implicitly) contained in the slides and the slide presentation as a structured document. As a consequence, we will capture the slides in *two* documents:

- a *knowledge-centered document*, which contains the knowledge conveyed in the course organized by its inherent logical structure
- a *narrative-structured document* references the knowledge items and adds rhetorical and didactic structure of a slide presentation.

This separation of concerns into two documents is good practice in marking up mathematical texts: It allows to make explicit the structure inherent in the respective domain and at the same time the structure of the presentation that is driven by didactic needs. We call knowledge-structured documents **content OMDoc**s and narrative-structured ones **narrative OMDoc**s. The separation also simplifies management of academic content: The content OMDoc of course will usually be shared between individual installments of the course, it will be added to, corrected, cross-referenced, and kept up to date by different authors. It will eventually embody the institutional memory of an organization like a university or a group of teachers. The accompanying narrative OMDocs will capture the different didactic tastes and approaches by individual teachers and can be adapted for the installments of the course. Since the narrative OMDocs are relatively light-weight structures (they are largely void of original content, which is referenced from the content OMDoc) constructing or tailoring a course to the needs of the particular audience becomes a simpler endeavor of choosing a path through a large repository of marked up knowledge embodied in the content OMDoc rather than re-authoring[1] the content with a new slant.

Let us look at the four slides in Figure 8.1. The first slide shows a graphic of a simple taxonomy of animals, the second one introduces first concepts from object-oriented programming, the third one gives examples for these interpreting the class hierarchy introduced in the first slide, finally the fourth slide gives code concrete snippets as examples for the concepts introduced in the first three ones.

We will first discuss content OMDoc and then the narrative OMDoc in Section 8.2.

[1] Since much of the re-authoring is done by copy and paste in the current model, it propagates errors in the course materials rather than corrections.

8.1 A Knowledge-Centered View

In this section, we will take a look at how we can make the knowledge that is contained in the slides in Figure 8.1 and its structure explicit so that a knowledge management system like MBase (see Section ??) or knowledge presentation system like ActiveMath (see Section ??) can take advantage of it. We will restrict ourselves to knowledge that is explicitly represented in the slides in some form, even though the knowledge document would probably acquire more and more knowledge in the form of examples, graphics, variant definitions, and explanatory text as it is re-used in many courses.

The first slide introduces a theory, which we call `animals-tax`; see Listing 8.1. It declares primitive symbols for all the concepts[2] (the ovals), and for all the links introduced in the graphic it has `axiom` elements stating that the parent node in the tree extends the child node. The axiom uses the symbol for concept extension from a theory `kr` for knowledge representation which we import in the theory and which we assume in the background materials for the course.

Listing 8.1. The OMDoc Representation for Slide 1 from Figure 8.1

```
<theory xml:id="animals-tax">
  <imports xml:id="tax_imports_taxonomy" from="#taxonomies"/>
  <imports xml:id="tax_imports_kr" from="#kr"/>
  <symbol name="human">
    <type system="stlc"><OMOBJ><OMS cd="kr" name="concept"/></OMOBJ></type>
  </symbol>
  <symbol name="mammal">
    <type system="stlc"><OMOBJ><OMS cd="kr" name="concept"/></OMOBJ></type>
  </symbol>
  ...
  <axiom xml:id="mammal-ext-human">
    <CMP><xhtml:p>Humans are Animals.</xhtml:p></CMP>
    <FMP>
      <OMOBJ>
        <OMA><OMS cd="kr" name="extends"/>
          <OMS cd="animal-taxonomy" name="mammal"/>
          <OMS cd="animal-taxonomy" name="human"/>
        </OMA>
      </OMOBJ>
    </FMP>
  </axiom>
  ...
</theory>

<private xml:id="tax-image" for="#animals-tax" reformulates="#animals-tax">
  <data format="image/jpeg" href="animals-taxonomy.jpg"/>
  <data format="application/postscript" href="animals-taxonomy.ps"/>
</private>
```

The `private` element contains the reference to the image in various formats. Its `reformulates` attribute hints that the image contained in this element can be used to illustrate the theory above (in fact, it will be the only thing used from this theory in the narrative OMDoc in Listing 8.6.)

[2] The type information in the symbols is not strictly included in the slides, but may represent the fact that the instructor said that the ovals represent "concepts".

The second slide introduces some basic concepts in object oriented programming. These give rise to the five primitive symbols of the theory. Note that this theory is basic, it does not import any other. The three text blocks are marked up as axioms, using the attribute `for` to specify the symbols involved in these axioms. The value of the `for` attribute is a whitespace-separated list of URI references to `symbol` elements.

Listing 8.2. The OMDoc Representation for Slide 2 from Figure 8.1

```
<theory xml:id="cvi">
  <symbol name="object" xml:id="cvi.object"/>
  <symbol name="instance" xml:id="cvi.instance"/>
  <symbol name="class" xml:id="cvi.class"/>
  <symbol name="inherits" xml:id="cvi.inherits"/>
  <symbol name="superclass" xml:id="cvi.superclass"/>

  <axiom xml:id="ax1" for="object instance class">
    <CMP>
      <xhtml:p>
        Every <xhtml:span style="font-style:italic;color:blue">object</xhtml:span>
        is an <xhtml:span style="font-style:italic; color:red">instance</xhtml:span>
        of a <xhtml:span style="font-style:italic; color:blue">class</xhtml:span>.
     </xhtml:p>
    </CMP>
  </axiom>

  <axiom xml:id="ax2" for="class">
    <CMP><xhtml:p>The characteristics of an object are defined by its class.</xhtml:p></CMP>
  </axiom>

  <axiom xml:id="ax3" for="inherits superclass">
    <CMP><xhtml:p>An object <xhtml:span style="font-style:italic;color:blue">inherits</xhtml:span>
      characteristics from all of its
      <xhtml:span style="font-style:italic; color:red">superclasses</xhtml:span>.</xhtml:p></CMP>
   </axiom>
</theory>
```

For the third slide it is not entirely obvious which of the OMDoc elements we want to use for markup. The intention of the slide is obviously to give some examples for the concepts introduced in the second slide in terms of the taxonomy presented in the first slide in Figure 8.1. However, the OMDoc `example` element seems to be too specific to directly capture the contents (see p. 163). What is immediately obvious is that the slide introduces some new knowledge and symbols, so we have to have a separate theory for this slide. The first item in the list headed by the word Example is a piece of new knowledge, it is therefore not an example at all, but an axiom[3]. The second item in the list is a statement that can be deduced from the knowledge we already have at our disposal from theories `animals-tax` and `cvi`. Therefore, the new theory `cvi-examples` in Listing 8.3 imports these two. Furthermore, it introduces the new symbol `danny` for “Danny Sleator” which is clarified in the `axiom` element with `xml:id="ax1"`. Finally, the third item in the list does not have the function of an example either, it introduces a new concept,

[3] We could say that the function of being an example has moved up from mathematical statements to mathematical theories; we will not pursue this here.

the "is a" relation[4]. So we arrive at the theory in Listing 8.3. Note that this markup treats the last text block on the third slide without semantic function in the theory – it points out that there are other relations among humans – and leaves it for the narrative-structured OMDoc in Section 8.2[5].

Listing 8.3. The OMDoc Representation for Slide 3 from Figure 8.1

```
<theory xml:id="cvi-examples">
  <imports from="#animals-tax"/><imports from="#cvi"/>

  <symbol name="danny" xml:id="cvi-examples.danny">
    <metadata><dc:description>Danny Sleator</dc:description></metadata>
  </symbol>

  <axiom xml:id="danny-professor" for="class instance danny">
    <CMP>
      <xhtml:p>
        <xhtml:span style="font-style:italic; color:blue">Danny Sleator</xhtml:span>
        is an <xhtml:span style="font-style:italic; color:red">instance</xhtml:span>
        of the <xhtml:span style="font-style:italic; color:blue">Professor</xhtml:span>
        class.
      </xhtml:p>
    </CMP>
  </axiom>

  <assertion xml:id="dannys-classes" type="theorem">
    <CMP>
      <xhtml:p>
        He is therefore also an instance of the
        <xhtml:span style="font-style:italic; color:blue">Human</xhtml:span>,
        <xhtml:span style="font-style:italic; color:blue">Mammal</xhtml:span>,
        <xhtml:span style="font-style:italic; color:blue">Animal</xhtml:span> classes.
      </xhtml:p>
    </CMP>
  </assertion>

  <symbol name="is_a" scope="global">
    <metadata><dc:subject>'is a' relation</dc:subject></metadata>
  </symbol>

  <definition xml:id="is_a-def" for="is_a" type="informal">
     <CMP>
        <xhtml:p>Sometimes we say that Danny Sleator
         &#x201C;<xhtml:span style="font-style:italic;color:red">is a</xhtml:span>&#x201D;
         Professor (or Human or Mammal&#x2026;)
       </xhtml:p>
     </CMP>
  </definition>
</theory>
```

An alternative, more semantic way to mark up the `assertion` element in the theory above would be to split it into multiple `assertion` and `example` elements, as in Listing 8.4, where we have also added formal content. We have split the assertion `dannys-classes` into three — we have only shown one of them in Listing 8.4 — separate assertions about class instances, and used them

[4] Actually, this text block introduces a new concept "by reference to examples", which is not a formal definition at all. We will neglect this for the moment.

[5] Of course this design decision is debatable, and depends on the intuitions of the author. We have mainly treated the text this way to show the possibilities of semantic markup

to justify the explicit examples. These are given as OMDoc `example` elements. The `for` attribute of an `example` element points to the concepts that are exemplified here (in this case the symbols for the concepts “instance”, “class” from the theory `cvi` and the concept “mammal” from the animal taxonomy). The `type` specifies that this is not a counter-example, and the `assertion` points to the justifying assertion. In this particular case, the reasoning behind the example is pretty straightforward (therefore it has been omitted in the slides), but we will make it explicit to show the mechanisms involved. The `assertion` element just re-states the assertion implicit in the example, we refrain from giving the formal statement in an `FMP` child here to save space. The `just-by` can be used to point to set of proofs for this assertion, in this case only the one given in Listing 8.4. We use the OMDoc `proof` element to mark up this proof. It contains a series of `derive` proof steps. In our case, the argument is very simple, we can see that Danny Sleator is an instance of the human class, using the knowledge that

1. Danny is a professor (from the axiom in the `cvi-examples` theory)
2. An object inherits all the characteristics from its superclasses (from the axiom `ax3` in the `cvi` theory)
3. The human class is a superclass of the professor class (from the axiom `human-extends-professor` in the `animal-taxonomy` theory).

The use of this knowledge in the proof step is made explicit by the `premise` children of the `derive` element.

The information in the proof could for instance be used to generate very detailed explanations for students who need help understanding the content of the original slides in Figure 8.1.

Listing 8.4. An Alternative Representation Using `example` Elements

```
...
<example xml:id="danny-mammal" type="for" assertion="#dannys-mammal-thm"
         for="#cvi.instance #cvi.class #animal-taxonomy.mammal">
  <CMP><xhtml:p>Danny Sleator is an instance of the
    <xhtml:span style="font-style:italic; color:blue">Mammal</xhtml:span> class.
  <xhtml:p></CMP>
  <OMOBJ><OMS cd="cvi-examples" name="danny"/></OMOBJ>
</example>

<assertion xml:id="dannys-mammal-thm" type="theorem" proofs="#danny-mammal-pf">
  <CMP><xhtml:p>Danny Sleator is an instance of the Human class.</xhtml:p></CMP>
</assertion>

<proof xml:id="danny-human-pf" for="#dannys-mammal-thm">
  <derive xml:id="d1">
    <CMP><xhtml:p>Danny Sleator is an instance of the human class.</xhtml:p></CMP>
    <method>
      <premise xref="#danny-professor"/>
      <premise xref="#cvi.ax3"/>
      <premise xref="#animal-tax.human-extends-professor"/>
    </method>
  </derive>
  <derive xml:id="concl">
    <CMP><xhtml:p>Therefore he is an instance of the human class.</xhtml:p></CMP>
    <method>
```

```
      <premise xref="#d1"/>
      <premise xref="#cvi.ax3"/>
      <premise xref="#animal−tax.mammal−extends−human"/>
    </method>
  </derive>
</proof>
...
```

The last slide contains a set of Java code fragments that are related to the material before. We have marked them up in the `code` elements in Listing 8.5. The actual code is encapsulated in a `data` element, whose `format` specifies the format the data is in. The program text is encapsulated in a CDATA section to suspend the XML parser (there might be characters like `<` or `&` in there which offend it). The `code` elements allow to document the input, output, and side-effects in `input`, `output`, `effect` elements as children of the `code` elements. Since the code fragments in question do not have input or output, we have only described the side-effect (class declaration and class extension). As the code elements do not introduce any new symbols, definitions or axioms, we do not have to place them in a theory. The second `code` element also carries a `requires` attribute, which specifies that to execute this code snippet, we need the previous one. An application can use this information to make sure that one is loaded before executing this code fragment.

Listing 8.5. OMDoc Representation of Program Code

```
<code xml:id="cvic−code1">
  <data format="Java"><![CDATA[public class Animal {... }]]></data>
  <effect><CMP>class declaration</CMP></effect>
</code>

<code xml:id="cvic−code2" requires="cvic−code1" >
  <data format="Java"><![CDATA[public class Mammal extends Animal {...}]]></data>
  <effect><CMP>class extension</CMP></effect>
</code>
...
```

8.2 A Narrative-Structured View

In this section we present an OMDoc document that captures the structure of the slide show as a document. It references the knowledge items from the theories presented in the last section and adds rhetorical and didactic structure of a slide presentation.

The individual slides are represented as `omgroup` elements with `type` `slide`.

The representation of the first slide in Figure 8.1 is rather straightforward: we use the `dc:title` element in `metadata` to represent the slide title. Its `class` attribute references a CSS `class` definition in a style file. To represent the image with the taxonomy tree we use an `omtext` element with an `omlet` element.

The second slide marks up the list structure of the slide with the `omgroup` element (the value `itemize` identifies it as an itemizes list). The items in the list are given by OMDoc references (see Section **??**) to the axioms in the knowledge-structured document (see Listing 8.2). The effect of this markup is shared between the document: the content of the axioms are copied over from the knowledge-structured document, when the narrative-structured is presented to the user. However, the OMDoc references cascades its `style` attribute (and the `class` attribute, if present) with the `style` and `class` attributes of the target element, essentially adding style directives during the copying process (see Section **??** for details). In our example, this adds positioning information and specifies a particular image for the list bullet type.

Listing 8.6. The Narrative OMDoc for Figure 8.1

```
...
<omgroup xml:id="slide-847" type="slide">
  <metadata>
    <dc:title  class="15-211-title">Inheritance: Taxonomy metaphor</dc:title>
  </metadata>

  <omtext xml:id="the-tax">
    <CMP><xhtml:p>
      <omlet data="#tax-image" style="width:540;height:366"
             action="display" show="embed"/></xhtml:p>
    </CMP>
  </omtext>
</omgroup>

<omgroup xml:id="slide-848" type="slide">
  <metadata><dc:title class="15-211-title">Classes vs. instances</dc:title></metadata>
  <omgroup type="itemize" style="list-style-type:url(square.gif)">
    <axiom style="position:30% 10%" xml:id="obj" xref="slide1_content.omdoc#ax1"/>
    <axiom style="position:55% 10%" xml:id="class" xref="slide1_content.omdoc#ax2"/>
    <axiom style="position:80% 10%" xml:id="inh" xref="slide1_content.omdoc#ax3"/>
  </omgroup>
</omgroup>

<omgroup xml:id="slide-849" type="slide">
  <metadata><dc:title class="15-211-title">Classes vs. instances</dc:title></metadata>
  <omgroup type="itemize" style="list-style-type:url(square.gif)">
    <omtext style="position:30% 10%" xml:id="ex"><CMP><xhtml:p>Example:</xhtml:p></CMP></omtext>
    <omgroup type="itemize" style="list-style-type:url(triangle.gif)">
      <axiom style="position:400% 15%"
           xml:id="danny" xref="slide1_content.omdoc#danny-professor"/>
      <axiom style="position:55% 15%"
           xml:id="inst" xref="slide1_content.omdoc#dannys-classes"/>
      <axiom style="position:70% 15%" xml:id="is_a" xref="slide1_content.omdoc#is_a-def"/>
    </omgroup>
    <omtext style="position:83% 10%" xml:id="has_a">
      <CMP>
        <xhtml:p>Danny also &#x201C;<xhtml:span style="font-style:italic;color:red">has
        a</xhtml:span>&#x201D; wife and son, who are also instances of the Human class
        </xhtml:p></CMP>
    </omtext>
  </omgroup>
</omgroup>

<omgroup xml:id="slide-850" type="slide">
  <metadata><dc:title class="15-211-title">In Java</dc:title></metadata>
  <omgroup type="itemize">
    <omtext xml:id="slide-850.t1" style="position:80% 10%;color:red">
      <CMP><xhtml:p>Implicitly extends class object</xhtml:p></CMP>
```

```
    </omtext>
    <omtext xml:id="slide−850.t2">
      <CMP><xhtml:p><omlet data="#cvic−code1" action="display" show="embed"/></xhtml:p></CMP>
    </omtext>
    <omtext xml:id="slide−850.t3">
      <CMP><xhtml:p><omlet data="#cvic−code2" action="display" show="embed"/></xhtml:p></CMP>
    </omtext>
  </omgroup>
</omgroup>
...
```

8.3 Choreographing Narrative and Content OMDoc

The interplay between the narrative and content OMDoc above was relatively simple. The content OMDoc contained three theories that were linearized according to the dependency relation. This is often sufficient, but more complex rhetoric/didactic figures are also possible. For instance, when we introduce a new concept, we often first introduce a naive reduced approximation $\mathcal{N}$ of the real theory $\mathcal{F}$, only to show an example $\mathcal{E}_\mathcal{N}$ of where this is insufficient. Then we propose a first (straw-man) solution $\mathcal{S}$, and show an example $\mathcal{E}_\mathcal{S}$ of why this does not work. Based on the information we gleaned from this failed attempt, we build the eventual version $\mathcal{F}$ of the concept or theory and demonstrate that this works on $\mathcal{E}_\mathcal{F}$.

Let us visualize the narrative- and content structure in Figure 8.2. The structure with the solid lines and boxes at the bottom of the diagram represents the content structure, where the boxes $\mathcal{N}$, $\mathcal{E}_\mathcal{N}$, $\mathcal{S}$, $\mathcal{E}_\mathcal{S}$, $\mathcal{F}$, and $\mathcal{E}_\mathcal{F}$ signify theories for the content of the respective concepts and examples, much in the way we had them in Section 8.1. The arrows represent the theory inheritance structure, e.g. Theory $\mathcal{F}$ imports theory $\mathcal{N}$.

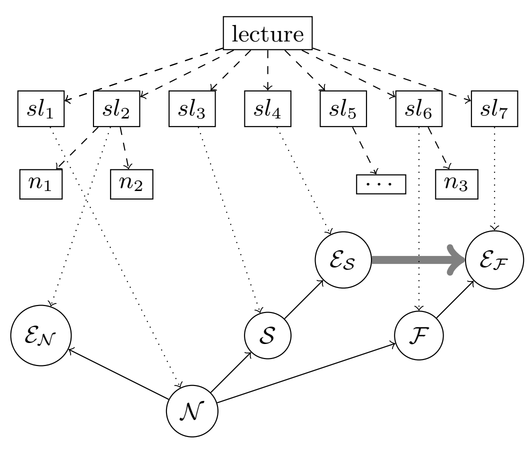


Fig. 8.2. An Introduction of a Concept via a Straw-Man Theory

The top part of the diagram with the dashed lines stands for the narrative structure, where the arrows mark up the document structure. For instance, the slides sl_i are grouped into a lecture. The dashed lines between the two documents visualize OMDoc references with pointers into the content structure. In the example in Figure 8.2, the second slide of "lecture" presents the first example: the text fragment n_1 links the content $\mathcal{E}_{\mathcal{N}}$, which is referenced from the content structure, to slide 1. The fragment n_2 might say something like "this did not work in the current situation, so we have to extend the conceptualization…".

Just as for content-based systems on the formula level, there are now MKM systems that generate presentation markup from content markup, based on general presentation principles, also on this level. For instance, the ActiveMath system [Mel+03] generates a simple narrative structure (the presentation; called a personalized book) from the underlying content structure (given in OMDoc) and a user model.

8.4 Summary

As we have seen, the narrative and content fulfill different, but legitimate content markup needs, that can coincide (as in the main example in this chapter), but need not (as in the example in the last section). In the simple case, where the dependency and narrative structure largely coincide, systems like the ActiveMath system described in Section **??** can generate narrative OMDocs from content OMDocs automatically. To generate more complex rhetoric/didactic figures, we would have to have more explicit markup for relations like "can act as a straw-man for". Providing standardized markup for such relations is beyond the scope of the OMDoc format, but could easily be expressed as metadata, or as external, e.g. RDF-based relations.

9

Communication with and between Mathematical Software Systems

OMDoc can be used as content language for communication protocols between mathematical software systems on the Internet. The ability to specify the context and meaning of the mathematical objects makes the OMDoc format ideally suited for this task.

In this chapter we will discuss a message interface in a fictitious software system MathWeb-WS[1], which connects a wide-range of reasoning systems (*mathematical services*), such as automated theorem provers, automated proof assistants, computer algebra systems, model generators, constraint solvers, human interaction units, and automated concept formation systems, by a common *mathematical software bus*. Reasoning systems integrated in MathWeb-WS can therefore offer new services to the pool of services, and can in turn use all services offered by other systems.

On the protocol level, MathWeb-WS uses Soap remote procedure calls with the HTTP binding [Gud+03] (see [Mit03] for an introduction to Soap) interface that allows client applications to request service objects and to use their service methods. For instance, a client can simply request a service object for the automated theorem prover Spass [Wei97] via the HTTP `GET` request in Listing 9.1 to a MathWeb-WS broker node.

Listing 9.1. Discovering Automated Theorem Provers (Request)

```
GET /ws.mathweb.org/broker/getService?name=SPASS HTTP/1.1
Host: ws.mathweb.org
```

[1] "MathWeb **W**eb **S**ervices"; The examples discussed in this chapter are inspired by the MathWeb-SB [FK99; ZK02] ("MathWeb **S**oftware **B**us") service infrastructure, which offers similar functionality based on the XML-RPC protocol (an XML encoding of Remote Procedure Calls (RPC) [Com]). We use the Soap-based formulation, since Soap (**S**imple **O**bject **A**ccess **P**rotocol) is the relevant W3C standard and we can show the embedding of OMDoc fragments into other XML namespaces. In XML-RPC, the XML representations of the content language OMDoc would be transported as base-64-encoded strings, not as embedded XML fragments.

```
Accept: application/soap+xml
```

As a result, the client receives a Soap message like the one in Listing 9.2 containing information about various instances of services embodying the Spass prover known to the broker service.

Listing 9.2. Discovering Automated Theorem Provers (Response)

```
   HTTP/1.1 200 OK
2  Content-Type: application/soap+xml
   Content-Length: 990

   <?xml version='1.0'?>
   <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
7    <env:Body>
       <ws:prover env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
           xmlns:ws="http://www.mathweb.org/ws-fictional">
         <ws:name>SPASS</ws:name>
         <ws:version>2.1</ws:version>
12       <ws:URL>http://spass.mpi-sb.mpg.de/webspass/soap</ws:URL>
         <ws:uptime>P3D5H6M45S</ws:uptime>
         <ws:sysinfo>
           <ws:ostype>SunOS 5.6</ws:ostype>
           <ws:mips>3825</ws:mips>
17       </ws:sysinfo>
       </ws:prover>
       <ws:prover env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
           xmlns:ws="http://www.mathweb.org/ws-fictional">
         <ws:name>SPASS</ws:name>
22       <ws:version>2.0</ws:version>
         <ws:URL>http://asuka.mt.cs.cmu.edu/atp/spass/soap</ws:URL>
         <ws:uptime>P5M2D15H56M5S</ws:uptime>
         <ws:sysinfo>
           <ws:ostype>linux-2.4.20</ws:ostype>
27         <ws:mips>1468</ws:mips>
         </ws:sysinfo>
       <ws:prover>
     </env:Body>
   </end:Envelope>
```

The client can then select one of the provers (say the first one, because it runs on the faster machine) and post theorem proving requests like the one in Listing 9.3[2] to the URL which uniquely identifies the service object in the Internet (this was part of the information given by the broker; see line 11 in Listing 9.2).

Listing 9.3. A Soap RPC call to Spass

```
POST http://spass.mpi-sb.mpg.de/webspass/soap HTTP/1.1
Host: http://spass.mpi-sb.mpg.de/webspass/soap
Content-Type: application/soap+xml;
Content-Length: 1123

<?xml version='1.0'?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>
    <ws:prove env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
        xmlns:ws="http://www.mathweb.org/ws-fictional">
      <omdoc:assertion xml:id="peter-hates-somebody" type="conjecture"
```

[2] We have made the namespaces involved explicit with prefixes in the examples, to show the mixing of different XML languages.

```
            xmlns:omdoc="http://omdoc.org/ns"
            theory="http://mbase.mathweb.org:8080/RPC2#lovelife">
        <omdoc:CMP>Peter hates somebody</omdoc:CMP>
        <omdoc:FMP>
          <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath">
            <om:OMBIND>
              <om:OMS cd="quant1" name="exists"/>
              <om:OMBVAR><om:OMV name="X"/></om:OMBVAR>
              <om:OMA>
                <om:OMS cd="lovelife" name="hate"/>
                  <om:OMS cd="lovelife" name="peter"/>
                  <om:OMV name="X"/>
              </om:OMA>
            </om:OMBIND>
          </om:OMOBJ>
        </omdoc:FMP>
      </omdoc:assertion>
      <ws:replyWith><ws:state>proof</ws:state></ws:replyWith>
      <ws:timeout>20</ws:timeout>
    </ws:prove>
  </env:Body>
</env:Envelope>
```

This Soap remote procedure call uses a generic method "`prove`" that can be understood by the first-order theorem provers on MathWeb-SB, and in particular the Spass system. This method is encoded as a `ws:prove` element; its children describe the proof problem and are interpreted by the Soap RPC node as a parameter list for the method invocation. The first parameter is an OMDoc representation of the assertion to be proven. The other parameters instruct the theorem prover service to reply with the proof (instead of e.g. just a yes/no answer) and gives it a time limit of 20 seconds to find it.

Note that OMDoc fragments can be seamlessly integrated into an XML message format like Soap. A Soap implementation in the client's implementation language simplifies this process drastically since it abstracts from HTTP protocol details and offers Soap nodes using data structures of the host language. As a consequence, developing MathWeb clients is quite simple in such languages. Last but not least, both MS Internet Explorer and the open source WWW browser FireFox now allow to perform Soap calls within JavaScript. This opens new opportunities for building user interfaces based on web browsers.

Note furthermore that the example in Listing 9.3 depends on the information given in the theory `lovelife` referenced in the `theory` attribute in the `assertion` element (see Section 15.6 for a discussion of the theory structure in OMDoc). In our instance, this theory might contain formalizations (in first-order logic) of the information that Peter hates everybody that Mary loves and that Mary loves Peter, which would allow Spass to prove the assertion. To get the information, the MathWeb-WS service based on Spass would first have to retrieve the relevant information from a knowledge base like the MBase system described in Section **??** and pass it to the Spass theorem prover as background information. As MBase is also a MathWeb-WS server, this can be done by sending the query in Listing 9.4 to the MBase service at `http://mbase.mathweb.org:8080`.

Listing 9.4. Requesting a Theory from MBase

```
GET /mbase.mathweb.org:8080/soap/getTheory?name=lovelife HTTP/1.1
Host: mbase.mathweb.org:8080
Accept: application/soap+xml
```

The answer would be of the form given in Listing 9.5. Here, the Soap envelope contains the OMDoc representation of the requested theory (irrespective of what the internal representation of MBase was).

Listing 9.5. The Background Theory for Message 9.3

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml
Content-Length: 602

<?xml version='1.0'?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>
    <theory xml:id="lovelife" xmlns="http://omdoc.org/ns">
      <symbol name="peter"/><symbol name="mary"/>
      <symbol name="love"/><symbol name="hate"/>
      <axiom xml:id="opposite">
        <CMP><xhtml:p>Peter hates everybody Mary loves</xhtml:p></CMP>
        <FMP>∀x.loves(mary, x) ⇒ hates(peter, x)</FMP>
      </axiom>
      <axiom xml:id="mary-loves-peter">
        <CMP><xhtml:p>Mary loves Peter</xhtml:p></CMP>
        <FMP>loves(mary, peter)</FMP>
      </axiom>
    </theory>
  </env:Body>
</env:Envelope>
```

This information is sufficient to prove the theorem in Listing 9.3; and the Spass service might reply to the request with the message in Listing 9.6 which contains an OMDoc representation of a proof (see Chapter 17 for details). Note that the `for` attribute in the `proof` element points to the original assertion from Listing 9.3.

Listing 9.6. A proof that Peter hates someone

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml
Content-Length: 588

<?xml version='1.0'?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>
    <proof xml:id="p347" for="#peter-hates-somebody"
          xmlns="http://omdoc.org/ns">
      <derive xml:id="d1">
        <FMP>hates(peter, peter)</FMP>
        <method xref="nd.omdoc#ND.chain">
          <premise xref="#lovelife.mary-loves-peter"/>
          <premise xref="#lovelife.opposite"/>
        </method>
      </derive>
      <derive xml:id="concl">
        <method xref="nd.omdoc#ND.ExI"><premise xref="#d1"/></method>
      </derive>
    </proof>
```

```
  </env:Body>
</env:Envelope>
```

The proof has two steps: The first one is represented in the `derive` element, which states that "Peter hates Peter". This fact is derived from the two axioms in the theory `lovelife` in Listing 9.5 (the `premise` elements point to them) by the "chaining rule" of the natural deduction calculus. This inference rule is represented by a symbol in the theory `ND` and referred to by the `xref` attribute in the `method` element. The second proof step is given in the second `derive` element and concludes the proof. Since the assertion of the conclusion is the statement of the proven assertion, we do not have a separate `FMP` element that states this here. The sole premise of this proof step is the previous one. For details on the representation of proofs in OMDoc see Chapter 17.

Note that the Spass theorem prover does not in itself give proofs in the natural deduction calculus, so the Spass service that provided this answer presumably enlisted the help of another MathWeb-WS service like the Tramp system [Mei00] that transforms resolution proofs (the native format of the Spass prover) to natural deduction proofs.

Part III

The OMDoc Document Format

The OMDoc (Open Mathematical Documents) format is a content markup scheme for (collections of) mathematical documents including articles, textbooks, interactive books, and courses. OMDoc also serves as the content language for agent communication of mathematical services on a mathematical software bus.

This part of the book is the specification of version 1.3 of the OMDoc format, the final and mature release of OMDoc version 1. It defines the OMDoc language features and their meaning. The content of this part is normative for the OMDoc format; an OMDoc document is valid as an OMDoc document, iff it meets all the constraints imposed here. OMDoc applications will normally presuppose valid OMDoc documents and only exhibit the intended behavior on such.

10

General Aspects of the OMDoc Format

10.1 OMDoc as a Modular Format

A modular approach to design is generally accepted as best practice in the development of any type of complex application. It separates the application's functionality into a number of "building blocks" or "modules", which are subsequently combined according to specific rules to form the entire application. This approach offers numerous advantages: The increased conceptual clarity allows developers to share ideas and code, and it encourages reuse by creating well-defined modules that perform a particular task. Modularization also reduces complexity by decomposition of the application's functionality and thus decreases debugging time by localizing errors due to design changes. Finally, flexibility and maintainability of the application are increased because single modules can be upgraded or replaced independently of others.

The OMDoc vocabulary has been split by thematic role, which we will briefly overview in Figures 10.1 and 10.2 before we go into the specifics of the respective modules in Chapters 13 to 21. To avoid repetition, we will introduce some attributes already in this chapter that are shared by elements from all modules. In Chapter 22 we will discuss the OMDoc document model and possible sub-languages of OMDoc that only make use of parts of the functionality (Section 22.3).

The modules in Figure 10.1 are required (mathematical documents without them do not really make sense), the ones in Figure 10.2 are optional.

The document-structuring elements in module DOC have an attribute `modules` that allows to specify which of the modules are used in a particular document (see Chapter 11 and Section 22.3).

10.2 The OMDoc Namespaces

The namespace for the OMDoc format is the URI `http://omdoc.org/ns`. Note that the OMDoc namespace does not reflect the versions, this is done in

Module	Title	Required?	Chapter
MOBJ	Mathematical Objects	yes	Chapter 13
Formulae are a central part of mathematical documents; this module integrates the content-oriented representation formats OPENMATH *and* MATHML *into* OMDOC			
MTXT	Mathematical Text	yes	Chapter 14
Mathematical vernacular, i.e. natural language with embedded formulae			
DOC	Document Infrastructure	yes	Chapter 11
A basic infrastructure for assembling pieces of mathematical knowledge into functional documents and referencing their parts			
RT	Rich Text Structure	no	Section **??**
Rich text structure in mathematical vernacular (lists, paragraphs, tables, ...)			
ST	Mathematical Statements	no	Chapter 15
Markup for mathematical forms like theorems, axioms, definitions, and examples that can be used to specify or define properties of given mathematical objects and theories to group mathematical statements and provide a notion of context.			
PF	Proofs and proof objects	no	Chapter 17
Structure of proofs and argumentations at various levels of details and formality			
PRES	Presentation Information	no	Chapter 19
Limited functionality for specifying presentation and notation information for local typographic conventions that cannot be determined by general principles alone			

Fig. 10.1. The OMDOC Modules

the `version` attribute on the document root element `omdoc` (see Chapter 11). As a consequence, the OMDOC vocabulary identified by this namespace is not static, it can change with each new OMDOC version. However, if it does, the changes will be documented in later versions of the specification: the latest released version can be found at [OMDoc].

In an OMDOC document, the OMDOC namespace must be specified either using a namespace declaration of the form `xmlns="http://omdoc.org/ns"` on the `omdoc` element or by prefixing the local names of the OMDOC elements by a namespace prefix (OMDOC customarily use the prefixes `omdoc:` or `o:`) that is declared by a namespace prefix declaration of the form `xmlns:o="http://omdoc.org/ns"` on some element dominating the OMDOC element in question (see Section 1.3 for an introduction). OMDOC also uses the namespaces in Figure 10.3[1] Thus a typical document root of an OMDOC document looks as follows:

[1] In this specification we will use the namespace prefixes above on all the elements we reference in text unless they are in the OMDOC namespace.

Module	Title	Required?	Chapter
DC	Dublin Core Metadata	yes	Sections 12.2 and 12.3
*Contains bibliographical "**data about data**", which can be used to annotate many OMDoc elements by descriptive and administrative information that facilitates navigation and organization*			
CC	Creative Commons Metadata	yes	Section 12.4
Licenses for text use			
ADT	Abstract Data Types	no	Chapter 16
Definition schemata for sets that are built up inductively from constructor symbols			
CTH	Complex Theories	no	Chapter 18
Theory morphisms; they can be used to structure mathematical theories			
DG	Development Graphs	no	Section **??**
Infrastructure for managing theory inclusions, change management			
EXT	Applets, Code, and Data	no	Chapter 20
Markup for applets, program code, and data (e.g. images, measurements, ...)			
QUIZ	Infrastructure for Assessments	no	Chapter 21
Markup for exercises integrated into the OMDoc document model			

Fig. 10.2. The OMDoc Modules

Format	namespace URI	see
Dublin Core	`http://purl.org/dc/elements/1.1/`	Sections 12.2 and 12.3
Creative Commons	`http://creativecommons.org/ns`	Section 12.4
MathML	`http://www.w3.org/1998/Math/MathML`	Section 13.2
OpenMath	`http://www.openmath.org/OpenMath`	Section 13.1
XSLT	`http://www.w3.org/1999/XSL/Transform`	Chapter 19

Fig. 10.3. OMDoc Namespaces

```
<?xml version="1.0" encoding="utf−8"?>
<omdoc xml:id="test.omdoc" version="1.3"
  xmlns="http://omdoc.org/ns"
  xmlns:cc="http://creativecommons.org/ns"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:om="http://www.openmath.org/OpenMath"
  xmlns:m="http://www.w3.org/1998/Math/MathML">
...
</omdoc>
```

10.3 Common Attributes in OMDoc

There are some attributes that are common to many OMDoc elements, so we will describe them here before we go into the specifics of the respective elements themselves

10.3.1 Foreign-Namespace Attributes

Generally, the OMDoc format allows any attributes from foreign (i.e. non-OMDoc) namespaces on the OMDoc elements. This is a commonly found feature that makes the XML encoding of the OMDoc format extensible. Note that the attributes defined in this specification are in the default (empty) namespace: they do not carry a namespace prefix. So any attribute of the form `na:xxx` is allowed as long as it is in the scope of a suitable namespace prefix declaration.

10.3.2 XML Identifiers

Many OMDoc elements have optional `xml:id` attributes that can be used as identifiers to reference them. These attributes are of type `ID`, they must be unique in the document which is important, since many XML applications offer functionality for referencing and retrieving elements by `ID`-type attributes. Note that unlike other `ID`-attributes, in this special case it is the name `xml:id` [MVW05] that defines the referencing and uniqueness functionality, not the type declaration in the DTD or XML schema (see Subsection 1.3.2 for a discussion).

Note that in the OMDoc format proper, all ID type attributes are of the form `xml:id`. However in the older OpenMath and MathML standards, they still have the form `id`. The latter are only recognized to be of type `ID`, if a document type or XMLschema is present. Therefore it depends on the application context, whether a DTD should be supplied with the OMDoc document.

10.3.3 CSS Attributes

For many occasions (e.g. for printing OMDoc documents), authors want to control a wide variety of aspects of the presentation. OMDoc is a content-oriented format, and as such only supplies an infrastructure to mark up content-relevant information in OMDoc elements. To address this dilemma XML offers an interface to Cascading Style Sheets (CSS) [Bos+98], which allow to specify presentational traits like text color, font variant, positioning, padding, or frames of layout boxes, and even aural aspects of the text.

To make use of CSS, most OMDoc elements (all that have `xml:id` attributes) have `style` attributes[2] that can be used to specify CSS directives for them. In the OMDoc fragment in Listing 10.1 we have used the `style` attribute to specify that the text content of the `omtext` element should be formatted in a centered box whose width is 80% of the surrounding box (probably the page box), and that has a 2 pixel wide solid frame of the specified RGB

[2] The treatment of the CSS attributes has changed from OMDoc1.1, see the discussion on page 225.

color. Generally CSS directives are of the form `A:V`, where `A` is the name of the aspect, and `V` is the value, several CSS directives can be combined in one `style` attribute as a semicolon-separated list (see [Bos+98] and the emerging CSS 3 standard).

Listing 10.1. Basic CSS Directives in a `style` Attribute

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/css" href="http://example.org/style.css"?>
<omdoc xml:id="stylish">
  ...
  <omtext xml:id="t1" style="width:80%;align:center;border:2px #006699 solid">
    <CMP><xhtml:p>Here comes something
      <xhtml:span style="font-weight:bold;color:green" class="emphasize">stylish</xhtml:span>!
    </xhtml:p></CMP>
  </omtext>
  ...
</omdoc>
```

Note that many CSS properties of parent elements are inherited by the children, if they are not explicitly specified in the child. We could for instance have set the font family of all the children of the `omtext` element by adding a directive `font-family:sans-serif` there and then override it by a directive for the property `font-family` in one of the children.

Frequently recurring groups of CSS directives can be given symbolic names in CSS style sheets, which can be referenced by the `class` attribute. In Listing 10.1 we have made use of this with the class `emphasize`, which we assume to be defined in the style sheet `style.css` associated with the document in the "style sheet processing instruction" in the prolog[3] of the XML document (see [Cla99a] for details). Note that an OMDoc element can have both `class` and `style` attributes, in this case, precedence is determined by the rules for CSS style sheets as specified in [Bos+98]. In our example in Listing 10.1 the directives in the `style` attribute take precedence over the CSS directives in the style sheet referenced by the `class` attribute on the `xhtml:span` element. As a consequence, the word "stylish" would appear in green, bold italics.

10.4 Structure Sharing

OMDoc is a content markup format, from which documents are produced via a presentation process. This "source character" of OMDoc documents allows to utilize structure sharing technologies in the markup[4]. For structure sharing OMDoc uses the `tref` attribute: all content elements can be used with the `tref` whose value is a URI reference to an OMDoc element instead of the normal element models. We call such an element an **OMDoc reference**. Semantically, OMDoc references are just placeholders for the OMDoc objects

[3] i.e. at the very beginning of the XML document before the document type declaration

[4] OMDoc 1.2 used the `ref` element with `type include` for this purpose. The new `tref`-based infrastructure supports validation much better.

they reference via their `tref` attribute. OMDoc references require OMDoc applications to process the document as if the OMDoc reference were replaced with the OMDoc fragment referenced in the `tref` attribute.

10.4.1 Ref-Reduction and Flattening

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

↔

```
<omgroup xml:id="text" type="sequence">
  <omtext tref="#t1"/>
  <omgroup tref="#enum"/>
  <omtext tref="#t4"/>
</omgroup>

<ignore type="targets"
        comment="already referenced">
  <omtext xml:id="t1">T1</omtext>
  <omtext xml:id="t2">T2</omtext>
  <omtext xml:id="t3">T3</omtext>
  <omtext xml:id="t4">T4</omtext>

  <omgroup xml:id="enum"
           type="enumeration">
    <omtext tref="#t2"/>
    <omtext tref="#t3"/>
  </omgroup>
</ignore>
```

Fig. 10.4. Flattening a Tree Structure

Let R be an OMDoc reference, we call the element the URI in the `tref` points to its **target**. We call the process of replacing an OMDoc reference by its target in a document **reference reduction**, and the document resulting from the process of systematically and recursively reducing all the OMDoc references the **ref normal form** of the source document. Note that ref-normalization may not always be possible, e.g. if the ref-targets do not exist or are inaccessible — or worse yet, if the relation given by the OMDoc references is cyclic. Moreover, even if it is possible to ref-normalize, this may not lead to a valid OMDoc document, e.g. since `ID` type attributes that were unique in the target documents are no longer in the ref-reduced one. We will call a document **ref-reducible**, iff its ref-normal form exists, and **ref-valid**, iff the ref-normal form exists and is a valid OMDoc document.

Note that it may make sense to use documents that are not ref-valid for narrative-centered documents, such as courseware or slides for talks that only allude to, but do not fully specify the knowledge structure of the mathematical knowledge involved. For instance the slides discussed in Section 8.2 do not contain the `theory` elements that would be needed to make the documents ref-valid.

OMDoc references also allow to "flatten" the tree structure in a document into a list of leaves and relation declarations (see Figure 10.4 for an example).

It also makes it possible to have more than one view on a document using `omgroup` structures that reference a shared set of OMDoc elements. Note that we have embedded the ref-targets of the top-level `omgroup` element into an `ignore` comment, so that an OMDoc transformation (e.g. to text form) does not encounter the same content twice.

10.4.2 Cascading of CSS Attributes

While the OMDoc approach to specifying document structure is a much more flexible (database-like) approach to representing structured documents[5] than the tree model, 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 10.4. Generally, any OMDoc element defines a fragment of the OMDoc it is contained in: everything between the start and end tags and (recursively) those elements that are reached from it by following the OMDoc references. In particular, the text fragment corresponding to the element with `xml:id="text"` in the right OMDoc of Figure 10.4 is just the one on the left.

In Section 10.3 we have introduced the CSS attributes `style` and `class`, which are present on all OMDoc elements. In the case of a OMDoc reference, there is a problem, since the content of these can be incompatible. In general, the rule for determining the style information for an element is that we treat the replacement element as if it were a child of the reference, and then determine the values of the CSS properties of the OMDoc reference by inheritance.

---

[5] 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 to make the structure of mathematical texts machine-understandable. Thus, an advanced presentation engine like the ActiveMath system [Sie+00] can — for instance — extract document fragments based on the preferences of the respective user.

11

Document Infrastructure (Module DOC)

Mathematical knowledge is largely communicated by way of a specialized set of documents (e.g. e-mails, letters, pre-prints, journal articles, and textbooks). These employ special notational conventions and visual representations to convey the mathematical knowledge reliably and efficiently.

When marking up mathematical knowledge, one always has the choice whether to mark up the structure of the document itself, or the structure of the mathematical knowledge that is conveyed in the document. Even though in most documents, the document structure is designed to help convey the structure of the knowledge, the two structures need not be the same. To frame the discussion we will distinguish two aspects of mathematical documents. In the *knowledge-centered view* we organize the mathematical knowledge by its function, and do not care about a way to present it to human recipients. In the *narrative-centered view* we are interested in the structure of the argument that is used to convey the mathematical knowledge to a human user.

We will call a document **knowledge-structured** and **narrative-structured**, based on which of the two aspects is prevalent in the organization of the material. Narrative-structured documents in mathematics are generally directed at human consumption (even without being in presentation markup). They have a general narrative structure: text interleaving with formal elements like assertions, proofs, ... Generally, the order of presentation plays a role in their effectiveness as a means of communication. Typical examples of this class are course materials or introductory textbooks. Knowledge-structured documents are generally directed at machine consumption or for referencing. They do not have a linear narrative spine and can be accessed randomly and even re-ordered without information loss. Typical examples of these are formula collections, OpenMath content dictionaries, technical specifications, etc.

The distinction between knowledge-structured and narrative-structured documents is reminiscent of the presentation vs. content distinction discussed in Section 2.1, but now it is on the level of document structure. Note that mathematical documents are often in both categories: a mathematical textbook can be read from front to end, but it can also be used as a reference, accessing it by the index and the table of contents. The way humans work with knowledge also involves a change of state. When we are taught or explore a mathematical domain, we have a linear/narrative path through the material, from which we abstract more and more, finally settling for a semantic representation that is relatively independent from the path we acquired it by. Systems like ActiveMath (see Section **??**) use the OMDoc format in exactly that way playing on the difference between the two classes and generating narrative-structured representations from knowledge-structured ones on the fly.

So, maybe the best way to think about this is that the question whether a document is narrative- or knowledge-structured is not a property of the document itself, but a property of the application processing this document.

OMDoc provides markup infrastructure for both aspects. In this chapter, we will discuss the infrastructure for the narrative aspect — for a working example we refer the reader to Chapter 8. We will look at markup elements for knowledge-structured documents in Section 15.6.

Even though the infrastructure for narrative aspects of mathematical documents is somewhat presentation-oriented, we will concentrate on content-markup for it. In particular, we will not concern ourselves with questions like font families, sizes, alignment, or positioning of text fragments. Like in most other XML applications, this kind of information can be specified in the CSS `style` and `class` attributes described in Section 10.3.

11.1 The Document Root

`omdoc`

The XML root element of the OMDoc format is the `omdoc` element, it contains all other elements described here. We call an OMDoc element a **top-level element**, if it can appear as a direct child of the `omdoc` element.

The `omdoc` element (and the `omgroup` element introduced below as well) has an optional attribute `xml:id` that can be used to reference the whole document. The `version` attribute is used to specify the version of the OMDoc format the file conforms to. It is fixed to the string `1.3` by this specification. This will prevent validation with a different version of the DTD or schema, or processing with an application using a different version of the OMDoc specification. The (optional) attribute `modules` allows to specify the OMDoc modules that are used in this document. The value of this attribute is a whitespace-separated list of module identifiers (e.g. MOBJ the left column in Figure **??**), OMDoc sub-language identifiers (see Figure 22.2), or URI references for externally given OMDoc modules or sub-language identifiers.[1]

[1] Allowing these external module references keeps the OMDoc format extensible. Like in the case with namespace URIs OMDoc do not mandate that these URI references reference an actual resource. They merely act as identifiers for the modules.

The intention is that if present, the `modules` specifies the list of all the modules used in the document (fragment). If a `modules` attribute is present, then it is an error, if the content of this element contains elements from a module that is not specified; spurious module declarations in the `modules` attributes are allowed.

The `omdoc` element acts as an implicit grouping element, just as the `omgroup` element to be introduced in Section 11.5. Both have an optional `type` attribute; we will discuss its values and meaning in Section 11.5.

Here and in the following we will use tables as the one in Figure 11.1 to give an overview over the respective OMDoc elements described in a chapter or section. The first column gives the element name, the second and third columns specify the required and optional attributes. We will use the fourth column labeled "DC" to indicate whether an OMDoc element can have a `metadata` child, which will be described in the next section. Finally the fifth column describes the content model — i.e. the allowable children — of the element. For this, we will use a form of Backus Naur Form notation also used in the DTD: `#PCDATA` stands for "parsed character data", i.e. text intermixed with legal OMDoc elements.) A synopsis of all elements is provided in Appendix B.

Element	Attributes		D	Content
	Required	Optional	C	
`omdoc`	`version`, `xmlns`	`xml:id, type, class, style, version, modules, theory`	+	⟪*front*⟫,(⟪*top-level*⟫)*,⟪*back*⟫
⟪*back*⟫			index?,bibliography?	
⟪*front*⟫			tableofcontents?	
`omgroup`		`xml:id, modules, type, class, style, theory`	+	(⟪*top-level*⟫)*
`metadata`		`xml:id, class, style`	–	⟪*MDelt*⟫*
`ref`	`xref`	`xml:id, type, class, style`	–	
`ignore`		`xml:id, type, comment`	–	ANY
`index`		`xml:id`	–	EMPTY
`bibliography`	`files`	`xml:id`	–	EMPTY
where ⟪*top-level*⟫ stands for top-level OMDoc elements, and ⟪*MDelt*⟫ for those introduced in Chapter 12				

Fig. 11.1. OMDoc Elements for Specifying Document Structure.

11.2 Front/Backmatter

Documents usually have and , OMDoc is no exception. Currently, the OMDoc front matter only consists of the `tableofcontents` element. The back matter consists of the optional elements `index` and `bibliography`.

The `tableofcontents` element represents the position of an table of contents in the document. Note that since OMDoc is a source format, we do not actually have to put the contents of the table of contents at this position, but `tableofcontents` only need to specify content properties of the table of contents is intended; the actual content can be generated by the presentation process. For that the `tableofcontents` element uses the optional `level` that can be used to specify the depth of the table of contents.

`bibliography`

The `bibliography` element is similar to `index`, but it specifies the position bibliography to be generated. The `bibliography` element has a single required attribute: the `files` specifies the bibliography files in LaTeXML form from which the actual references can be generated.

`index`

The `index` element represents the position of an index in the document.

11.3 Metadata

The World Wide Web was originally built for human consumption, and although everything on it is machine-readable, most of it is not machine-understandable. The accepted solution is to provide metadata (data about data) to describe the documents on the web in a machine-understandable format that can be processed automatically. Metadata commonly specifies aspects of a document like title, authorship, language usage, and administrative aspects like modification dates, distribution rights, and identifiers.

In general, metadata can either be embedded in the respective document, or be stated in a separate one. The first facilitates maintenance and control (metadata is always at your fingertips, and it can only be manipulated by the document's authors), the second one enables inference and distribution. OMDoc allows to embed metadata into the document, from where it can be harvested for external metadata formats, such as the XML resource description framework (RDF [LS99]). We use one of the best-known metadata schemata for documents – the *Dublin Core* (cf. Sections 12.2 and 12.3). The purpose of annotating metadata in OMDoc is to facilitate the administration of documents, e.g. digital rights management, and to generate input for metadata-based tools, e.g. RDF-based navigation and indexing of document collections. Unlike most other document formats OMDoc allows to add metadata at many levels, also making use of the metadata for document-internal markup purposes to ensure consistency.

`metadata`

The `metadata` element contains elements for various metadata formats including bibliographic data from the Dublin Core vocabulary (as mentioned above), licensing information from the Creative Commons Initiative (see Section 12.4), as well as information for OpenMath content dictionary management. Application-specific metadata elements can be specified by adding corresponding OMDoc modules that extend the content model of the `metadata` element.

The OMDoc`metadata` element can be used to provide information about the document as a whole (as the first child of the `omdoc` element), as well as about specific fragments of the document, and even about the top-level mathematical elements in OMDoc. This reinterpretation of bibliographic metadata as general data about knowledge items allows us to extract document fragments and re-assemble them to new aggregates without losing information about authorship, source, etc.

11.4 Document Comments

Many content markup formats rely on commenting the source for human understanding; in fact source comments are considered a vital part of document markup. However, as XML comments (i.e. anything between "`<!--`" and "`-->`" in a document) need not even be read by some XML parsers, we cannot guarantee that they will survive any XML manipulation of the OMDoc source.

Therefore, anything that would normally go into comments should be modeled with an `omtext` element (`type comment`, if it is a text-level comment; see Section 14.4) or with the `ignore` element for persistent comments, i.e. comments that survive processing. The content of the `ignore` element can be any well-formed OMDoc, it can occur as an OMDoc top-level element or inside mathematical texts (see Chapter 14). This element should be used if the author wants to comment the OMDoc representation, but the end user should not see their content in a final presentation of the document, so that OMDoc text elements are not suitable, e.g. in `ignore`

```
<ignore type="todo" comment="this does not make sense yet, rework">
  <assertion xml:id="heureka">...</assertion>
</ignore>
```

Of course, `ignore` elements can be nested, e.g. if we want to mark up the comment text (a pure string as used in the example above is not enough to express the mathematics). This might lead to markup like

```
<ignore type="todo" comment="rework">
  <ignore type="todo−comment">
    <CMP><xhtml:p>This does not make sense yet, in particular, the equation
      <OMOBJ>...</OMOBJ> cannot be true, think of <OMOBJ>...</OMOBJ>
    </xhtml:p></CMP>
  </ignore>
  <assertion xml:id="heureka">...</assertion>
</ignore>
```

Another good use of the `ignore` element is to use it as an analogon to the in-place error markup in OpenMath objects (see Subsection 13.1.2). In this case, we use the `type` attribute to specify the kind of error and the content for the faulty OMDoc fragment. Note that since the whole object must be a valid OMDoc object (or at least licensed by a DTD or schema), the content itself must be a well-formed OMDoc fragment. As a consequence, the `ignore` element can only be used for "mathematical errors" like sibling `CMP` or `FMP` elements that do not have the same meaning as in Listing 11.1. XML-well-formedness and validity errors will have to be handled by the XML tools involved.

Listing 11.1. Marking up Mathematical Errors Using `ignore`

```
<ignore type="CMP−lang−error"
        comment="multilingual CMPs are not translations of each other">
  <assertion xml:id="ass1">
    <CMP><xhtml:p>The proof is trivial.</xhtml:p></CMP>
    <CMP xml:lang="de"><xhtml:p>Der Beweis ist extrem schwer</xhtml:p></CMP>
  </assertion>
</ignore>
```

For another use of the `ignore` element, see Figure 10.4 in Section 10.4.

11.5 Document Structure

omgroup

Like other documents mathematical ones are often divided into units like chapters, sections, and paragraphs by tags and nesting information. OMDoc makes these document relations explicit by using the `omgroup` element with an optional attribute `type`. It can take the values[2]

`sequence` for a succession of paragraphs. This is the default, and the normal way narrative texts are built up from paragraphs, mathematical statements, figures, etc. Thus, if no `type` is given the type `sequence` is assumed.

`itemize` for unordered lists. The children of this type of `omgroup` will usually be presented to the user as indented paragraphs preceded by a bullet symbol. Since the choice of this symbol is purely presentational, OMDoc use the CSS `style` or `class` attributes on the children to specify the presentation of the bullet symbols (see Section 10.3).

`enumeration` for ordered lists. The children of this type of `omgroup` are usually presented like unordered lists, only that they are preceded by a running number of some kind (e.g. "1.", "2."… or "a)", "b)"…; again the `style` or `class` attributes apply).

`sectioning` The children of this type of `omgroup` will be interpreted as sections. This means that the children will be usually numbered hierarchically, and their metadata will be interpreted as section heading information. For instance the `metadata/dc:title` information (see Section 12.2 for details) will be used as the section title. Note that OMDoc does not provide direct markup for particular hierarchical levels like "chapter", "section", or "paragraph", but assumes that these are determined by the application that presents the content to the human or specified using the CSS attributes.

[2] Version 1.1 of OMDoc also allowed values `dataset` and `labeled-dataset` for marking up tables. These values are deprecated in Version 1.2 of OMDoc, since we provide tables in module RT; see Section **??** for details. Furthermore, Version 1.1 of OMDoc allowed the value `narrative`, which was synonymous with `sequence`.

Other values for the `type` attribute are also admissible, they should be URI references to documents explaining their intension.

We consider the `omdoc` element as an implicit `omgroup`, in order to allow plugging together the content of different OMDoc documents as `omgroup`s in a larger document. Therefore, all the attributes of the `omdoc` element also appear on `omgroup` elements and behave exactly like those.

12

Metadata (Modules DC and CC)

Metadata is "data about data" — in the case of OMDoc data about documents, such as titles, authorship, language usage, or administrative aspects like modification dates, distribution rights, and identifiers. To accommodate such data, OMDoc offers the `metadata` element in many places. The most commonly used metadata standard is the Dublin Core vocabulary, which is supported in some form by most formats. OMDoc uses this vocabulary for compatibility with other metadata applications and extends it for document management purposes in OMDoc. Most importantly OMDoc extends the use of metadata from documents to other (even mathematical) elements and document fragments to ensure a fine-grained authorship and rights management.

12.1 General Metadata

OMDoc 1.3 already integrates the metadata framework for OMDoc 2 based on the recently stabilized RDFa [Adi+08] a standard for flexibly embedding metadata into X(HT)ML documents. This design decision allows us to separate the *syntax* (which is standardized in RDFa) from the *semantics*, which we externalize in metadata ontologies, which can be encoded in OMDoc.

Given the need to incorporate additional metadata into OMDoc, and considering the deficiencies of the metadata support in OMDoc 1.2, we developed a new framework. The requirements were as follows:

1. Stay backwards-compatible with OMDoc 1.2 concerning expressivity. That is, continue supporting Dublin Core and Creative Commons, and the custom extensions.
2. Expose the formal semantics of metadata vocabularies to OMDoc-based applications; additionally be compatible to semantic web applications.
3. Incorporate a vocabulary for versioning – particularly aiming at technical specifications.

4. Do not hard-code a fixed set of vocabularies into the language but stay flexible and extensible for many applications, including future and unknown ones.

Given the fact that many existing metadata vocabularies, including Dublin Core and Creative Commons, have an RDF semantics, and that with RDFa a standard for flexibly embedding metadata into XML had recently stabilized, we chose to incorporate RDFa into OMDoc, and to look for metadata vocabularies with RDF-based implementations to satisfy our further requirements.

So far, RDFa has only been specified for the host languages XHTML [Adi+08]. The specification is generally biased towards XHTML but nevertheless foresees a future adoption of RDFa as an annotation sublanguage by other XML languages. The vector graphics format SVG Tiny already includes RDFa in the same way as XHTML, referring to the XHTML +RDFa specification but making a few minor deviations from it. Other languages are starting to adopt RDFa as well [IL10].

Full RDFa *in* OMDoc

After initial discussions on how much of RDFa to incorporate into OMDoc, we decided to give authors who want to model complex annotations freedom to use the full expressivity of RDFa, but to particularly recommend a metadata syntax that resembles the one of OMDoc 1.2 and allows for expressing most metadata that could also be expressed there. The other reason for fully integrating RDFa is compatibility to RDFa tools. When publishing the sources of OMDoc documents on the web, linked data crawlers like Sindice [TDO07] may find them. While they would not be able to make any sense of OMDoc's own XML vocabulary (e. g. understanding that a *proof* element denotes an instance of the *oo:Proof* class), they would at least be able to understand the annotations made in RDFa, and thus enable users to search for, e. g., OMDoc resources having the *dc:creator* Michael Kohlhase.

A full integration of RDFa means that the following attribute have to be added to OMDoc, with the same semantics as specified for XHTML +RDFa (quoted from [Adi+08]; technical terms explained below):

`rel` a whitespace-separated list of CURIEs, used for expressing relationships between two resources (*predicates in RDF terminology);
`rev` a whitespace separated list of CURIEs, used for expressing reverse relationships between two resources (also *predicates);
`content` a string, for supplying machine-readable content for a literal (a *plain literal object, in RDF terminology);
[XHTML-specific attributes omitted]
`about` a URI or safe CURIE, used for stating what the data is about (a *subject in RDF terminology);

`property` a whitespace separated list of CURIEs, used for expressing relationships between a subject and some literal text (also a *predicate);

`resource` a URI or safe CURIE for expressing the partner resource of a relationship that is not intended to be *clickable (also an *object);

`datatype` a CURIE representing a datatype, to express the datatype of a literal;

`typeof` a whitespace separated list of CURIEs that indicate the RDF type(s) to associate with a subject.

A CURIE (Compact URI, specified as a part of RDFa, but also in a specification of its own [BM09]) is a way of abbreviating a URI as *namespace:localname*, but in contrast to XML local names, the local name definition of SparQL [PS08] is used, which is more liberal, e. g. permitting leading digits. As in SparQL, the underscore prefix is reserved for blank nodes, such as *_:bnode-id*, and names in the default namespace are written with an empty prefix, i. e. as *:localname*. However, the latter namespace is *not* intended to be the default namespace declared in the surrounding XML, but a fixed namespace specified for the language. In addition to that, CURIEs also allow for completely unprefixed names, such as *localname*, which can be reserved words whose mapping to URIs is specified as a part of the language specification. The mappings to URIs for the default namespace and for unprefixed names have been specified for RDFa in XHTML, but as there is currently no standard way of declaring these mappings for a different host language, e. g. in its XML schema, we do not anticipate that any RDFa-aware software would be able to interpret such CURIEs. Therefore, we leave the specification of how OMDoc should handle such CURIEs as future work. Some RDFa attributes allow URIs and CURIEs, which are generally hard to distinguish.[1] Therefore, a CURIE in such an attribute has to be surrounded by square brackets. This syntax is called safe CURIE.

Also note that full RDFa compatibility leads to a syntactical redundancy in all OMDoc elements that carry metadata. In OMDoc 1.2, it was clear (by the human-readable specification, not necessarily for machines!) that metadata contained in an XML element E referred to the concept denoted by E, e. g., that the *dc:title* in listing **??** is the title of the proof with the URI `#fermat-proof`. RDFa requires the subject of annotations to be set explicitly, using the `about`ttribute:

```
<proof xml:id="fermat−proof" about="#fermat−proof">
  <metadata>
     ...
  </metadata>
</proof>
```

Otherwise the parent subject would be reused, which is initially the base URI, i. e. , unless specified otherwise, the URI of the whole document – which

[1] The incoherent use of URIs vs. CURIEs in the RDFa attributes is likely to change in future versions [Bir09].

may, of course, contain many other metadata records. RDFa in XHTML is often used for talking about different things than the elements of the XHTML document itself, such as the book described in a paragraph of the document, except for annotations on the top level for expressing, e. g., the document's author and license. In contrast, metadata in OMDoc are always intended to be annotations for the things modeled in the document, such as theories or statements. It is recommended for all of these things to have a URI, which is defined by the `xml:id`ttribute.[2]

It would be tempting to specify that, for elements that have metadata and an `xml:id` the RDFa subject of the metadata annotations implicitly gets set to the URI of the respective element. One could even specify that, if an element carrying metadata does not have an `xml:id` a blank node will be generated for it. However, XHTML is – and will always be – much more widespread than OMDoc, RDFa has first been designed for annotating XHTML and is still currently biased towards XHTML, and RDFa-aware software will probably not be able to handle custom reinterpretations of the RDFa syntax and semantics soon, at least not as long as there is no way of specifying them in a machine-understandable way[3]. Now suppose we had an OMDoc document at an URI U containing a proof with RDFa metadata but without an explicit `about`ttribute. Suppose the relation of the proof to the theorem it proves were, for some reason, not modeled in OMDoc syntax, but in RDFa, using the OMDoc ontology, i. e. as `<link rel="oo:proves" resource="#fermats-last-theorem">`, which is perfectly legal. An RDFa crawler not knowing OMDoc would extract the triple `<U> oo:proves <#fermats-last-theorem>` from that annotation. From the domain of the *oo:proves* property, any RDFS reasoner would then infer that U is an instance of *oo:Proof*, which is clearly not the case; actually, this would even lead to a contradiction for an Owlreasoner, as *oo:Proof* is disjoint with *oo:Document*, of which U actually is an instance.

Realizing that the web should not be polluted with such invalid RDF triples[4], we therefore specify that RDFa metadata in OMDoc must only be used together with correctly placed `about`ttributes. A relaxation of this policy is subject to future additions to the RDFa specification that might allow for defining parsing rules specific to particular host languages.

Recommended Syntax for RDFa *Metadata*

I will not cover full RDFa in further detail here; for an introduction, see [Her+13; HHA08]. Instead, I will continue with the recommended syntax for using metadata: We introduce the elements *meta* and *link* as children

[2] The MMT URIs of OMDoc 1.6 will enable additional ways of giving URIs to OMDoc concepts, but from an RDFa point of view the principle remains the same.

[3]

[4] See also the Pedantic Web initiative [HC09].

of any *metadata* block.[5,6] Their semantics is roughly inspired by the namesake elements that can occur in the *head* of an XHTML document: *meta* is a literal-valued metadata field, whereas *link* points to another resource by referring to its URI. Resources with document-local identifiers only, i. e. *blank nodes*, can be created using the *resource* element. The elements are shown in table 12.1; an example for using them is given in listing 12.1.

Element	Attributes	Children
meta	`property` `content` `datatype`	literal text or XML (optional)
link	`rel` `rev` `resource`	(*resource*—*meta*—*link*)*
resource	`about` `typeof`	(*meta*—*link*)*

Table 12.1. Elements of the recommended RDFa syntax for OMDoc metadata

Relevant Metadata Vocabularies

Due to the inherent flexibility of RDFa, any metadata vocabulary can be used. However, we give particular recommendations for metadata in the above-mentioned domains of special interest. Using Dublin Core and Creative Commons metadata with the new RDFa syntax for OMDoc is largely trivial. Concerning Dublin Core, we recommend using the more modern DCMI terms vocabulary instead of the DCMES, which is now possible by way of a simple namespace declaration. While the MARC roles had been used as annotations of triples with the *dc:contributor* property in OMDoc 1.2, there is a specification of how to use them in RDF, defining them as sub-properties of *dc:contributor* [Joh05]. Most Creative Commons license declarations will become much easier than in OMDoc 1.2, as we will follow the more recently recommended practice of not always constructing licenses from scratch, but directly linking resources to *existing* Creative Commons licenses using the *xhv:license* property[7]; for example <link rel="xhv:license" resource="http://creativecommons.org/licenses/by/3.0/de/">. It should also be noted that the OMDoc 1.2 syntax allowed for constructing licenses that contradicted the ccREL ontology. For example, it was pos-

[5] Actually, the *link* element has existed before, as a part of OMDoc's rich text (RT) module [Koh06b, section 14.6]. However, this usage does not conflict with its usage as a *metadata* child.

[6] Note that the *metadata* element does not exist for RDFa processors, as it does not carry any RDFa attributes. It is merely a means of structuring the OMDoc syntax.

[7] This property from the XHTML vocabulary supersedes the former *cc:license* property [Abe+08]. By the implementation of the ccREL ontology, this property is also a subproperty of *dc:license*, which in turn is a subproperty of *dc:rights*.

sible to say `<cc:permission derivative_works="prohibited">`, although *cc:DerivativeWorks* is not in the range of the property *cc:prohibits*.[8]

The OMDoc 1.2 Dublin Core extensions for revision logs were not immediately RDF-compatible. We were able to partly replace them by the revisioning vocabulary of DCMI terms. Listing 12.1 shows the proof of Fermat's last theorem once more, now redone using RDFa metadata, and using DCMI terms for the revision history. Comparing this to listing **??**, particularly note the following features:

- We are able to link to resources, such as FOAFprofiles, that describe people (creators, contributors, etc.) in further detail.
- More than one predicate can be given per subject and objects. This makes it convenient to say that a person is both an editor and a publisher of a document.[9]
- The complete revision history can be embedded into the document.
- Versions (or persons, or licenses) can also be described (as blank nodes) if they are only known in this document, i. e. are not globally identifiable by a URI.
- The DCMI Terms vocabulary allows for modeling the history of revisions more faithfully than the Dublin Core extensions of OMDoc 1.2. We can use more specific subproperties of *dct:date*, such as *dct:created* or *dct:issued*. Date can be made really explicit to automated parsers by declaring a datatype for them; otherwise the parser would have to know that *dct:date* and its subproperties usually have an ISO 8601 date value [BM04], or it would have to apply heuristics. Successive revisions can be modeled as a linked list via *dct:replaces*, in addition to referring to them by *dct:hasVersion*. We did not model Michael Kohlhase's digitalization of Wiles's proof as such a replacement, but as a resource that is based on Wiles's proof via the *dct:requires* and *dct:source* properties.
- The license of this document is a ready-to-use Creative Commons license that can simply be referenced by its URI. Alternatively, we can construct it in place.

Compared to OMDoc 1.2, one aspect cannot be expressed with DCMI Terms: the actions that lead to new revisions. One state-of-the-art ontology that offers the desired expressivity is the Ontology Metadata Vocabulary [Har+; Pal+09] for describing ontologies. Instances of *omv:Ontology*

---

8 Given that semantic web reasoning usually assumes an open world, one cannot easily conclude from the *absence* of the *permission* to create derivative works that it is prohibited [Her+08]. Therefore, it is unclear whether one can effectively prohibit derivative works using the ccREL vocabulary. This Orwellian approach to restricting thinking about illiberal licenses by restricting language (cf. [Orw49]) may be debatable, but the ccREL ontology currently specifies it like this, so we have to accept it for the sake of compatibility, or – eventually – model our own licensing ontology that extends ccREL.

9 *marcrel:AUT* is only a subproperty of *dc:contributor*.

Listing 12.1. Proof of Fermat's last theorem, with OMDoc's new RDFa metadata

```
<proof xml:id="fermat-proof" about="#fermat-proof" for="#fermats-last-theorem"
 xmlns:dct="http://purl.org/dc/terms/"
 xmlns:marcrel="http://www.loc.gov/loc.terms/relators/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
 xmlns:xhv="http://www.w3.org/1999/xhtml/vocab#"
 xmlns:cc="http://creativecommons.org/ns#">
  <metadata>
    <meta property="dct:title">Proof of Fermat's Last Theorem</meta>
    <link rel="dct:creator" resource="http://dbpedia.org/resource/Pierre_de_Fermat"/>
    <link rel="marcrel:AUT" resource="http://math.princeton.edu/~awiles/foaf.rdf#me"/>
    <link rel="marcrel:EDT dct:publisher"
     resource="http://kwarc.info/kohlhase/"/>
    <link rel="dct:hasVersion">
      <resource about="[_:initial]">
        <!-- Anonymous resource (bnode). We could also point to a URI by which
             the previous version can actually be retrieved from a repository -->
        <link rel="dct:creator"
         resource="http://dbpedia.org/resource/Pierre_de_Fermat"/>
        <meta property="dct:created" datatype="xsd:date">1637-06-13T00:00:00</meta>
      </resource>
      <resource about="[_:correct]">
        <link rel="dct:replaces" resource="[ _:initial ]"/>
        <link rel="dct:creator"
         resource="http://math.princeton.edu/~awiles/foaf.rdf#me"/>
        <meta property="dct:date" datatype="xsd:date">1995-05-01T00:00:00</meta>
      </resource>
      <resource about="[_:digitalized]">
        <link rel="dct:requires dct:source"
         resource="[_:correct]"/>
        <link rel="dct:creator"
         resource="http://kwarc.info/kohlhase/"/>
        <meta property="dct:issued" datatype="xsd:date">2006-08-28T00:00:00</meta>
      </resource>
    </link>
    <link rel="xhv:license"><!-- actually recommended: directly using
     the pre-defined license http://creativecommons.org/licenses/by/3.0/de/,
     which is the same as what we are constructing here -->
      <meta property="cc:jurisdiction" content="de"/>
      <link rel="cc:permits">
        <resource about="[cc:Reproduction]"/>
        <resource about="[cc:Distribution]"/>
        <resource about="[cc:DerivativeWorks]"/>
      </link>
      <link rel="cc:requires">
        <resource about="[cc:Notice]"/>
        <resource about="[cc:Attribution]"/>
      </link>
    </link>
  </metadata>
  <!-- The actual body of the proof -->
</proof>
```

can be arranged into a list linked via *omv:hasPriorVersion*. As an overlay list to the mere sequence of revisions, a sequence of changes can be given. An *omv:ChangeSpecification* connects two ontology versions by its properties *omv:changeFromVersion* and *omv:changeToVersion* and consists of a set of one or more *omv:Change*s chained together by *omv:hasPreviousChange*. A change has an author (an *omv:Person*), a date, and a few more properties. OMV offers a lot of change subclasses specific to RDFS and OWLontologies; we could easily add change types for mathematical documents, theories, or statements, e. g. a change type for adding a type declaration to a symbol.

```
<!-- TODO: THIS IS OBSOLETE; I WILL REWORK IT INTO AN EXAMPLE USING OMV -->
<link rel="rev:created_by_act" href="[_:creation]"/>
<link rel="rev:current_version" href="[_:current]"/>
<link rel="rev:has_version">
  <resource about="[_:v1]" typeof="rev:Revision">
    <link rel="rev:content" href="fermats-last-theorem?rev=1"/>
    <link rel="rev:created_by_act">
      <resource about="[_:creation]" typeof="chg:Creation">
        <link rel="event:agent" href="http://dbpedia.org/page/Pierre_de_Fermat"/>
        <dc:date>1637-06-13T00:00:00</dc:date>
      </resource>
    </link>
  </resource>
</link>
<!-- revision 2 (Wiles's proof) left out to save space -->
<link rel="rev:has_version">
  <resource about="[_:current]" typeof="rev:Revision">
    <link rel="rev:content" href="fermats-last-theorem?rev=3"/>
    <link rel="rev:created_by_act">
      <resource typeof="chg:Import">
        <link rel="event:agent" href="http://kwarc.info/kohlhase/foaf.rdf#me"/>
        <dc:date>2006-08-28T00:00:00</dc:date>
        <link rel="rev:prior_version" href="[_:v2]"/>
      </resource>
    </link>
  </resource>
</link>
```

Pragmatic Metadata

As the listing in Sect. **??** shows, the new RDFA-based metadata syntax is much more verbose than the old one of OMDOC 1.2. Therefore, we suggest two ways of facilitating the annotation: For manual authoring, one can keep the old, pragmatic OMDOC 1.2 syntax and specify a transformation of such annotations to the new, strict RDFA syntax – implementable, e. g., in XSLT.

also consider STEX as an even more pragmatic metadata syntax .

Respecifying Metadata Inheritance

As I modeled our metadata ontologies in OMDOC, I am now able to extend it by a formal specification of certain rules that had only informally been stated in the OMDOC 1.2 specification: for example, that most DC metadata propagate from document sections down into subsections unless subsections specify different values, or that any *dc:creator* of a subsection of a document becomes a *dc:contributor* to the whole document.

12.2 The Dublin Core Elements (Module DC)

In the following we will describe the variant of Dublin Core metadata elements used in OMDoc[10]. Here, the `metadata` element can contain any number of instances of any Dublin Core elements described below in any order. In fact, multiple instances of the same element type (multiple `dc:creator` elements for example) can be interspersed with other elements without change of meaning. OMDoc extends the Dublin Core framework with a set of roles (from the MARC relator set [MR03]) on the authorship elements and with a rights management system based on the Creative Commons Initiative.

Element	Attributes		Content
	Req.	Optional	
dc:creator		xml:id, class, style, role, type, scheme	text
dc:contributor hline dc:title		xml:id, class, style, role, type, scheme xml:lang, type, scheme	text ⟪*math vernacular*⟫
dc:subject		xml:lang, type, scheme	⟪*math vernacular*⟫
dc:description		xml:lang, type, scheme	⟪*math vernacular*⟫
dc:publisher		xml:id, class, style, type, scheme	ANY
dc:date		action, who, type, scheme	ISO 8601
dc:type		type, scheme	fixed: "Dataset" or "Text"
dc:format		type, scheme	fixed: "application/omdoc+xml"
dc:identifier		type,scheme	ANY
dc:source		type, scheme	ANY
dc:language		type, scheme	ISO 639
dc:relation		type, scheme	ANY
dc:rights		type, scheme	ANY
for ⟪*math vernacular*⟫ see Section 14.1			

Fig. 12.1. Dublin Core Metadata in OMDoc

The descriptions in this section are adapted from [DUB03a], and augmented for the application in OMDoc where necessary. All these elements live in the Dublin Core namespace `http://purl.org/dc/elements/1.1/`, for which we traditionally use the namespace prefix `dc:`.

`dc:title` The title of the element — note that OMDoc metadata can be specified at multiple levels, not only at the document level, in particular, the Dublin Core `dc:title` element can be given to assign a title to a theorem, e.g. the "Substitution Value Theorem". `dc:title`

The `dc:title` element can contain mathematical vernacular, i.e. the same content as the `CMP` defined in Section 14.1. Also like the `CMP` element, the `dc:title` element has an `dc:lang` attribute that specifies the language of the content. Multiple `dc:title` elements inside a `metadata` element are assumed to be translations of each other.

[10] Note that OMDoc1.2 systematically changes the Dublin Core XML tags to synchronize with the tag syntax recommended by the Dublin Core Initiative. The tags were capitalized in OMDoc1.1

dc:creator

dc:creator A primary creator or author of the publication. Additional contributors whose contributions are secondary to those listed in dc:creator elements should be named in dc:contributor elements. Documents with multiple co-authors should provide multiple dc:creator elements, each containing one author. The order of dc:creator elements is presumed to define the order in which the creators' names should be presented.
As markup for names across cultures is still un-standardized, OMDoc recommends that the content of a dc:creator element consists in a single name (as it would be presented to the user). The dc:creator element has an optional attribute dc:id so that it can be cross-referenced and a role attribute to further classify the concrete contribution to the element. We will discuss its values in Section 12.3.

dc:contributor

dc:contributor A party whose contribution to the publication is secondary to those named in dc:creator elements. Apart from the significance of contribution, the semantics of the dc:contributor is identical to that of dc:creator, it has the same restriction content and carries the same attributes plus a dc:lang attribute that specifies the target language in case the contribution is a translation.

dc:subject

dc:subject This element contains an arbitrary phrase or keyword, the attribute dc:lang is used for the language. Multiple instances of the dc:subject element are supported per dc:lang for multiple keywords.

dc:description

dc:description A text describing the containing element's content; the attribute dc:lang is used for the language. As description of mathematical objects or OMDoc fragments may contain formulae, the content of this element is of the form "mathematical text" described in Chapter 14. The dc:description element is only recommended for omdoc elements that do not have a CMP group (see Section 14.1), or if the description is significantly shorter than the one in the CMPs (then it can be used as an abstract).

dc:publisher

dc:publisher The entity for making the document available in its present form, such as a publishing house, a university department, or a corporate entity. The dc:publisher element only applies if the metadata is a direct child of the root element (omdoc) of a document.

dc:date

dc:date The date and time a certain action was performed on the element that contains this. The content is in the format defined by XML Schema data type dateTime (see [BM04] for a discussion), which is based on the ISO 8601 norm for dates and times.
Concretely, the format is $\langle\!\langle YYYY\rangle\!\rangle$-$\langle\!\langle MM\rangle\!\rangle$-$\langle\!\langle DD\rangle\!\rangle$T$\langle\!\langle hh\rangle\!\rangle$:$\langle\!\langle mm\rangle\!\rangle$:$\langle\!\langle ss\rangle\!\rangle$ where $\langle\!\langle YYYY\rangle\!\rangle$ represents the year, $\langle\!\langle MM\rangle\!\rangle$ the month, and $\langle\!\langle DD\rangle\!\rangle$ the day, preceded by an optional leading "-" sign to indicate a negative number. If the sign is omitted, "+" is assumed. The letter "T" is the date/time separator and $\langle\!\langle hh\rangle\!\rangle$, $\langle\!\langle mm\rangle\!\rangle$, $\langle\!\langle ss\rangle\!\rangle$ represent hour, minutes, and seconds respectively. Additional digits can be used to increase the precision of fractional seconds if desired, i.e the format $\langle\!\langle ss\rangle\!\rangle$.$\langle\!\langle sss\ldots\rangle\!\rangle$ with any number of digits after the decimal point is supported. The dc:date element

has the attributes `action` and `who` to specify who did what: The value of `who` is a reference to a `dc:creator` or `dc:contributor` element and `action` is a keyword for the action undertaken. Recommended values include the short forms `updated`, `created`, `imported`, `frozen`, `review-on`, `normed` with the obvious meanings. Other actions may be specified by URIs pointing to documents that explain the action.

`dc:type` Dublin Core defines a vocabulary for the document types in [DUB03b]. The best fit values for OMDoc are

`Dataset` defined as "*information encoded in a defined structure (for example lists, tables, and databases), intended to be useful for direct machine processing*."

`Text` defined as "*a resource whose content is primarily words for reading. For example – books, letters, dissertations, poems, newspapers, articles, archives of mailing lists. Note that facsimiles or images of texts are still of the genre text.*"

`Collection` defined as "*an aggregation of items. The term collection means that the resource is described as a group; its parts may be separately described and navigated*".

The more appropriate should be selected for the element that contains the `dc:type`. If it consists mainly of formal mathematical formulae, then `Dataset` is better, if it is mainly given as text, then `Text` should be used. More specifically, in OMDoc the value `Dataset` signals that the order of children in the parent of the `metadata` is not relevant to the meaning. This is the case for instance in formal developments of mathematical theories, such as the specifications in Chapter 18. `dc:type`

`dc:format` The physical or digital manifestation of the resource. Dublin Core suggests using MIME types [FB96]. Following [MSLK01] we fix the content of the `dc:format` element to be the string `application/omdoc+xml` as the MIME type for OMDoc. `dc:format`

`dc:identifier` A string or number used to uniquely identify the element. The `dc:identifier` element should only be used for public identifiers like ISBN or ISSN numbers. The numbering scheme can be specified in the `scheme` attribute. `dc:identifier`

`dc:source` Information regarding a prior resource from which the publication was derived. We recommend using either a URI or a scientific reference including identifiers like ISBN numbers for the content of the `dc:source` element. `dc:source`

`dc:relation` Relation of this document to others. The content model of the `dc:relation` element is not specified in the OMDoc format. `dc:relation`

`dc:language` If there is a primary language of the document or element, this can be specified here. The content of the `dc:language` element must be an ISO 639 norm two-letter language specifier, like `en` $\widehat{=}$ English, `de` $\widehat{=}$ German, `fr` $\widehat{=}$ French, `nl` $\widehat{=}$ Dutch, ... `dc:language`

`dc:rights` Information about rights held in and over the document or element content or a reference to such a statement. Typically, a `dc:rights` `dc:rights`

element will contain a rights management statement, or reference a service providing such information. `dc:rights` information often encompasses Intellectual Property rights (IPR), Copyright, and various other property rights. If the `dc:rights` element is absent (and no `dc:rights` information is inherited), no assumptions can be made about the status of these and other rights with respect to the document or element.
OMDoc supplies specialized elements for the Creative Commons licenses to support the sharing of mathematical content. We will discuss them in Section 12.4.

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

12.3 Roles in Dublin Core Elements

Because the Dublin Core metadata fields for `dc:creator` and `dc:contributor` do not distinguish roles of specific parties (such as author, editor, and illustrator), we will follow the Open eBook specification [Gro99] and use an optional `role` attribute for this purpose, which is adapted for OMDoc from the MARC relator code list [MR03].

`aut` (author) Use for a person or corporate body chiefly responsible for the intellectual content of an element. This term may also be used when more than one person or body bears such responsibility.
`ant` (scientific/bibliographic antecedent) Use for the author responsible for a work upon which the element is based.
`clb` (collaborator) Use for a person or corporate body that takes a limited part in the elaboration of a work of another author or that brings complements (e.g., appendices, notes) to the work of another author.
`edt` (editor) Use for a person who prepares a document not primarily his/her own for publication, such as by elucidating text, adding introductory or other critical matter, or technically directing an editorial staff.
`ths` (thesis advisor) Use for the person under whose supervision a degree candidate develops and presents a thesis, memoir, or text of a dissertation.
`trc` (transcriber) Use for a person who prepares a handwritten or typewritten copy from original material, including from dictated or orally recorded material. This is also the role (on the `dc:creator` element) for someone who prepares the OMDoc version of some mathematical content.
`trl` (translator) Use for a person who renders a text from one language into another, or from an older form of a language into the modern form. The target language can be specified by `dc:lang`.

As OMDoc documents are often used to formalize existing mathematical texts for use in mechanized reasoning and computation systems, it is sometimes subtle to specify authorship. We will discuss some typical examples to give a guiding intuition. Listing 12.2 shows metadata for a situation where editor R gives the sources (e.g. in LaTeX) of an element written by author A to secretary S for conversion into OMDoc format.

Listing 12.2. A Document with Editor (`edt`) and Transcriber (`trc`)

```
<metadata>
  <dc:title>The Joy of Jordan C* Triples</dc:title>
  <dc:creator role="aut">A</dc:creator>
  <dc:contributor role="edt">R</dc:contributor>
  <dc:contributor role="trc">S</dc:contributor>
</metadata>
```

In Listing 12.3 researcher R formalizes the theory of natural numbers following the standard textbook B (written by author A). In this case we recommend the first declaration for the whole document and the second one for specific math elements, e.g. a definition inspired by or adapted from one in book B.

Listing 12.3. A Formalization with Scientific Antecedent (`ant`)

```
<omdoc xml:id="NNat" version="1.3" xmlns:dc="http://purl.org/dc/elements/1.1/">
  <metadata><dc:title>Natural Numbers</dc:title></metadata>
  ...
  <theory xml:id="NNat.thy">
    <metadata>
      <dc:title>Natural Numbers</dc:title>
      <dc:creator role="aut">R</dc:creator>
      <dc:contributor role="ant">A</dc:contributor>
      <dc:source>B</dc:source>
    </metadata>
    ...
  </theory>
  ...
</omdoc>
```

12.4 Managing Rights by Creative Commons Licenses (Module CC)

The Dublin Core vocabulary provides the `dc:rights` element for information about rights held in and over the document or element content, but leaves the content model unspecified. While it is legally sufficient to describe this information in natural language, a content markup format like OMDoc should support a machine-understandable format. As one of the purposes of the OMDoc format is to support the sharing and re-use of mathematical content, OMDoc provides markup for rights management via the Creative Commons (CC) licenses. Digital rights management (DRM) and licensing of intellectual property has become a hotly debated topic in the last years. We feel that the Creative Commons licenses that encourage sharing of content and enhance the (scientific) public domain while giving authors some control over their intellectual property establish a good middle ground. Specifying rights is important, since in the absence of an explicit or implicit (via inheritance) `dc:rights` element no assumptions can be made about the status of the document or fragment. Therefore OMDoc adds another child to the

cc:license

`metadata` element. This `cc:license` element is a symbolic representation of the Creative Commons legal framework, adapted to the OMDoc setting: The Creative Commons Metadata Initiative specifies various ways of embedding CC metadata into documents and electronic artefacts like pictures or MP3 recordings. As OMDoc is a source format, from which various presentation formats are generated, we need a content representation of the CC metadata from which the end-user representations for the respective formats can be generated.

Element	Attributes		Content
	Req.	Optional	
cc:license		jurisdiction	permissions, prohibitions, requirements
cc:permissions		reproduction, distribution, derivative_works	EMPTY
cc:prohibitions		commercial_use	EMPTY
cc:requirements		notice, copyleft, attribution	EMPTY

Fig. 12.2. The OMDoc Elements for Creative Commons Metadata

The Creative Commons Metadata Initiative [Cre] divides the license characteristics in three types: **permissions**, **prohibitions** and **requirements**, which are represented by the three elements, which can occur as children of the `cc:license` element. The `cc:license` element has two optional argument:

`jurisdiction` which allows to specify the country in whose jurisdiction the license will be enforced[11]. It's value is one of the top-level domain codes of the "Internet Assigned Names Authority (IANA)" [Ian]. If this attribute is absent, then the original US version of the license is assumed.

`version` which allows to specify the version of the license. If the attribute is not present, then the newest released version is assumed (version 2.0 at the time of writing this book)

The following three empty elements can occur as children of the `cc:license` element; their attribute specify the rights bestowed on the user by the license.

[11] The Creative Commons Initiative is currently in the process of adapting their licenses to jurisdictions other than the USA, where the licenses originated. See [Urlb] for details and to check for progress.

All these elements have the namespace `http://creativecommons.org/ns`, for which we traditionally use the namespace prefix `cc:`.

- `cc:permissions` are the rights granted by the license, to model them the element has three attributes, which can have the values `permitted` (the permission is granted by the license) and `prohibited` (the permission isn't): `cc:permissions`

Attribute	Permission	Default
`reproduction`	the work may be reproduced	`permitted`
`distribution`	the work may be distributed, publicly displayed, and publicly performed	`permitted`
`derivative_works`	derivative works may be created and reproduced	`permitted`

- `cc:prohibitions` are the things the license prohibits. `cc:prohibitions`

Attribute	Prohibition	Default
`commercial_use`	stating that rights may be exercised for commercial purposes.	`permitted`

- `cc:requirements` are restrictions imposed by the license. `cc:requirements`

Attribute	Requirement	Default
`notice`	copyright and license notices must be kept intact	`required`
`attribution`	credit must be given to copyright holder and/or author	`required`
`copyleft`	derivative works, if authorized, must be licensed under the same terms as the work	`required`

This vocabulary is directly modeled after the Creative Commons Metadata [Urla] which defines the meaning, and provides an RDF [LS99] based implementation. As we have discussed in Section 11.3, OMDoc follows an approach that specifies metadata in the document itself; thus we have provided the elements described here. In contrast to many other situations in OMDoc, the rights model is not extensible, since only the current model is backed by legal licenses provided by the creative commons initiative.

Listing 12.4 specifies a license grant using the Creative Commons "share-alike" license: The copyright is retained by the author, who licenses the content to the world, allowing others to reproduce and distribute it without restrictions as long as the copyright notice is kept intact. Furthermore, it allows others to create derivative works based on the content as long as it attributes the original work of the author and licenses the derived work under the identical license (i.e. the Creative Commons "share-alike" as well).

Listing 12.4. A Creative Commons License

```
<metadata>
  <dc:rights>Copyright (c) 2004 Michael Kohlhase</dc:rights>
  <license jurisdiction="de" xmlns="http://creativecommons.org/ns">
    <permissions reproduction="permitted" distribution="permitted"
                 derivative_works="permitted"/>
    <prohibitions commercial_use="permitted"/>
    <requirements notice="required" copyleft="required" attribution="required"/>
  </license>
</metadata>
```

13

Mathematical Objects (Module MOBJ)

A distinguishing feature of mathematics is its ability to represent and manipulate ideas and objects in symbolic form as mathematical formulae. OMDoc uses the OpenMath and Content-MathML formats to represent mathematical formulae and objects. Therefore, the OpenMath standard [Bus+04] and the MathML 2.0 recommendation (second edition) [Aus+03a] are part of this specification. We will review OpenMath objects (top-level element `om:OMOBJ`) in Section 13.1 and Content-MathML (top-level element `m:math`) in Section 13.2, and specify an OMDoc element for entering mathematical formulae (element `legacy`) in Section 13.5.

Element	Attributes		Content
	Required	Optional	
`OMOBJ`	`id`	`class, style`	See Figure 13.2
`m:math`		`id, xlink:href`	See Figure 13.5
`legacy`	`format`	`xml:id, formalism`	`#PCDATA`

Fig. 13.1. Mathematical Objects in OMDoc

The recapitulation in the next two sections is not normative, please consult Section 2.1 for a general introduction and history and the OpenMath standard and the MathML 2.0 Recommendation for details and clarifications.

13.1 OpenMath

OpenMath is a markup language for mathematical formulae that concentrates on the meaning of formulae building on an extremely simple kernel (markup primitive for syntactical forms of content formulae), and adds an extension mechanism for mathematical concepts, the **content dictionaries**. These are machine-readable documents that define the meaning of mathematical concepts expressed by OpenMath symbols. The current released version of the OpenMath standard is OpenMath2, which incorporates many of the experiences of the last years, particularly with embedding OpenMath into the OMDoc format.

We will only review the XML encoding of OpenMath objects here, since it is most relevant to the OMDoc format. All elements of the XML encoding live in the namespace `http://www.openmath.org/OpenMath`, for which we traditionally use the namespace prefix `om:`.

Element	Attributes		Content
	Required	Optional	
OMOBJ		id, cdbase, class, style	⟪*OMel*⟫?
OMS	cd, name	id, cdbase, class, style	EMPTY
OMV	name	id, class, style	EMPTY
OMA		id, cdbase, class, style	⟪*OMel*⟫*
OMBIND		id, cdbase, class, style	⟪*OMel*⟫,OMBVAR,⟪*OMel*⟫
OMBVAR		id, class, style	(OMV | OMATTR)+
OMFOREIGN		id, cdbase, class, style	ANY
OMATTR		id, cdbase, class, style	⟪*OMel*⟫
OMATP		id, cdbase, class, style	(OMS, (⟪*OMel*⟫|OMFOREIGN))+
OMI		id, class, style	[0-9]*
OMB		id, class, style	#PCDATA
OMF		id, class, style, dec, hex	#PCDATA
OME		id, class, style	⟪*OMel*⟫?
OMR	href		⟪*OMel*⟫?
where ⟪*OMel*⟫ is (OMS|OMV|OMI|OMB|OMSTR|OMF|OMA|OMBIND|OME|OMATTR)			

Fig. 13.2. OpenMath Objects in OMDoc

13.1.1 The Representational Core of OpenMath

om:OMOBJ

om:OMA

om:OMV

om:OMS

The central construct of the OpenMath is that of an **OpenMath object** (represented by the `om:OMOBJ` element in the XML encoding), which has a tree-like representation made up of applications (`om:OMA`), binding structures (`om:OMBIND` using `om:OMBVAR` to tag bound variables), variables (`om:OMV`), and symbols (`om:OMS`).

The `om:OMA` element contains representations of the function and its argument in "prefix-" or "Polish notation", i.e. the first child is the representation of the function and all the subsequent ones are representations of the arguments in order.

Objects and concepts that carry meaning independent of the local context (they are called **symbol**s in OpenMath) are represented as `om:OMS` elements, where the value of the `name` attribute gives the name of the symbol. The `cd` attribute specifies the relevant content dictionary, a document that defines the meaning of a collection of symbols including the one referenced by the `om:OMS`. This document can either be an original OpenMath content dictionary or an OMDoc document that serves as one (see Subsection 15.6.2 for a discussion). The optional `cdbase` on an `om:OMS` element contains a URI that can be used to disambiguate the content dictionary. Alternatively, the `cdbase` attribute can be given on an OpenMath element that is a parent to the `om:OMS` in question: The `om:OMS` inherits the `cdbase` of the nearest ancestor (inducing the usual XML scoping rules for declarations).

The OpenMath2 standard proposes the following mechanism for determining a canonical identifying URI for the symbol declaration referenced by an OpenMath symbol of the form `<OMS cd="foo" name="bar"/>` with the `cdbase`-value e.g. `http://www.openmath.org/cd`: it is the URI reference `http://www.openmath.org/cd/foo#bar`, which by convention identifies an `omcd:CDDefinition` element with a child `omcd:Name` whose value is `bar` in a content dictionary resource `http://www.openmath.org/cd/foo.ocd` (see Subsection 2.1.2 for a very brief introduction to OpenMath content dictionaries).

Variables are represented as `om:OMV` element. As variables do not carry a meaning independent of their local content, `om:OMV` only carries a `name` attribute (see Section 13.4 for further discussion).

For instance, the formula $\sin(x)$ would be modeled as an application of the sin function (which in turn is represented as an OpenMath symbol) to a variable:

```
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA cdbase="http://www.openmath.org/cd">
    <OMS cd="transc1" name="sin"/>
    <OMV name="x"/>
  </OMA>
</OMOBJ>
```

In our case, the function sin is represented as an `om:OMS` element with name `sin` from the content dictionary `transc1`. The `om:OMS` inherits the `cdbase`-value `http://www.openmath.org/cd`, which shows that it comes from the OpenMath standard collection of content dictionaries from the `om:OMA` element above. The variable x is represented in an `om:OMV` element with `name`-value `x`.

For the `om:OMBIND` element consider the following representation of the formula $\forall x.\sin(x) \leq \pi$. `om:OMBIND`

```
<OMOBJ cdbase="http://www.openmath.org/cd">
  <OMBIND>
    <OMS cd="quant1" name="forall"/>
    <OMBVAR><OMV name="x"/></OMBVAR>
    <OMA>
      <OMS cd="arith1" name="leq"/>
      <OMA><OMS cd="transc1" name="sin"/><OMV name="x"/></OMA>
      <OMS cd="nums1" name="pi"/>
    </OMA>
  </OMBIND>
</OMOBJ>
```

The `om:OMBIND` element has exactly three children, the first one is a “binding operator”[1] — in this case the universal quantifier, the second one is a list of

[1] The binding operator must be a symbol which either has the role `binder` assigned by the OpenMath content dictionary (see [Bus+04] for details) or the symbol

om:OMBVAR

bound variables that must be encapsulated in an om:OMBVAR element, and the third is the body of the binding object, in which the bound variables can be used. OpenMath uses the om:OMBIND element to unambiguously specify the scope of bound variables in expressions: the bound variables in the om:OMBVAR element can be used only inside the mother om:OMBIND element, moreover they can be systematically renamed without changing the meaning of the binding expression. As a consequence, bound variables in the scope of an om:OMBIND are distinct as OpenMath objects from any variables outside it, even if they share a name.

om:OMATTR

om:OMATP

OpenMath offers an element for annotating (parts of) formulae with external information (e.g. MathML or LaTeX presentation): the om:OMATTR element that pairs an OpenMath object with an attribute-value list. To annotate an OpenMath object, it is embedded as the second child in an om:OMATTR element. The attribute-value list is specified by children of the preceding om:OMATP (Attribute value Pair) element, which has an even number of children: children at odd positions must be om:OMS (specifying the attribute, they are called **key**s or **feature**s)[2], and children at even positions are the **value**s of the keys specified by their immediately preceding siblings. In the OpenMath fragment in Listing 13.1 the expression $x + \pi$ is annotated with an alternative representation and a color. Listing 13.4 has a more complex one involving types.

Listing 13.1. Associating Alternate Representations with an OpenMath Object

```
<OMATTR>
  <OMATP>
    <OMS cd="alt-rep" name="ascii"/>
    <OMSTR>(x+1)</OMSTR>
    <OMS cd="alt-rep" name="svg"/>
    <OMFOREIGN encoding="application/svg+xml">
      <svg xmlns='http://www.w3.org/2000/svg'>...</svg>
    </OMFOREIGN>
    <OMS cd="pres" name="color"/>
    <OMS cd="pres" name="red"/>
  </OMATP>
  <OMA>
    <OMS cd="arith1" name="plus"/>
    <OMV name="x"/>
    <OMS cd="nums1" name="pi"/>
  </OMA>
</OMATTR>
```

A special application of the om:OMATTR element is associating non-OpenMath objects with OpenMath objects. For this, OpenMath2 allows to use

declaration in the OMDoc content dictionary must have the value binder for the attribute role (see Subsection 15.2.1).

[2] There are two kinds of keys in OpenMath distinguished according to the role value on their symbol declaration in the contentdictionary: attribution specifies that this attribute value pair may be ignored by an application, so it should be used for information which does not change the meaning of the attributed OpenMath object. The role is used for keys that modify the meaning of the attributed OpenMath object and thus cannot be ignored by an application.

an `om:OMFOREIGN` element in the even positions of an `om:OMATP`. This element can be used to hold arbitrary XML content (in our example above SVG: Scalable Vector Graphics [JFF02]), its required `encoding` attribute specifies the format of the content. We recommend a MIME type [FB96] (see Section **??** for an application). `om:OMFOREIGN`

13.1.2 Programming Extensions of OpenMath Objects

For representing objects in computer algebra systems OpenMath also provides other basic data types: `om:OMI` for integers, `om:OMB` for byte arrays, `om:OMSTR` for strings, and `om:OMF` for floating point numbers. These do not play a large role in the context of OMDoc, so we refer the reader to the OpenMath standard [Bus+04] for details. `om:OMI` `om:OMB` `om:OMSTR` `om:OMF`

The `om:OME` element is used for in-place error markup in OpenMath objects, it can be used almost everywhere in OpenMath elements. It has two children; the first one is an error operator[3], i.e. an OpenMath symbol that specifies the kind of error, and the second one is the faulty OpenMath object fragment. Note that since the whole object must be a valid OpenMath object, the second child must be a well-formed OpenMath object fragment. As a consequence, the `om:OME` element can only be used for "semantic errors" like non-existing content dictionaries, out-of-bounds errors, etc. XML-well-formedness and DTD-validity errors will have to be handled by the XML tools involved. In the following example, we have marked up two errors in a faulty representation of $\sin(\pi)$. The outer error flags an arity violation (the function sin only allows one argument), and the inner one flags the typo in the representation of the constant π (we used the name `po` instead of `pi`). `om:OME`

```
<OME>
  <OMS cd="type−error" name="arity−violation"/>
  <OMA>
    <OMS cd="transc1" name="sin"/>
    <OME>
      <OMS cd="error" name="unexpected_symbol"/>
      <OMS cd="nums1" name="po"/>
    </OME>
    <OMV name="x"/>
  </OMA>
</OME>
```

As we can see in this example, errors can be nested to encode multiple faults found by an OpenMath application.

13.1.3 Structure Sharing in OpenMath

As we have seen above, OpenMath objects are essentially trees, where the leaves are symbols or variables. In many applications mathematical objects

[3] An error operator is like a binding operator in footnote 1, only the symbol has role `error`.

can grow to be very large, so that more space-efficient representations are needed. Therefore, OpenMath2 supports structure sharing[4] in OpenMath objects. In Figure 13.3 we have contrasted the tree representation of the object $1+1+1+1+1+1+1+1$ with the structure-shared one, which represents the formula as a directed acyclic graph (DAG). As any DAG can be exploded into a tree by recursively copying all sub-graphs that have more than one incoming graph edge, DAGs can conserve space by structure sharing. In fact the tree on the left in Figure 13.3 is exponentially larger than the corresponding DAG on the right.

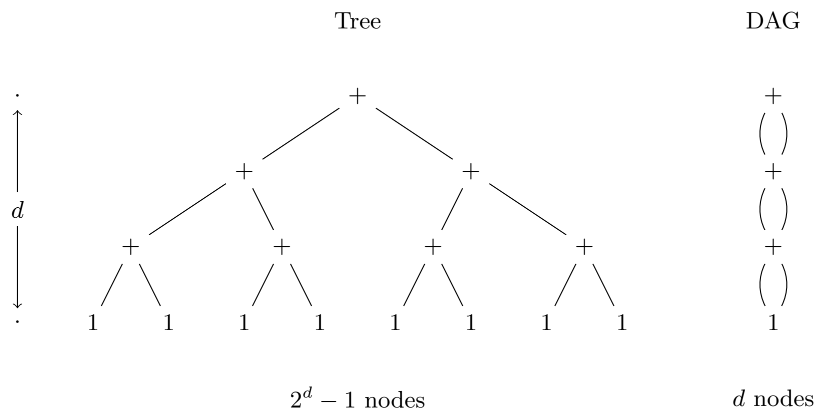


Fig. 13.3. Structure Sharing by Directed Acyclic Graphs

om:OMR

To support DAG structures, OpenMath2 provides the (optional) attribute `id` on all OpenMath objects and an element `om:OMR`[5] for the purpose of cross-referencing. The `om:OMR` element is empty and has the required attribute `href`; The OpenMath element represented by this `om:OMR` element is a copy of the OpenMath element pointed to in the `href` attribute. Note that the representation of the `om:OMR` element is *structurally equal*, but not identical to the element it points to.

Using the `om:OMR` element, we can represent the OpenMath objects in Figure 13.3 as the XML representations in Figure 13.4.

[4] Structure sharing is a well-known technique in computer science that tries to gain space efficiency in algorithms by re-using data structures that have already been created by pointing to them rather than copying.

[5] OpenMath1 and OMDoc 1.0 did now know structure sharing, OMDoc 1.1 added `xref` attributes to the OpenMath elements `om:OMOBJ`, `om:OMA`, `om:OMBIND` and `om:OMATTR` instead of `om:OMR` elements. This usage is deprecated in OMDoc 1.2, in favor of the `om:OMR`-based solution from the OpenMath2 standard. Obviously, both representations are equivalent, and a transformation from `xref`-based mechanism to the `om:OMR`-based one is immediate.

Shared	Exploded
<OMOBJ>	<OMOBJ>
<OMA>	<OMA>
<OMS cd="nat" name="plus"/>	<OMS cd="nat" name="plus"/>
<OMA id="t1">	<OMA>
<OMS cd="nat" name="plus"/>	<OMS cd="nat" name="plus"/>
<OMA id="t11">	<OMA>
<OMS cd="nat" name="plus"/>	<OMS cd="nat" name="plus"/>
<OMI>1</OMI>	<OMI>1</OMI>
<OMI>1</OMI>	<OMI>1</OMI>
</OMA>	</OMA>
<OMR href="#t11"/>	<OMA>
	<OMS cd="nat" name="plus"/>
	<OMI>1</OMI>
	<OMI>1</OMI>
	</OMA>
</OMA>	</OMA>
<OMR href="#t1"/>	<OMA>
	<OMS cd="nat" name="plus"/>
	<OMA>
	<OMS cd="nat" name="plus"/>
	<OMI>1</OMI>
	<OMI>1</OMI>
	</OMA>
	<OMA>
	<OMS cd="nat" name="plus"/>
	<OMI>1</OMI>
	<OMI>1</OMI>
	</OMA>
	</OMA>
</OMA>	</OMA>
</OMOBJ>	</OMOBJ>

Fig. 13.4. The OpenMath Objects from Figure 13.3 in XML Encoding

To ensure that the XML representations actually correspond to directed acyclic graphs, the occurrences of the `om:OMR` must obey the global acyclicity constraint below, where we say that an OpenMath element **dominate**s all its children and all elements they dominate; The `om:OMR` also dominates its **target**[6], i.e. the element that carries the `id` attribute pointed to by the `href` attribute. For instance, in the representation in Figure 13.4 the `om:OMA` element with `xml:id="t1"` and also the second `om:OMA` element dominate the `om:OMA` element with `xml:id="t11"`.

OpenMath Acyclicity Constraint:
An OpenMath element may not dominate itself.

Listing 13.2. A Simple Cycle

```
<OMOBJ>
  <OMA id="foo">
    <OMS cd="nat" name="divide"/>
    <OMI>1</OMI>
    <OMA><OMS cd="nat" name="plus"/>
```

[6] The target of an OpenMath element with an `id` attribute is defined analogously

```
    <OMI>1</OMI>
    <OMR href="#foo"/>
  </OMA>
 </OMA>
</OMOBJ>
```

In Listing 13.2 the `om:OMA` element with `xml:id="foo"` dominates its third child, which dominates the `om:OMR` with `href="foo"`, which dominates its target: the `om:OMA` element with `xml:id="foo"`. So by transitivity, this element dominates itself, and by the acyclicity constraint, it is not the XML representation of an OpenMath object. Even though it could be given the interpretation of the continued fraction

$$\frac{1}{1+\frac{1}{1+\cdots}}$$

this would correspond to an infinite tree of applications, which is not admitted by the OpenMath standard. Note that the acyclicity constraint is not restricted to such simple cases, as the example in Listing 13.3 shows. Here, the `om:OMA` with `xml:id="bar"` dominates its third child, the `om:OMR` element with `href="baz"`, which dominates its target `om:OMA` with `xml:id="baz"`, which in turn dominates its third child, the `om:OMR` with `href="bar"`, this finally dominates its target, the original `om:OMA` element with `xml:id="bar"`. So again, this pair of OpenMath objects violates the acyclicity constraint and is not the XML encoding of an OpenMath object.

Listing 13.3. A Cycle of Order Two

```
<OMOBJ>
 <OMA id="bar">
  <OMS cd="nat" name="plus"/>
  <OMI>1</OMI>
  <OMR href="#baz"/>
 </OMA>
</OMOBJ>
```

```
<OMOBJ>
 <OMA id="baz">
  <OMS cd="nat" name="plus"/>
  <OMI>1</OMI>
  <OMR href="#bar"/>
 </OMA>
</OMOBJ>
```

13.2 Content MathML

Content-MathML is a content markup format that represents the abstract structure of formulae in trees of logical sub-expressions much like OpenMath. However, in contrast to that, Content-MathML provides a lot of primitive tokens and constructor elements for the K-14 fragment of mathematics (Kindergarten to 14^{th} grade (i.e. undergraduate college level)).

The current released version of the MathML recommendation is the second edition of MathML 2.0 [Aus+03a], a maintenance release for the MathML 2.0 recommendation [Aus+03b] that cleans up many semantic issues in the content MathML part. We will now review those parts of MathML 2.0 that are relevant to OMDoc; for the full story see [Aus+03a].

Even though OMDOC allows full Content-MATHML, we will advocate the use of the Content-MATHML fragment described in this section, which is largely isomorphic to OPENMATH (see Subsection 13.2.2 for a discussion).

<table>
<tr><td>Element</td><td colspan="2">Attributes</td><td>Content</td></tr>
<tr><td></td><td>Required</td><td>Optional</td><td></td></tr>
<tr><td>m:math</td><td></td><td>id, xlink:href</td><td>《CMel》+</td></tr>
<tr><td>m:apply</td><td></td><td>id, xlink:href</td><td>m:bvar?,《CMel》*</td></tr>
<tr><td>m:csymbol</td><td>definitionURL</td><td>id, xlink:href</td><td>m:EMPTY</td></tr>
<tr><td>m:ci</td><td></td><td>id, xlink:href</td><td>#PCDATA</td></tr>
<tr><td>m:cn</td><td></td><td>id, xlink:href</td><td>([0-9]|,|.)(*|e([0-9]|,|.)*)?</td></tr>
<tr><td>m:bvar</td><td></td><td>id, xlink:href</td><td>m:ci|m:semantics</td></tr>
<tr><td>m:semantics</td><td></td><td>id, xlink:href, definitionURL</td><td>《CMel》,(m:annotation | m:annotation-xml)*</td></tr>
<tr><td>m:annotation</td><td></td><td>definitionURL, encoding</td><td>#PCDATA</td></tr>
<tr><td>m:annotation-xml</td><td></td><td>definitionURL, encoding</td><td>ANY</td></tr>
<tr><td colspan="4">where 《CMel》 is m:apply|m:csymbol|m:ci|m:cn|m:semantics</td></tr>
</table>

Fig. 13.5. Content-MATHML in OMDOC

13.2.1 The Representational Core of Content-MathML

The top-level element of MATHML is the `m:math`[7] element, see Figure 13.7 for an example. Like OPENMATH, Content-MATHML organizes the mathematical objects into a functional tree. The basic objects (MATHML calls them token elements) are

m:math

identifiers (element `m:ci`) corresponding to variables. The content of the `m:ci` element is arbitrary Presentation-MATHML, used as the name of the identifier.

m:ci

numbers (element `m:cn`) for number expressions. The attribute `type` can be used to specify the mathematical type of the number, e.g. `complex`, `real`, or `integer`. The content of the `m:cn` element is interpreted as the value of the number expression.

m:cn

symbols (element `m:csymbol`) for arbitrary symbols. Their meaning is determined by a `definitionURL` attribute that is a URI reference that points to a symbol declaration in a defining document. The content of the `m:csymbol` element is a Presentation-MATHML representation that used to depict the symbol.

m:csymbol

---

[7] For DTD validation OMDOC uses the namespace prefix "`m:`" for MATHML elements, since the OMDOC DTD needs to include the MATHML DTD with an explicit namespace prefix, as both MATHML and OMDOC have a `selector` element that would clash otherwise (DTDs are not namespace-aware).

Apart from these generic elements, Content-MathML provides a set of about 80 empty content elements that stand for objects, functions, relations, and constructors from various basic mathematic fields.

m:apply

The `m:apply` element does double duty in Content-MathML: it is not only used to mark up applications, but also represents binding structures if it has an `m:bvar` child; see Figure 13.7 below for a use case in a universal quantifier.

m:bvar

m:semantics

The `m:semantics` element provides a way to annotate Content-MathML elements with arbitrary information. The first child of the `m:semantics` element is annotated with the information in the `m:annotation-xml` (for XML-based information) and `m:annotation` (for other information) elements that follow it. These elements carry `definitionURL` attributes that point to a "definition" of the kind of information provided by them. The optional `encoding` is a string that describes the format of the content.

m:annotation-xml

m:annotation

13.2.2 OpenMath vs. Content MathML

OpenMath and MathML are well-integrated; there are semantics-preserving converters between the two formats. MathML supports the `m:semantics` element, that can be used to annotate MathML presentations of mathematical objects with their OpenMath encoding. Analogously, OpenMath supports the `presentation` symbol in the `om:OMATTR` element, that can be used for annotating with MathML presentation. OpenMath is the designated extension mechanism for MathML beyond K-14 mathematics: Any symbol outside can be encoded as a `m:csymbol` element, whose `definitionURL` attribute points to the OpenMath CD that defines the meaning of the symbol. Moreover all of the MathML content elements have counterparts in the OpenMath core content dictionaries [OMCD]. For the purposes of OMDoc, we will consider the various representations following four representations of a content symbol in Figure 13.6 as equivalent. Note that the URI in the `definitionURL` attribute does not point to a specific file, but rather uses its base name for the reference. This allows a MathML (or OMDoc) application to select the format most suitable for it.

In Figure 13.7 we have put the OpenMath and content MathML encoding of the law of commutativity for the real numbers side by side to show the similarities and differences. There is an obvious line-by-line similarity for the tree constructors and token elements. The main difference is the treatment of types and variables.

13.3 Representing Types in Content-MathML and OpenMath

Types are representations of certain simple sets that are treated specially in (human or mechanical) reasoning processes. In typed representations vari-

```
<m:plus/>
```
Content-MathML token element

```
<m:plus definitionURL="http://www.openmath.org/cd/arith1#plus"/>
```
Content-MathML token element with explicit pointer

```
<m:csymbol definitionURL="http://www.openmath.org/cd/arith1#plus"/>
```
empty Content-MathML `m:csymbol`

```
<m:csymbol definitionURL="http://www.openmath.org/cd/arith1#plus">
  <m:mo>+</m:mo>
</m:csymbol>
```
Content-MathML `m:csymbol` with presentation

```
<OMS cdbase="http://www.openmath.org/cd" cd="arith1" name="plus"/>
```
OpenMath symbol

Fig. 13.6. Four equivalent Representations of a Content Symbol

ables and constants are usually associated with types to support more guided reasoning processes. Types are structurally like mathematical objects (i.e. arbitrary complex trees). Since types are ubiquitous in representations of mathematics, we will briefly review the best practices for representing them in OMDoc.

MathML supplies the `type` attribute to specify types that can be taken from an open-ended list of type names. OpenMath uses the `om:OMATTR` element to associate a type (in this case the set of real numbers as specified in the `setname1` content dictionary) with the variable, using the feature symbol `type` from the `sts` content dictionary. This mechanism is much more heavyweight in our special case, but also more expressive: it allows to use arbitrary content expressions for types, which is necessary if we were to assign e.g. the type $(\mathbb{R} \to \mathbb{R}) \to (\mathbb{R} \to \mathbb{R})$ for functionals on the real numbers. In such cases, the second edition of the MathML2 Recommendation advises a construction using the `m:semantics` element (see [KD03b] for details). Listings 13.4 and 13.5 show the realizations of a quantification over a variable of functional type in both formats.

Listing 13.4. A Complex Type in OpenMath

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

OpenMath

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

   </OMATTR>
   <OMATTR>
    <OMATP>
     <OMS cd="sts" name="type"/>
     <OMS cd="setname1" name="R"/>
    </OMATP>

    <OMV name="b"/>
   </OMATTR>
  </OMBVAR>
   <OMA>
    <OMS cd="relation" name="eq"/>
    <OMA>
     <OMS cd="arith1" name="plus"/>
     <OMV name="a"/>
     <OMV name="b"/>
    </OMA>
    <OMA>
     <OMS cd="arith1" name="plus"/>
     <OMV name="b"/>
     <OMV name="a"/>
    </OMA>
  </OMA>
 </OMBIND>
</OMOBJ>
```

MathML

```
<m:math>
 <m:apply>
  <m:forall/>
  <m:bvar>






   <m:ci type="real">a</m:ci>
  </m:bvar>






  <m:bvar>
   <m:ci type="real">b</m:ci>

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

Fig. 13.7. OpenMath vs. C-MathML for Commutativity

```
          </OMA>
        </OMA>
      </OMATP>
      <OMV name="F"/>
    </OMATTR>
  </OMBVAR>
  ...
 </OMBIND>
</OMOBJ>
```

Note that we have essentially used the same URI (to the `sts` content dictionary) to identify the fact that the annotation to the variable is a type (in a particular type system).

Listing 13.5. A Complex Type in Content-MathML

```
<m:math>
  <m:apply>
    <m:forall/>
    <m:bvar>
      <m:semantics>
```

```
        <m:ci>F</m:ci>
        <m:annotation−xml definitionURL="http://www.openmath.org/cd/sts#type">
          <m:apply>
            <m:csymbol definitionURL="http://www.openmath.org/cd/sts#mapsto"/>
            <m:apply>
              <m:csymbol definitionURL="http://www.openmath.org/cd/sts#mapsto"/>
              <m:csymbol definitionURL="http://www.openmath.org/cd/setname1#real"/>
              <m:csymbol definitionURL="http://www.openmath.org/cd/setname1#real"/>
            </m:apply>
            <m:apply>
              <m:csymbol definitionURL="http://www.openmath.org/cd/sts#mapsto"/>
              <m:csymbol definitionURL="http://www.openmath.org/cd/setname1#real"/>
              <m:csymbol definitionURL="http://www.openmath.org/cd/setname1#real"/>
            </m:apply>
          </m:apply>
        </m:annotation−xml>
      </m:semantics>
    </m:bvar>
    . . .
  </m:apply>
</m:math>
```

13.4 The Semantics of Variables in OpenMath and Content-MathML

A more subtle, but nonetheless crucial difference between OpenMath and MathML is the handling of variables, symbols, their names, and equality conditions. OpenMath uses the `name` attribute to identify a variable or symbol, and delegates the presentation of its name to other methods such as style sheets. As a consequence, the elements `om:OMS` and `om:OMV` are empty, and we have to understand the value of the `name` attribute as a pointer to a defining occurrence. In case of symbols, this is the symbol declaration in the content dictionary identified in the `cd` attribute. A symbol <OMS cd="$\langle\!\langle cd_1\rangle\!\rangle$" name="$\langle\!\langle name_1\rangle\!\rangle$"/> is equal to <OMS cd="$\langle\!\langle cd_2\rangle\!\rangle$" name="$\langle\!\langle name_2\rangle\!\rangle$"/>, iff $\langle\!\langle cd_1\rangle\!\rangle=\langle\!\langle cd_2\rangle\!\rangle$ and $\langle\!\langle name_1\rangle\!\rangle=\langle\!\langle name_2\rangle\!\rangle$ as XML simple names. In case of variables this is more difficult: if the variable is bound by an `om:OMBIND` element[8], then we interpret all the variables <OMV name="x"/> in the `om:OMBIND` element as equal and different from any variables <OMV name="x"/> outside. In fact the OpenMath standard states that bound variables can be renamed without changing the object (**α-conversion**). If <OMV name="x"/> is not bound, then the scope of the variable cannot be reliably defined; so equality with other occurrences of the variable <OMV name="x"/> becomes an ill-defined problem. We therefore discourage the use of unbound variables in OMDoc; they are very simple to avoid by using symbols instead, introducing suitable theories if necessary (see Section 15.6).

[8] We say that an `om:OMBIND` element **binds** an OpenMath variable <OMV name="x"/>, iff this `om:OMBIND` element is the nearest one, such that <OMV name="x"/> occurs in (second child of the `om:OMATTR` element in) the `om:OMBVAR` child (this is the **defining occurrence** of <OMV name="x"/> here).

MathML goes a different route: the `m:csymbol` and `m:ci` elements have content that is Presentation-MathML, which is used for the presentation of the variable or symbol name.[9] While this gives us a much better handle on presentation of objects with variables than OpenMath (where we are basically forced to make due with the ASCII[10] representation of the variable name), the question of scope and equality becomes much more difficult: Are two variables (semantically) the same, even if they have different colors, sizes, or font families? Again, for symbols the situation is simpler, since the `definitionURL` attribute on the `m:csymbol` element establishes a global identity criterion (two symbols are equal, iff they have the same `definitionURL` value (as URI strings; see [BLFM98]).) The second edition of the MathML standard adopts the same solution for bound variables: it recommends to annotate the `m:bvar` elements that declare the bound variable with an `id` attribute and use the `definitionURL` attribute on the bound occurrences of the `m:ci` element to point to those. The following example is taken from [KD03a], which has more details.

```
<m:lambda>
  <m:bvar><m:ci xml:id="the−boundvar">complex presentation</m:ci></m:bvar>
  <m:apply>
    <m:plus/>
    <m:ci definitionURL="#the−boundvar">complex presentation</m:ci>
    <m:ci definitionURL="#the−boundvar">complex presentation</m:ci>
  </m:apply>
</m:lambda>
```

For presentation in MathML, this gives us the best of both approaches, the `m:ci` content can be used, and the pointer gives a simple semantic equivalence criterion. For presenting OpenMath and Content-MathML in other formats OMDoc makes use of the infrastructure introduced in module PRES; see Section **??** for a discussion.

13.5 Legacy Representation for Migration

`legacy`

Sometimes, OMDoc is used as a migration format from legacy texts (see Chapter 4 for an example). In such documents it can be too much effort to convert all mathematical objects and formulae into OpenMath or Content-MathML form. For this situation OMDoc provides the `legacy` element, which can contain arbitrary math markup[11]. The `legacy` element can occur

[9] Note that surprisingly, the empty Content-MathML elements are treated more in the OpenMath spirit.

[10] In the current OpenMath standard, variable names are restricted to alphanumeric characters starting with a letter. Note that unlike with symbols, we cannot associate presentation information with variables via style sheets, since these are not globally unique (see Section **??** for a discussion of the OMDoc solution to this problem).

[11] If the content is an XML-based, format like Scalable Vector Graphics [JFF02], the DTD must be augmented accordingly for validation.

wherever an `om:OMOBJ` or `m:math` can and has an optional `xml:id` attribute for identification. The content is described by a pair of attributes:

- `format` (required) specifies the format of the content using URI reference. OMDoc does not restrict the possible values, possible values include `TeX`, `pmml`, `html`, and `qmath`.
- `formalism` is optional and describes the formalism (if applicable) the content is expressed in. Again, the value is unrestricted character data to allow a URI reference to a definition of a formalism.

For instance in the following `legacy` element, the identity function is encoded in the untyped λ-calculus, which is characterized by a reference to the relevant Wikipedia article.

```
<legacy format=”TeX” formalism=”http://en.wikipedia.org/wiki/Lambda_calculus”>
  \lambda{x}{x}
</legacy>
```

14

Mathematical Text (Modules MTXT and RT)

The everyday mathematical language used in textbooks, conversations, and written onto blackboards all over the world consists of a rigorous, slightly stylized version of natural language interspersed with mathematical formulae, that is sometimes called **mathematical vernacular**[1].

Element	Attributes		D	Content
	Required	Optional	C	
`omtext`		`xml:id, type, for, from, class, style, verbalizes`	+	`uses*, CMP+, FMP*`
`CMP`		`xml:id, xml:lang`	–	⟪*XHTML Block Level*⟫
`uses`	`from`	`id, type, class, style`	+	

Fig. 14.1. Mathematical Text

14.1 Multilingual Mathematical Vernacular

OMDoc models mathematical vernacular as parsed text interspersed with content-carrying elements. Most prominently, the `om:OMOBJ`, `m:math`, and `legacy` elements are used for mathematical objects, see Chapter 13. Other elements structure the text. In Figure 14.2 we have given an overview over the ones described in this book. The last two modules in Figure 14.2 are optional (see Section 22.3). Other (external or future) OMDoc modules can introduce further elements; natural extensions come when OMDoc is applied to areas outside mathematics, for instance computer science vernacular

[1] The term "mathematical vernacular" was first introduced by Nicolaas Govert de Bruijn in the 1970s (see [Bru94] for a discussion). It derives from the word "vernacular" used in the Catholic church to distinguish the language used by laymen from the official Latin.

needs to talk about code fragments (see Section 20.1 and [Koh]), chemistry vernacular about chemical formulae (e.g. represented in Chemical Markup Language [MR+]).

Recall that OpenMath objects are only well-formed, if all the content dictionaries are declared in a local catalog. given by OMDoc theories (see Section 15.6). As mathematical vernacular contains OpenMath objects, we need a mechanism that allows us to define a CD catalog. OMDoc provides

uses

the `uses` elements for this. An element

Listing 14.1. Opening a CD Catalog

```
<uses from="http://example.org/cds/foo.omdoc#foo"/>
```

opens the OMDoc content dictionary `foo` from the OMDoc document `http://example.org/cds/foo.omdoc`. Note that in contrast to the `imports` element (see Section ??) the `uses` element can be used outside a `theory` element thus does not contribute to the available catalogs.

The `uses` element can be used as a direct child of any OMDoc element that can take a metadata element, and its scope is limited to this element.

Module	Elements	Comment	see
MOBJ	`om:OMOBJ`, `m:math`, `legacy`	mathematical Objects	p. 121
MTXT	`oref`, `term`	phrase-level markup	below
DOC	`ignore`	document structure	p. 97
RT	Block/Inline level XHTML	rich text structure	p. ??
EXT	`omlet`	for applets, images, …	p. 205

Fig. 14.2. OMDoc Modules Contributing to Mathematical Vernacular

14.1.1 Commented Mathematical Properties

CMP

To be able to support multilingual documents, the mathematical vernacular is represented as a groups of `CMP`[2] elements which contain the vernacular and have an optional `xml:lang` attribute that specifies the language they are written in. Conforming with the XML recommendation, we use the ISO 639 two-letter country codes (`de` $\widehat{=}$ German, `en` $\widehat{=}$ English, `fr` $\widehat{=}$ French, `nl` $\widehat{=}$ Dutch, …). If no `xml:lang` is given, then `en` is assumed as the default value. It is forbidden to have two or more sibling `CMP` with the same value of `xml:lang`, moreover, `CMP`s that are siblings must be translations of each other.[3] We speak of a multilingual group of `CMP` elements if this is the case.

[2] The name comes from "Commented Mathematical Property" and was originally taken from OpenMath content dictionaries for continuity reasons. Note that XML does note confuse the two, since they are in different namespaces.

[3] The translation requirement may be alleviated in the future, when further variant relations are encoded in `CMP` groups (see [KK06b] for a discussion in the context

Listing 14.2. A Multilingual Group of `CMP` Elements

```
<CMP><xhtml:p>
  Let <OMOBJ id="set"><OMV name="V"/></OMOBJ> be a set.
  A <term role="definiendum">unary operation</term> on
  <OMOBJ><OMR href="#set"/></OMOBJ> is a function
  <OMOBJ id="fun"><OMV name="F"/></OMOBJ> with
  <OMOBJ id="im">
    <OMA>
      <OMS cd="relations1" name="eq"/>
      <OMA><OMS cd="fns1" name="domain"/><OMV name="F"/></OMA>
      <OMV name="V"/>
    </OMA>
  </OMOBJ> and
  <OMOBJ id="ran">
    <OMA>
      <OMS cd="relations1" name="eq"/>
      <OMA><OMS cd="fns1" name="range"/><OMV name="F"/></OMA>
      <OMV name="V"/>
    </OMA>
  </OMOBJ>.</xhtml:p>
</CMP>
<CMP xml:lang="de"><xhtml:p>
  Sei <OMOBJ><OMR href="#set"/></OMOBJ> eine Menge.
  Eine <term role="definiendum">unäre Operation</term>
  ist eine Funktion <OMOBJ><OMR href="#fun"/></OMOBJ>, so dass
  <OMOBJ><OMR href="#im"/></OMOBJ> und
  <OMOBJ><OMR href="#ran"/></OMOBJ>.</xhtml:p>
</CMP>
<CMP xml:lang="fr"><xhtml:p>
  Soit <OMOBJ><OMR href="#set"/></OMOBJ> un ensemble.
  Une <term role="definiendum">opération unaire</term> sûr
  <OMOBJ><OMR href="#set"/></OMOBJ> est une fonction
  <OMOBJ><OMR href="#fun"/></OMOBJ> avec
  <OMOBJ><OMR href="#im"/></OMOBJ> et
  <OMOBJ><OMR href="#ran"/></OMOBJ>.</xhtml:p>
</CMP>
```

Listing 14.2 shows an example of such a multilingual group. Here, the OpenMath extension by DAG representation (see Section 13.1) facilitates multi-language support: Only the language-dependent parts of the text have to be rewritten, the (language-independent) formulae can simply be re-used by cross-referencing.

14.1.2 Paragraph-Level Text Markup

Paragraph-level markup is given mostly given by block-level elements of the XHTML Text, Table, and List modules of XHTML 1.1 [McC10]. Concretely, these are the elements `address`, `blockquote`, `div`, `p`, `pre`, `ul`, `ol`, `dl`, and `table`. All of these elements have been updated to allow `metadata` elements and the `verbalizes`, `type`, and `index` attributes, which we explain next

14.1.3 The Verbalization Relation

The value of the `verbalizes` attribute is a whitespace-separated list of URI references that act as pointers to other OMDoc elements. This has two ap-

---

of “communities of practice”). Then a generalized uniqueness condition must be observed in `CMP` groups, so that systems can choose between the supplied variants.

plications: the first is another kind of parallel markup where we can state that a phrase corresponds to (and thus "verbalizes") a part of formula in a sibling `FMP` element.

Listing 14.3. Parallel Markup between Formal and Informal

```
<CMP>
  <xhtml:p>
    If <xhtml:span verbalizes="#isaG">⟨G,∘⟩ is a group</xhtml:span>, then of course
      <xhtml:span verbalizes="#isaM">it is a monoid</xhtml:span> by construction.
  </xhtml:p>
</CMP>
<FMP>
  <OMOBJ>
    <OMA><OMS cd="logic1" name="implies"/>
      <OMA id="isaG"><OMS cd="algebra" name="group"/>
        <OMA id="GG"><OMS cd="set" name="pair">
          <OMV name="G"/><OMV name="op"/>
        </OMA>
      </OMA>
      <OMA xml:id="isaM"><OMS cd="algebra" name="monoid"/>
        <OMR href="GG"/>
      </OMA>
    </OMA>
  </OMOBJ>
</FMP>
```

Another important application of the `verbalizes` is the case of inline mathematical statements, which we will discuss in Section 15.5.

14.1.4 Parallel Multilingual Markup

Recall that sibling `CMP` elements form multilingual groups of text fragments. We can use the paragraph-level and phrase-level element to make the correspondence relation on text fragments more fine-grained: elements in sibling `CMP`s that have the same `index` value are considered to be equivalent. Of course, the value of an `index` has to be unique in the dominating `CMP` element (but not beyond). Thus the `index` attributes simplify manipulation of multilingual texts, see Listing 14.4 for an example at the discourse level.

Listing 14.4. Multilingual Parallel Markup

```
<omtext xml:id="animals.overview">
  <CMP>
    <xhtml:p index="intro">Consider the following animals:</xhtml:p>
    <xhtml:ul index="animals">
      <xhtml:li index="first">a tiger,</xhtml:li>
      <xhtml:li index="second">a dog.</xhtml:li>
    </xhtml:ul>
  </CMP>
  <CMP xml:lang="de">
    <xhtml:p index="intro">Betrachte die folgenden Tiere:</xhtml:p>
    <xhtml:ul index="animals">
      <xhtml:li index="first">Ein Tiger</xhtml:li>
      <xhtml:li index="second">Ein Hund</xhtml:li>
    </xhtml:ul>
  </CMP>
</omtext>
```

14.2 Phrase-Level Markup of Mathematical Vernacular

To make the sentence-internal structure of mathematical vernacular more explicit, OMDoc provides an infrastructure to mark up natural language phrases in sentences. Linguistically, a **phrase** is a group of words that functions as a single unit in the syntax of a sentence. Examples include "noun phrases, verb phrases, or prepositional phrases".

14.2.1 XHTML Phrase-Level Markup

Phrase-level markup is given mostly given by inline-level elements of the XHTML Text, Applet, Bi-directional Text, Hypertext, Image, and Object, modules of XHTML 1.1 [McC10]. Concretely, these are the elements `abbr`, `acronym`, `br`, `cite`, `code`, `dfn`, `em`, `kbd`, `q`, `samp`, `span`, `strong`, `var`.

14.2.2 OMDoc References

OMDoc supplies the `oref` element for referencing fragments of other documents[4]. `oref` is an inline element that specifies the target element to be referenced via a `oref` attribute. Its content is the default link text. The processing of the `oref` is application specific. It is recommended to generate an appropriate label and (optionally) supply a hyper-reference. If that is not possible, the default text in the body of the `oref` element can be used. `oref`

Element	Attributes		Content
citation	bibrefs	empty	
oref	href	default text	

Fig. 14.3. Hyperlinkds, Citations, & References

14.2.3 Citations

The `citation` element is marks up a citation. Its `bibrefs` attribute references entries in a LaTeXML bibTeX/XML file. `citation`

The `type` attribute can be used to specify the (linguistic or mathematical) type of the phrase, currently OMDoc does not make any restrictions on the values of this attribute, for the mathematical type we recommend to use values for the `type` attribute specified in Section 14.4.

14.2.4 Notes

The `note` element is the closest approximation to a footnote or endnote, `note`

[4] OMDoc1.2 used the `ref` element with `type cite` for this purpose.

Element	Attributes		D	Content
	Required	Optional	C	
term	cd, name	cdbase, role, xml:id, class, style	–	⟪*math vernacular*⟫
citation	href | bibref	xml:id, class, style	–	⟪*math vernacular*⟫
note		type, xml:id, style, class, index, verbalizes	+	⟪*math vernacular*⟫

Fig. 14.4. Phrase-level Markup

where the kind of note is determined by the `type` attribute. OMDoc supplies `footnote` as a default value, but does not restrict the range of values. Its `for` attribute allows it to be attached to other OMDoc elements externally where it is not allowed by the OMDoc document type. In our example, we have attached a footnote by reference to a table row, which does not allow `note` children.

14.2.5 Index Markup

idx

idt

The `idx` element is used for index markup in OMDoc. It contains an optional `idt` element that contains the index text, i.e. the phrase that is indexed.

Element	Attributes		D	Content
idx		(xml:id|xref)	–	idt?, ide+
ide		index, sort-by, see, seealso, links	–	idp*
idt		style, class	–	⟪*math vernacular*⟫
idp		sort-by, see, seealso, links	–	⟪*math vernacular*⟫

Fig. 14.5. Index Markup

ide

idp

The remaining content of the index element specifies what is entered into various indexes. For every index this phrase is registered to there is one `ide` element (index entry); the respective entry is specified by name in its optional `index` attribute. The `ide` element contains a sequence of index phrases given in `idp` elements. The content of an `idp` element is regular mathematical text. Since index entries are usually sorted, (and mathematical text is difficult to sort), they carry an attribute `sort-by` whose value (a sequence of Unicode characters) can be sorted lexically [DW05]. Moreover, each `idp` and `ide` element carries the attributes `see`, `seealso`, and `links`, that allow to specify extra information on these. The values of the first ones are references to `idx` elements, while the value of the `links` attribute is a whitespace-separated list of (external) URI references. The formatting of the index text is governed by the attributes `style` and `class` on the `idt` element. The `idx` element can carry either an `xml:id` attribute (if this is the defining occurrence of the index text) or an `xref` attribute. In the latter case, all the `ide` elements from the defining `idx` (the one that has the `xml:id` attribute) are imported into the referring `idx` element (the one that has the `xref` attribute).

14.2.6 Technical Terms

In OMDoc we can give the notion of a **technical term** a very precise meaning: it is a phrase representing a concept for which a declaration exists in a content dictionary (see Subsection 15.2.1). In this respect it is the natural language equivalent for an OpenMath symbol or a Content-MathML token[5]. Let us consider an example: We can equivalently say "$0 \in \mathbb{N}$" and "the number zero is a natural number". The first rendering in a formula, we would cast as the following OpenMath object:

```
<OMOBJ>
  <OMA><OMS cd="set1" name="in"/>
    <OMS cd="nat" name="zero"/>
    <OMS cd="nat" name="Nats"/>
  </OMA>
</OMOBJ>
```

with the effect that the components of the formula are disambiguated by pointing to the respective content dictionaries. Moreover, this information can be used by added-value services e.g. to cross-link the symbol presentations in the formula to their definition (see Chapter ??), or to detect logical dependencies. To allow this for mathematical vernacular as well, we provide the `term` element: in our example we might use the following markup.

```
...<term cd="nat" name="zero">the number zero</term> is an
<term cd="nat" name="Nats">natural number</term>...
```

The `term` element has two required attributes: `cd` and `name`, and optionally `cdbase`, which together determine the meaning of the phrase just like they do for `om:OMS` elements (see the discussion in Section 13.1 and Subsection 15.6.2). The `term` element also allows the attribute `xml:id` for identification of the phrase occurrence, the CSS attributes for styling and the optional `role` attribute that allows to specify the role the respective phrase plays. We reserve the value `definiens` for the defining occurrence of a phrase in a definition. This will in general mark the exact point to point to when presenting other occurrences of the same[6] phrase. Other attribute values for the `role` are possible, OMDoc does not fix them at the current time. Consider for instance the following text fragment from Figure 4.1 in Chapter 4. `term`

> Definition 1. *Let E be a set. A mapping of $E \times E$ is called a* **law of composition** *on E. The value $f(x, y)$ of f for an ordered pair $(x, y) \in E \times E$ is called the* **composition of** *x and y under this law. A set with a law of composition is called a magma.*

[5] and is subject to the same visibility and scoping conditions as those; see Section 15.6 for details

[6] We understand this to mean with the same `cd` and `name` attributes.

Here the first boldface term is the definiendum for a “law of composition”, the second one for the result of applying this to two arguments. It seems that this is not a totally different concept that is defined here, but is derived systematically from the concept of a “law of composition” defined before. Pending a thorough linguistic investigation we will mark up such occurrences with `definiens-applied`, for instance in

Listing 14.5. Marking up the Technical Terms

```
Let E be a set. A mapping of E × E is called a
<term cd="magmas" name="law_of_comp" role="definiendum">law of composition</term> on E.
The value f(x, y) of f for an ordered pair (x, y) ∈ E × E is called the
<term cd="magmas"name="law_of_comp" role="definiendum−applied">composition of</term>
x and y under this law.
```

There are probably more such systematic correlations; we leave their categorization and modeling in OMDOC to the future.

14.3 Declarations and Discourse Referents

In mathematics we often see phrases like

1. “*Let S be a set,...*”,
2. “*..., where f is a smooth function on the reals*”, or
3. “*for some $\epsilon > \delta > 0$, ...*”.

These introduce identifiers for the use in the text and formulae (with a given scope.) Traditionally, these are thought of as variables, but the concept of **discourse referents** from DRT (Discourse Representation Theory [KR96]) seems more suitable.

Element	Attributes		D	Content
`declaration`	`for`	`id, role`	–	⟪*math vernacular*⟫, `condition*, restrictions*`
`condition`	`for`	`id`	–	⟪*math vernacular*⟫
`restriction`	`for`		–	⟪*math vernacular*⟫

Fig. 14.6. Declaration Markup

`declaration`

`restriction`

We use `declaration` element to markup such text fragments and the `restriction` element for marking up the noun phrase that further specifies the (mathematical) range of the declared identifiers. The `for` attributes on those elements reference the (`id` attributes of) the identifiers declared. Listing 14.6 shows a simple example.

Listing 14.6. A simple Declaration with Restriction

```
<declaration for="#theS" role="indefinite universal">
  Let <om:OMOBJ><om:OMV id="theS" name="S"/></om:OMOBJ> be
  <restriction for="#theS">a set</restriction>, ...
</declaration>
```

For the third example we get the form in Listing 14.7. The conditions marked up in the `condition` elements are grammatically of sentence category, they state conditions on (some of the) discourse referents. Note that declarations can have more than one conditions and restrictions whose `for` attributes reference a subset of the identifiers of the declaration – which they should cover together.

`condition`

Listing 14.7. A Declaration with Condition

```
<declaration for="#e #d" role="indefinite existential">
  for some
  <condition for="#e #d">
    <om:OMOBJ>
      <om:OMA><om:OMS cd="relations" name="greater"/>
        <om:OMV id="e" name="ϵ"/>
        <om:OMV id="d" name="δ"/>
        <om:OMI>0</om:OMI>
      </om:OMA>
    </om:OMOBJ>
  </condition>
</declaration>
```

Note also that `declaration` elements can be nested; they can even appear in `condition` and `restriction` elements, e.g. for example in "*Let x be the integer part of $\sqrt{p}$ for some prime number p.*", which is given in Listing 14.8.

Listing 14.8. Nested Declarations

```
<declaration role="definite">
  Let x be
  <condition>
    the integer part of √p
    <declaration role="indefinite existential">
      for <restriction>some prime number<restriction> p.
    </declaration>
  </condition>
</declaration>
```

In all of these, the `role` gives the "type" of the declaration. The first two examples were indefinite, i.e. the declared discourse referents range over the domain given by the restriction or condition existentially or universally in the current context, whereas the outer declaration in the last example is definite, i.e. the value of the discourse referent is fixed by the condition or restriction (in the current context).

14.4 Text Fragments and their Rhetoric/Mathematical Roles

As we have explicated above, all mathematical documents state properties of mathematical objects — informally in mathematical vernacular or formally (as logical formulae), or both. OMDoc uses the `omtext` element to mark up text passages that form conceptual units, e.g. paragraphs, statements, or remarks. `omtext` elements have an optional `xml:id` attribute, so that they can

`omtext`

be cross-referenced, the intended purpose of the text fragment in the larger document context can be described by the optional attribute `type`. This can take e.g. the values `abstract`, `introduction`, `conclusion`, `comment`, `thesis`, `antithesis`, `elaboration`, `motivation`, `evidence`, `note`, `transition` with the obvious meanings. In the last five cases `omtext` also has the extra attribute `for`, and in the last one, also an attribute `from`, since these are in reference to other OMDoc elements.

The content of an `omtext` element is mathematical vernacular contained in a multi-lingual `CMP` group, followed by an (optional) multi-logic `FMP` group that expresses the same content. This `CMP` group can be preceded by a `metadata` element that can be used to specify authorship, give the passage a title, etc. (see Section 12.2).

We have used the `type` attribute on `omtext` to classify text fragments by their rhetoric role. This is adequate for much of the generic text that makes up the narrative and explanatory text in a mathematical textbook. But many text fragments in mathematical documents directly state properties of mathematical objects (we will call them mathematical statements; see Chapter 15 for a more elaborated markup infrastructure). These are usually classified as definitions, axioms, etc. Moreover, they are of a form that can (in principle) be formalized up to the level of logical formula; in fact, mathematical vernacular is seen by mathematicians as a more convenient form of communication for mathematical statements that can ultimately be translated into a foundational logical system like axiomatic set theory [Ber91]. For such text fragments, OMDoc reserves the following values for the `type` attribute:

`axiom` (fixes or restricts the meaning of certain symbols or concepts.) An axiom is a piece of mathematical knowledge that cannot be derived from anything else we know.
`definition` (introduces new concepts or symbols.) A definition is an axiom that introduces a new symbol or construct, without restricting the meaning of others.
`example` (for or against a mathematical property).
`proof` (a proof), i.e. a rigorous — but maybe informal — argument that a mathematical statement holds.
`hypothesis` (a local assumption in a proof that will be discharged later) for text fragments that come from (parts of) proofs.
`derive` (a step in a proof), we will specify the exact meanings of this and the two above in Chapter 17 and present more structured counterparts.

Finally, OMDoc also reserves the values `assertion`, `theorem`, `proposition`, `lemma`, `corollary`, `postulate`, `conjecture`, `false-conjecture`, `assumption`, `obligation`, `rule` and `formula` for statements that assert properties of mathematical objects (see Figure 15.5 in Subsection 15.3.1 for explanations). Note that the differences between these values are largely pragmatic or proof-theoretic (conjectures become theorems once there is a proof). Mathematical `omtext` elements (such with one of these types) can have additional `FMP` elements (Formal Mathematical Property) that formally represents the meaning of the descriptive text in the CMPs (if that is feasible).

Further types of text can be specified by providing a URI that points to a description of the text type (much like the `definitionURL` attribute on the `m:csymbol` elements in Content-MathML).

Of course, the `type` only allows a rough classification of the mathematical statements at the text level, and does not make the underlying content structure explicit or reveals their contribution and interaction with mathematical context. For that purpose OMDoc supplies a set of specialized elements, which we will discuss in Chapter 15. Thus `omtext` elements will be used to give informal accounts of mathematical statements that are better and more fully annotated by the infrastructure introduced in Chapter 15. However, in narrative documents, we often want to be informal, while maintaining a link to the formal element. For this purpose OMDoc provides the optional `verbalizes` attribute on the `omtext` element. Its value is a whitespace-separated list of URI references to formal representations (see Section 15.5 for further discussion).

14.5 Formal Mathematical Properties

An FMP[7] element is the general element for representing formal mathematical content in the form of OpenMath objects. FMPs always appear in groups, which can differ in the value of their `logic` attribute, which specifies the logical formalism. The value of this attribute specifies the logical system used in formalizing the content. All members of the group have to formalize the same mathematical object or property, i.e. they have to be translations of each other, like siblings CMPs, we speak of a **multi-logic FMP group** in this case. Furthermore, if an FMP group has CMP siblings, all must express the same content. FMP

Element	Attributes		D	Content
	Required	Optional	C	
FMP		xml:id, logic	–	(assumption*, conclusion*) | OMOBJ |m:math |legacy
assumption		xml:id, inductive, class, style	+	(OMOBJ |m:math |legacy)
conclusion		xml:id, class, style	+	(OMOBJ |m:math |legacy)

Fig. 14.7. Formal Mathematical Properties

In Listing 14.9 we see two FMP elements, that state the property of being a unary operation in two logics. The first one (`fol` for first-order logic) uses

[7] The name comes from "Formal Mathematical Properties" and was originally taken from OpenMath content dictionaries for continuity reasons.

an equivalence to convey the restriction, the second one (hol for higher-order logic) has λ-abstraction and can therefore define the binary predicate binop directly.

Listing 14.9. A multi-logic FMP group for Listing 14.2.

```
<omtext xml:id="binop−def" type="definition">
  ... the content of Listing 14.2 here ...
  <FMP logic="fol">∀V, F.binop(F, V) ⇔ Im(F) = V ∧ Dom(F) = V</FMP>
  <FMP logic="hol">binop = λV, F.Im(F) = V ∧ Dom(F) = V</FMP>
</omtext>
```

assumption

conclusion

As mathematical statements of properties of objects often come as **sequent**s, i.e. as sets of conclusions drawn from a set of assumptions, OMDoc also allows the content of an FMP to be a (possibly empty) set of assumption elements followed by a (possibly empty) set of conclusion elements. The intended meaning is that the FMP asserts that one of the conclusions is entailed by the assumptions together in the current context. As a consequence

```
<FMP><conclusion>A</conclusion></FMP>
```

is equivalent to <FMP>A</FMP>, where A is an OpenMath, Content-MathML, or legacy representation of a mathematical formula. The assumption and conclusion elements allow to specify the content by an om:OMOBJ, m:math, or legacy element. The assumption and conclusion elements carry an optional xml:id attribute, which can be used for structure sharing. This is important for specifying sequent-style proofs (see Chapter 17), where the assumptions and conclusions of sequents are largely invariant over a proof and would have to be copied otherwise. The assumption element carries an additional optional attribute inductive for inductive hypotheses.

In the (somewhat contrived) example in Listing 14.10 we show a sequent for a simple fact about set intersection. Here the knowledge in both assumptions (together) is enough to entail one of the conclusions (the first in this case).

Listing 14.10. Representing Vernacular as an FMP Sequent

```
<CMP><xhtml:p>If a ∈ U and a ∈ V, then a ∈ U ∩ V or
  <xhtml:span index="moon_cheese">the moon is made of green cheese</xhtml:span>.
</xhtml:p></CMP>
<FMP>
  <assumption xml:id="A">a ∈ U</assumption>
  <assumption xml:id="B">a ∈ V</assumption>
  <conclusion xml:id="C">a ∈ U ∩ V</conclusion>
  <conclusion xml:id="moon_cheese">made_of(moon, gc)</conclusion>
</FMP>
```

15

Mathematical Statements (Module ST)

In this chapter we will look at the OMDoc infrastructure to mark up the *functional structure* of mathematical statements and their interaction with a broader mathematical context.

15.1 Types of Statements in Mathematics

In the last chapter we introduced mathematical statements as special text fragments that state properties of the mathematical objects under discussion and categorized them as definitions, theorems, proofs,.... A set of statements about a related set of objects make up the context that is needed to understand other statements. For instance, to understand a particular theorem about finite groups, we need to understand the definition of a group, its properties, and some basic facts about finite groups first. Thus statements interact with context in two ways: the context is built up from (clusters of) statements, and statements only make sense with reference to a context. Of course this dual interaction of statements with *context*[1] applies to any text and to communication in general. In mathematics, where the problem is aggravated by the load of notation and the need for precision for the communicated concepts and objects, contexts are often discussed under the label of **mathematical theories**. We will distinguish two classes of statements with respect to their interaction with theories: We view axioms and definitions as *constitutive* for a given theory, since changing this information will yield a different theory (with different mathematical properties, see the discussion in Section 2.2). Other mathematical statements like theorems or the proofs that support them are not constitutive, since they only illustrate the mathematical objects in the theory by explicitly stating the properties that are implicitly determined by the constitutive statements.

[1] In linguistics and the philosophy of language this phenomenon is studied under the heading of "discourse theories", see e.g. [KR93] for a start and references.

To support this notion of context OMDoc supports an infrastructure for theories using special `theory` elements, which we will introduce in Section 15.6 and extend in Chapter 18. Theory-constitutive elements must be contained as children in a `theory` element; we will discuss them in Section 15.2, non-constitutive statements will be defined in Section 15.3. They are allowed to occur outside a `theory` element in OMDoc documents (e.g. as top-level elements), however, if they do they must reference a theory, which we will call their **home theory** in a special `theory` attribute. This situates them into the context provided by this theory and gives them access to all its knowledge. The home theory of theory-constitutive statements is given by the theory they are contained in.

The division of statements into constitutive and non-constitutive ones and the encapsulation of constitutive elements in `theory` elements add a certain measure of safety to the knowledge management aspect of OMDoc. Since XML elements cannot straddle document borders, all constitutive parts of a theory must be contained in a single document; no constitutive elements can be added later (by other authors), since this would change the meaning of the theory on which other documents may depend on.

Before we introduce the OMDoc elements for theory-constitutive statements, let us fortify our intuition by considering some mathematical examples. *Axioms* are assertions about (sets of) mathematical objects and concepts that are assumed to be true. There are many forms of axiomatic restrictions of meaning in mathematics. Maybe the best-known are the five Peano Axioms for natural numbers.

1. 0 is a natural number.
2. The successor $s(n)$ of a natural number n is a natural number.
3. 0 is not a successor of any natural number.
4. The successor function is one-one (i.e. injective).
5. The set $\mathbb{N}$ of natural numbers contains only elements that can be constructed by axioms 1. and 2.

Fig. 15.1. The Peano Axioms

The Peano axioms in Figure 15.1 (implicitly) introduce three symbols: the number 0, the successor function s, and the set $\mathbb{N}$ of natural numbers. The five axioms in Figure 15.1 jointly constrain their meaning such that conforming structures exist (the natural numbers we all know and love) any two structures that interpret 0, s, and $\mathbb{N}$ and satisfy these axioms must be isomorphic. This is an ideal situation — the axioms are neither too lax (they allow too many mathematical structures) or too strict (there are no mathematical structures) — which is difficult to obtain. The latter condition (**inconsistent** theories) is especially unsatisfactory, since any statement is a theorem in such theories.

As consistency can easily be lost by adding axioms, mathematicians try to keep axiom systems minimal and only add axioms that are safe.

Sometimes, we can determine that an axiom does not destroy consistency of a theory $\mathcal{T}$ by just looking at its form: for instance, axioms of the form $s = \mathbf{A}$, where s is a symbol that does not occur in $\mathcal{T}$ and $\mathbf{A}$ is a formula containing only symbols from $\mathcal{T}$ will introduce no constraints on the meaning of $\mathcal{T}$-symbols. The axiom $s = \mathbf{A}$ only constrains the meaning of the **new symbol** to be a unique object: the one denoted by $\mathbf{A}$. We speak of a **conservative extension** in this case. So, if $\mathcal{T}$ was a consistent theory, the extension of $\mathcal{T}$ with the symbol s and the axiom $s = \mathbf{A}$ must be one too. Thus axioms that result in conservative extensions can be added safely — i.e. without endangering consistency — to theories.

Generally an axiom $\mathcal{A}$ that results in a conservative extension is called a **definition** and any new symbol it introduces a **definiendum** (usually marked e.g. in boldface font in mathematical texts), and we call **definiens** the material in the definition that determines the meaning of the definiendum. We say that a definiendum is **well-defined**, iff the corresponding definiens uniquely determines it; adding such definitions to a theory always results in a conservative extension.

Definiendum	Definiens	Type
The number 1	$1{:}= s(0)$ (1 is the successor of 0)	simple
The exponential function $e^{\cdot}$	The **exponential function** $e^{\cdot}$ is the solution to the differential equation $\partial f = f$ [where $f(0) = 1$].	implicit
The addition function $+$	**Addition** on the natural numbers is defined by the equations $x + 0 = x$ and $x + s(y) = s(x + y)$.	recursive

Fig. 15.2. Some Common Definitions

Definitions can have many forms, they can be

- equations where the left hand side is the defined symbol and the right hand side is a term that does not contain it, as in our discussion above or the first case in Figure 15.2. We call such definitions **simple**.
- general statements that uniquely determine the meaning of the objects or concepts in question, as in the second definition in Figure 15.2. We call such definitions **implicit**; the Peano axioms are another example of this category.

  Note that this kind of definitions requires a proof of unique existence to ensure well-definedness. Incidentally, if we leave out the part in square brackets in the second definition in Figure 15.2, the differential equation only characterizes the exponential function up to additive real constants. In this case, the “definition” only restricts the meaning of the exponential function to a set of possible values. We call such a set of axioms a **loose** definition.

- given as a set of equations, as in the third case of Figure 15.2, even though this is strictly a special case of an implicit definition: it is a sub-case, where well-definedness can be shown by giving an argument why the systematic applications of these equations terminates, e.g. by exhibiting an ordering that makes the left hand sides strictly smaller than the right-hand sides. We call such a definition **inductive**.

15.2 Theory-Constitutive Statements in OMDoc

The OMDoc format provides an infrastructure for four kinds of theory-constitutive statements: symbol declarations, type declarations, (proper) axioms, and definitions. We will take a look at all of them now.

Element	Attributes		D	Content
	Required	Optional	C	
symbol	`name`	`xml:id, role, scope, style, class`	+	`type*`
type		`xml:id, system, style, class`	–	`CMP*`, ⟪*mobj*⟫
axiom		`xml:id, for, type, style, class`	+	`CMP*,FMP*`
definition	`for`	`xml:id, type, style, class, uniqueness, existence, consistency, exhaustivity`	+	`CMP*, (FMP* | requation+ |` ⟪*mobj*⟫`)?, measure?, ordering?`
requation		`xml:id, style, class`	–	⟪*mobj*⟫,⟪*mobj*⟫
measure		`xml:id, style, class`	–	⟪*mobj*⟫
ordering		`xml:id, style, class`	–	⟪*mobj*⟫
where ⟪*mobj*⟫ is `(OMOBJ |m:math |legacy)`				

Fig. 15.3. Theory-Constitutive Elements in OMDoc

15.2.1 Symbol Declarations

`symbol`

The `symbol` element declares a symbol for a mathematical concept, such as 1 for the natural number "one", + for addition, = for equality, or `group` for the property of being a group. Note that we not only use the `symbol` element for mathematical objects that are usually written with mathematical symbols, but also for any concept or object that has a definition or is restricted in its meaning by axioms.

We will refer to the mathematical object declared by a `symbol` element as a "symbol", iff it is usually communicated by specialized notation in mathematical practice, and as a "concept" otherwise. The name "symbol" of the `symbol` element in OMDoc is in accordance with usage in the philosophical literature (see e.g. [NS81]): A **symbol** is a *mental or physical* representation of a concept. In particular, a symbol may, but need not be representable by a (set of) glyphs (symbolic notation). The definiendum objects in Figure 15.2 would be considered as "symbols" while the concept of a "group" in mathematics would be called a "concept".

The `symbol` element has a required attribute `name` whose value uniquely identifies it in a theory. Since the value of this attribute will be used as an OpenMath symbol name, it must be an XML name[2] as defined in XML 1.1 [Bra+04]. The optional attribute `scope` takes the values `global` and `local`, and allows a simple specification of visibility conditions: if the `scope` attribute of a `symbol` has value `local`, then it is not exported outside the theory; The `scope` attribute is deprecated, a formalization using the `hiding` attribute on the `imports` element should be used instead. Finally, the optional attribute `role` that can take the values[3]

`binder` The symbol may appear as a binding symbol of an binding object, i.e. as the first child of an `om:OMBIND` object, or as the first child of an `m:apply` element that has an `m:bvar` as a second child.

`attribution` The symbol may be used as key in an OpenMath `om:OMATTR` element, i.e. as the first element of a key-value pair, or in an equivalent context (for example to refer to the value of an attribution). This form of attribution may be ignored by an application, so should be used for information which does not change the meaning of the attributed OpenMath object.

`semantic-attribution` This is the same as `attribution` except that it modifies the meaning of the attributed OpenMath object and thus cannot be ignored by an application.

`error` The symbol can only appear as the first child of an OpenMath error object.

`application` The symbol may appear as the first child of an application object.

`constant` The symbol cannot be used to construct a compound object.

`type` The symbol denotes a sets that is used in a type systems to annotate mathematical objects.

`sort` The symbol is used for a set that are inductively built up from constructor symbols; see Chapter 16.

If the `role` is not present, the value `object` is assumed.

The children of the `symbol` element consist of a multi-system group of `type` elements (see Subsection 15.2.3 for a discussion). For this group the

[2] This limits the characters allowed in a name to a subset of the characters in Unicode 2.0; e.g. the colon : is not allowed. Note that this is not a problem, since the name is just used for identification, and does not necessarily specify how a symbol is presented to the human reader. For that, OMDoc provides the notation definition infrastructure presented in Chapter 19.

[3] The first six values come from the OpenMath2 standard. They are specified in content dictionaries; therefore OMDoc also supplies them.

order does not matter. In Listing 15.1 we have a symbol declaration for the concept of a monoid. Keywords or simple phrases that describes the symbol in mathematical vernacular can be added in the `metadata` child of `symbol` as `dc:subject` and `dc:descriptions`; the latter have the same content model as the `CMP` elements, see the discussion in Section 14.1). If the document containing their parent `symbol` element were stored in a data base system, it could be looked up via these metadata. As a consequence the symbol `name` need only be used for identification. In particular, it need not be mnemonic, though it can be, and it need not be language-dependent, since this can be done by suitable `dc:subject` elements.

Listing 15.1. An OMDoc `symbol` Declaration

```
<symbol name="monoid">
  <metadata>
    <dc:subject xml:lang="en">monoid</dc:subject>
    <dc:subject xml:lang="de">Monoid</dc:subject>
    <dc:subject xml:lang="it">monoide</dc:subject>
  </metadata>
  <type system="simply−typed">set[any] → (any → any → any) → any → bool</type>
  <type system="props">
    <OMOBJ><OMS cd="arities" name="ternary−relation"/></OMOBJ>
  </type>
</symbol>
```

15.2.2 Axioms

`axiom`

The relation between the components of a monoid would typically be specified by a set of axioms (e.g. stating that the base set is closed under the operation). For this purpose OMDoc uses the `axiom` element, which allows as children a multilingual group of `CMP`s, which express the mathematical content of the axiom and a multi-logic `FMP` group that expresses this as a logical formula. `axiom` elements may have a `generated-from` attribute, which points to another OMDoc element (e.g. an `adt`, see Chapter 16) which subsumes it, since it is a more succinct representation of the same mathematical content. Finally the `axiom` element has an optional `for` attribute to specify salient semantic objects it uses as a whitespace-separated list of URI references to symbols declared in the same theory, see Listing 15.2 for an example. Finally, the `axiom` element can have an `type` attribute, whose values we leave unspecified for the moment.

Listing 15.2. An OMDoc `axiom`

```
<axiom xml:id="mon.ax" for="monoid">
  <CMP><xhtml:p>If (M, ∗) is a semigroup with unit e, then (M, ∗, e) is a monoid.</xhtml:p></CMP>
</axiom>
```

15.2.3 Type Declarations

Types (also called sorts in some contexts) are representations of certain simple sets that are treated specially in (human or mechanical) reasoning processes. A **type declaration** $e\colon t$ makes the information that a symbol or expression e is in a set represented by a type t available to a specified mathematical process. For instance, we might know that 7 is a natural number, or that expressions of the form $\sum_{i=1}^{n} a_i x^i$ are polynomials, if the a_i are real numbers, and exploit this information in mathematical processes like proving, pattern matching, or while choosing intuitive notations. If a type is declared for an expression that is not a symbol, we will speak of a **term declaration**.

type

OMDoc uses the `type` element for type declarations. The optional attribute `system` contains a URI reference that identifies the type system which interprets the content. There may be various sources of the set membership information conveyed by a type declaration, to justify it this source may be specified in the optional `just-by` attribute. The value of this attribute is a URI reference that points to an `assertion` or `axiom` element that asserts $\forall x_1, \ldots, x_n . e \in t$ for a type declaration $e\colon t$ with variables $x_1, \ldots, x_n$. If the `just-by` attribute is not present, then the type declaration is considered to be generated by an implicit axiom, which is considered theory-constitutive[4].

The `type` element contains one or two mathematical objects. In the first case, it represents a type declaration for a symbol (we call this a **symbol declaration**), which can be specified in the optional `for` attribute or by embedding the `type` element into the respective `symbol` element. For instance in Listing 15.1, the type declaration of `monoid` characterizes a monoid as a three-place predicate (taking as arguments the base set, the operation, and a neutral element).

A `type` element with two mathematical objects represents a term declaration $e\colon t$, where the first object represents the expression e and the second one the type t (see Listing 15.7 for an example). There the term $x + x$ is declared to be an even number by a term declaration.

As reasoning processes vary, information pertaining to multiple type systems may be associated with a single symbol and there can be more than one `type` declaration per expression and type system, this just means that the object has more than one type in the respective type system (not all type systems admit principal types).

15.2.4 Definitions

definition

Definitions are a special class axioms that completely fix the meaning of symbols. Therefore `definition` elements that represent definitions carry the required `for` attribute, which contain a whitespace-separated list of names of symbols in the same theory. Note that this use of the `for` attribute is different from the other usages, which are URI references.

We call symbols that are referenced in definitions **defined** and **primitive** otherwise. `definition` contain a multilingual `CMP` group to describe the meaning of the defined symbols.

[4] It is considered good practice to make the axiom explicit in formal contexts, as this allows an extended automation of the knowledge management process.

In Figure 15.2 we have seen that there are many ways to fix the meaning of a symbol, therefore OMDoc `definition` elements are more complex than `axiom`s. In particular, the `definition` element supports several kinds of definition mechanisms with specialized content models specified in the `type` attribute (cf. the discussion at the end of Section 15.1):

`simple` In this case the `definition` contains a mathematical object that can be substituted for the symbol specified in the `for` attribute of the definition. Listing 15.3 gives an example of a simple definition of the number one from the successor function and zero. OMDoc treats the `type` attribute as an optional attribute. If it is not given explicitly, it defaults to `simple`.

Listing 15.3. A Simple OMDoc `definition`.

```
<symbol name="one"/>
<definition xml:id="one.def" for="one" type="simple">
 <CMP><xhtml:p><OMOBJ><OMS cd="nat" name="one"/></OMOBJ> is the successor of
      <OMOBJ><OMS cd="nat" name="zero"/></OMOBJ>.</xhtml:p></CMP>
 <OMOBJ>
   <OMA>
     <OMS cd="nat" name="suc"/>
     <OMS cd="nat" name="zero"/>
   </OMA>
 </OMOBJ>
</definition>
```

`implicit` This kind of definition is often (more accurately) called "*definition by description*", since the definiendum is described so accurately, that there is exactly one object satisfying the description. The "description" of the defined symbol is given as a multi-system `FMP` group whose content uniquely determines the value of the symbols that are specified in the `for` attribute of the definition. The necessary statement of unique existence can be specified in the `existence` and `uniqueness` attribute, whose values are URI references to to assertional statements (see Subsection 15.3.4) that represent the respective properties. We give an example of an implicit definition in Listing 15.4.

Listing 15.4. An Implicit Definition of the Exponential Function

```
<definition xml:id="exp-def" for="exp" type="implicit"
            uniqueness="#exp-unique" existence="#exp-exists">
  <FMP>exp' = exp ∧ exp(0) = 1</FMP>
</definition>
<assertion xml:id="exp-unique">
  <CMP><xhtml:p>
    There is at most one differentiable function that solves the
     differential equation in definition <oref xref="#exp-def"/>.
  </xhtml:p></CMP>
</assertion>
<assertion xml:id="exp-exists">
  <CMP><xhtml:p>
    The differential equation in <oref xref="#exp-def"/> is solvable.
  </xhtml:p></CMP>
</assertion>
```

`inductive` This is a variant of the `implicit` case above. It defines a **recursive function** by a set of recursive equations (in `requation` elements) whose left and right hand sides are specified by the two children. The first one is called the **pattern**, and the second one the **value**. The intended meaning of the defined symbol is, that the value (with the variables suitably substituted) can be substituted for a formula that matches the pattern element. In this case, the `definition` element can carry the optional attributes `exhaustivity` and `consistency`, which point to `assertion`s stating that the cases spanned by the patterns are exhaustive (i.e. all cases are considered), or that the values are consistent (where the cases overlap, the values are equal).

`requation`

Listing 15.5 gives an example of a a recursive definition of the addition on the natural numbers.

Listing 15.5. A recursive definition of addition

```
<definition xml:id="plus.def" for="plus" type="inductive"
            consistency="#s−not−0" exhaustivity="#s−or−0">
  <metadata><dc:subject>addition</dc:subject></metadata>
  <CMP><xhtml:p>Addition is defined by recursion on the second argument.</xhtml:p></CMP>
  <requation>x + 0 ⇝ x</requation>
  <requation>x + s(y) ⇝ s(x + y)</requation>
</definition>
```

To guarantee termination of the recursive instantiation (necessary to ensure well-definedness), it is possible to specify a measure function and well-founded ordering by the optional `measure` and `ordering` elements which contain mathematical objects. The elements contain mathematical objects. The content of the `measure` element specifies a measure function, i.e. a function from argument tuples for the function defined in the parent `definition` element to a space with an ordering relation which is specified in the `ordering` element. This element also carries an optional attribute `terminating` that points to an `assertion` element that states that this ordering relation is a terminating partial ordering.

`measure`

`ordering`

`pattern` This is a special degenerate case of the recursive definition. A function is defined by a set of `requation` elements, but the defined function does not occur in the second children. This form of definition is often used instead of `simple` in logical languages that do not have a function constructor. It allows to define a function by its behavior on patterns of arguments, for instance in

$$\sin(z) := \frac{1}{2i}(e^{iz} - e^{-iz})$$

As termination is trivial in this case, no `measure` and `ordering` elements appear in the body.

`informal` The definition is completely informal, it only contains a `CMP` element.

15.3 The Unassuming Rest

The bulk of mathematical knowledge is in form of statements that are not theory-constitutive: statements of properties of mathematical objects that are entailed by the theory-constitutive ones. As such, these statements are logically redundant, they do not add new information about the mathematical objects, but they do make their properties explicit. In practice, the entailment is confirmed e.g. by exhibiting a proof of the assertion; we will introduce the infrastructure for proofs in Chapter 17.

Element	Attributes		D	`Content`
	Required	Optional	C	
`assertion`		`xml:id, type, theory, class, style, status, just-by`	+	`CMP*, FMP*`
`type`	`system`	`xml:id, for, just-by, theory, class, style`	−	`CMP*,` ⟪*mobj*⟫`,`⟪*mobj*⟫
`example`	`for`	`xml:id, type, assertion, theory, class, style`	+	`CMP* |` ⟪*mobj*⟫`*`
`alternative`	`for, theory, entailed-by, entails, entailed-by-thm, entails-thm`	`xml:id, type, theory, class, style`	+	`CMP*, (FMP* | requation+ |` ⟪*mobj*⟫`)?, measure?, ordering?`
where ⟪*mobj*⟫ is `(OMOBJ |m:math |legacy)`				

Fig. 15.4. Assertions, Examples, and Alternatives in OMDoc

15.3.1 Assertions

`assertion`

OMDoc uses the `assertion` element for all statements (proven or not) about mathematical objects (see Listing 15.6) that are not axiomatic (i.e. constitutive for the meaning of the concepts or symbols involved). 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 (e.g. 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, which can have the values in Figure 15.5. The `assertion` element also takes an optional `xml:id` element that allows to reference it in a document, an optional `theory` attribute to specify the theory that provides the context for this assertion, and an optional attribute `generated-from`, that points to a higher syntactic construct that generates these assertions, e.g. an abstract data type declaration given by an `adt` element (see Chapter 16).

Value	Explanation
`theorem`, `proposition`	an important assertion with a proof
Note that the meaning of the `type` (in this case the existence of a proof) is not enforced by OMDoc applications. It can be appropriate to give an assertion the `type theorem`, if the author knows of a proof (e.g. in the literature), but has not formalized it in OMDoc yet.	
`lemma`	a less important assertion with a proof
The difference of importance specified in this `type` is even softer than the other ones, since e.g. reusing a mathematical paper as a chapter in a larger monograph, may make it necessary to downgrade a theorem (e.g. the main theorem of the paper) and give it the status of a lemma in the overall work.	
`corollary`	a simple consequence
An assertion is sometimes marked as a corollary to some other statement, if the proof is considered simple. This is often the case for important theorems that are simple to get from technical lemmata.	
`postulate`, `conjecture`	an assertion without proof or counter-example
Conjectures are assertions, whose semantic value is not yet decided, but which the author considers likely to be true. In particular, there is no proof or counter-example (see Section 15.4).	
`false-conjecture`	an assertion with a counter-example
A conjecture that has proven to be false, i.e. it has a counter-example. Such assertions are often kept for illustration and historical purposes.	
`obligation`, `assumption`	an assertion on which the proof of another depends
These kinds of assertions are convenient during the exploration of a mathematical theory. They can be used and proven later (or assumed as an axiom).	
`formula`	if everything else fails
This type is the catch-all if none of the others applies.	

Fig. 15.5. Types of Mathematical Assertions

Listing 15.6. An OMDoc Assertion About Semigroups

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

To specify its proof-theoretic status of an assertion `assertion` carries the two optional attributes `status` and `just-by`. The first contains a keyword for the status and the second a whitespace-separated list of URI references to OMDoc elements that justify this status of the assertion. For the specification of the status we adapt an ontology for deductive states of assertion from [SZS04] (see Figure 15.6). Note that the states in Figure 15.6 are not mutually exclusive, but have the inclusions depicted in Figure 15.7.

`status`	`just-by` points to
`tautology`	Proof of $\mathcal{F}$
All $\mathcal{T}$-interpretations satisfy $\mathcal{A}$ and some $\mathcal{C}_i$	
`tautologous-conclusion`	Proof of $\mathcal{F}_c$.
All $\mathcal{T}$-interpretations satisfy some $\mathcal{C}_j$	
`equivalent`	Proofs of $\mathcal{F}$ and $\mathcal{F}^{-1}$
$\mathcal{A}$ and $\mathcal{C}$ have the same $\mathcal{T}$-models (and there are some)	
`theorem`	Proof of $\mathcal{F}$
All $\mathcal{T}$-models of $\mathcal{A}$ (and there are some) satisfy some $\mathcal{C}_i$	
`satisfiable`	Model of $\mathcal{A}$ and some $\mathcal{C}_i$
Some $\mathcal{T}$-models of $\mathcal{A}$ (and there are some) satisfy some $\mathcal{C}_i$	
`contradictory-axioms`	Refutation of $\mathcal{A}$
There are no $\mathcal{T}$-models of $\mathcal{A}$	
`no-consequence`	$\mathcal{T}$-model of $\mathcal{A}$ and some $\mathcal{C}_i$, $\mathcal{T}$-model of $\mathcal{A} \cup \overline{\mathcal{C}}$.
Some $\mathcal{T}$-models of $\mathcal{A}$ (and there are some) satisfy some $\mathcal{C}_i$, some satisfy $\overline{\mathcal{C}}$	
`counter-satisfiable`	Model of $\mathcal{A} \cup \overline{\mathcal{C}}$
Some $\mathcal{T}$-models of $\mathcal{A}$ (and there are some) satisfy $\overline{\mathcal{C}}$	
`counter-theorem`	Proof of $\overline{\mathcal{C}}$ from $\mathcal{A}$
All $\mathcal{T}$-models of $\mathcal{A}$ (and there are some) satisfy $\overline{\mathcal{C}}$	
`counter-equivalent`	Proof of $\overline{\mathcal{C}}$ from $\mathcal{A}$ and proof of $\mathcal{A}$ from $\overline{\mathcal{C}}$
$\mathcal{A}$ and $\overline{\mathcal{C}}$ have the same $\mathcal{T}$-models (and there are some)	
`unsatisfiable-conclusion`	Proof of $\overline{\mathcal{C}}$
All $\mathcal{T}$-interpretations satisfy $\overline{\mathcal{C}}$	
`unsatisfiable`	Proof of $\neg\mathcal{F}$
All $\mathcal{T}$-interpretations satisfy $\mathcal{A}$ and $\overline{\mathcal{C}}$	

Where $\mathcal{F}$ is an assertion whose `FMP` has `assumption` elements $\mathcal{A}_1, \ldots, \mathcal{A}_n$ and `conclusion` elements $\mathcal{C}_1, \ldots, \mathcal{C}_m$. Furthermore, let $\mathcal{A} := \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ and $\mathcal{C} := \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$, and $\mathcal{F}^{-1}$ be the sequent that has the $\mathcal{C}_i$ as assumptions and the $\mathcal{A}_i$ as conclusions. Finally, let $\overline{\mathcal{C}} := \{\overline{\mathcal{C}_1}, \ldots, \overline{\mathcal{C}_m}\}$, where $\overline{\mathcal{C}_i}$ is a negation of $\mathcal{C}_i$.

Fig. 15.6. Proof Status for Assertions in a Theory $\mathcal{T}$

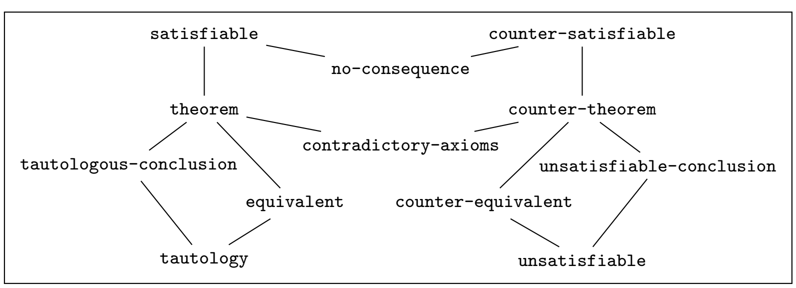


Fig. 15.7. Relations of Assertion States

15.3.2 Type Assertions

In the last section, we have discussed the `type` elements in `symbol` declarations. These were axiomatic (and thus theory-constitutive) in character, declaring a symbol to be of a certain type, which makes this information available to type checkers that can check well-typedness (and thus plausibility) of the represented mathematical objects.

However, not all type information is axiomatic, it can also be deduced from other sources knowledge. We use the same `type` element we have discussed in Subsection 15.2.3 for such **type assertions**, i.e. non-constitutive statements that inform a type-checker. In this case, the `type` element can occur at top level, and even outside a `theory` element (in which case they have to specify their home theory in the `theory` attribute).

Listing 15.7 contains a type assertion $x+x\colon evens$, which makes the information that doubling an integer number results in an even number available to the reasoning process.

Listing 15.7. A Term declaration in OMDoc.

```
<type xml:id="double−even.td" system="#POST"
      theory="adv.int" for="plus" just−by="#double−even">
  <m:math>
    <m:apply><m:plus/>
      <m:ci type="integer">X</m:ci>
      <m:ci type="integer">X</m:ci>
    </m:apply>
  </m:math>
  <m:math>
    <m:csymbol definitionURL="http://omdoc.org/cd/integers/evens"/>
  </m:math>
</type>

<assertion xml:id="double−even" type="lemma" theory="adv.int">
  <FMP>
    <m:math>
      <m:apply><m:forall/>
        <m:bvar><m:ci xml:id="x13" type="integer">X</m:ci></m:bvar>
        <m:apply><m:in/>
          <m:apply><m:plus/>
            <m:ci definitionURL="x13" type="integer">X</m:ci>
            <m:ci definitionURL="x13" type="integer">X</m:ci>
          </m:apply>
          <m:csymbol definitionURL="http://omdoc.org/cd/nat/evens"/>
        </m:apply>
      </m:apply>
    </m:math>
  </FMP>
</assertion>
```

The body of a type assertion contains two mathematical objects, first the type of the object and the second one is the object that is asserted to have this type.

15.3.3 Alternative Definitions

In contrast to what we have said about conservative extensions at the end of Subsection 15.2.4, mathematical documents often contain multiple definitions alternative

for a concept or mathematical object. However, if they do, they also contain a careful analysis of equivalence among them. OMDoc allows us to model this by providing the `alternative` element. Conceptually, an alternative definition or axiom is just a group of assertions that specify the equivalence of logical formulae. Of course, alternatives can only be added in a consistent way to a body of mathematical knowledge, if it is guaranteed that it is equivalent to the existing ones. The `for` on the `alternative` points to the symbol to which the alternative definition pertains. Therefore, `alternative` has the attributes `entails` and `entailed-by`, that specify `assertion`s that state the necessary entailments. It is an integrity condition of OMDoc that any `alternative` element references at least one `definition` or `alternative` element that entails it and one that it is entailed by (more can be given for convenience). The `entails-thm`, and `entailed-by-thm` attributes specify the corresponding assertions. This way we can always reconstruct equivalence of all definitions for a given symbol. As alternative definitions are not theory-constitutive, they can appear outside a `theory` element as long as they have a `theory` attribute.

15.3.4 Assertional Statements

There is another distinction for statements that we will need in the following. Some kinds of mathematical statements add information about the mathematical objects in question, whereas other statements do not. For instance, a symbol declaration only declares an unambiguous name for an object. We will call the following OMDoc elements **assertional**: `axiom` (it asserts central properties about an object), `type` (it asserts type properties about an object), `definition` (this asserts properties of a new object), and of course `assertion`.

The following elements are considered non-assertional: `symbol` (only a name is declared for an object), `alternative` (here the assertional content is carried by the `assertion` elements referenced in the structure-carrying attributes of `alternative`). For the elements introduced below we will discuss whether they are assertional or not in their context. In a nutshell, only statements introduced by the module ADT (see Chapter 16) will be assertional.

15.4 Mathematical Examples in OMDoc

In mathematical practice examples play a great role, 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 $\mathcal{E}$ as a pair $(\mathcal{W}, \mathbf{A})$, where $\mathcal{W} = (\mathcal{W}_1, \ldots, \mathcal{W}_n)$ is an n-tuple of mathematical objects and $\mathbf{A}$ is an assertion. If $\mathcal{E}$ is an example for a mathematical concept given as an OMDoc symbol $\mathbf{S}$, then $\mathbf{A}$ must be of the form $\mathbf{S}(\mathcal{W}_1, \ldots, \mathcal{W}_n)$.

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

OMDoc specifies this intuition in an `example` element that contains a multilingual `CMP` group for the description and n mathematical objects (the witnesses). It has the attributes `example`

`for` specifying for which concepts or assertions it is an example. This is a reference to a whitespace-separated list of URI references to `symbol`, `definition`, `axiom`, `alternative`, or `assertion` elements.

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

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

`example` elements are considered non-assertional in OMDoc, since the assertional part is carried by the `assertion` element referenced in the `assertion` attribute.

Note that the list of mathematical objects in an `example` element does not represent multiple examples, but corresponds to the argument list of the symbol, they exemplify. In the example below, the symbol for monoid is a three-place relation (see the type declaration in Listing 15.1), so we have three witnesses.

Listing 15.8. An OMDoc representation of a mathematical example

```
<symbol name="strings-over"/>
<definition xml:id="strings.def" for="strings-over">... A* ...</definition>
<symbol name="concat"/>
<definition xml:id="concat.def" for="concat">... :: ...</definition>
<symbol name="empty-string"/>
<definition xml:id="empty-string.def" for="empty-string">... ϵ ...</definition>
...
<assertion xml:id="string.struct.monoid" type="lemma">
  <CMP><xhtml:p>(A*, ::, ϵ) is a monoid.</xhtml:p></CMP>
  <FMP>mon(A*, ::, ϵ)</FMP>
</assertion>
...
<example xml:id="mon.ex1" for="monoid" type="for"
        assertion="string.struct.monoid">
  <CMP><xhtml:p>The set of strings with concatenation is a monoid.</xhtml:p></CMP>
  <OMOBJ>
    <OMA id="nat-strings">
      <OMS cd="strings" name="strings"/>
      <OMS cd="setname1" name="N"/>
    </OMA>
  </OMOBJ>
  <OMOBJ><OMS cd="strings" name="concat"/></OMOBJ>
  <OMOBJ><OMS cd="strings" name="empty-string"/></OMOBJ>
</example>
```

```
<assertion xml:id="monoid.are.groups" type="false−conjecture">
 <CMP><xhtml:p>Monoids are groups.</xhtml:p></CMP>
 <FMP>∀S, o, e.mon(S, o, e) → ∃i.group(S, o, e, i)</FMP>
</assertion>

<example xml:id="mon.ex2" for="#monoids.are.groups" type="against"
         assertion="strings.isnt.group">
  <CMP><xhtml:p>The set of strings with concatenation is not a group.</xhtml:p></CMP>
  <OMOBJ><OMR href="#nat−strings"/></OMOBJ>
  <OMOBJ><OMS cd="strings" name="strings"/></OMOBJ>
  <OMOBJ><OMS cd="strings" name="concat"/></OMOBJ>
  <OMOBJ><OMS cd="strings" name="empty−string"/></OMOBJ>
</example>

<assertion xml:id="strings.isnt.group" type="theorem">
  <CMP><xhtml:p>(A*, ::, ε) is a monoid, but there is no inverse function for it.</xhtml:p></CMP>
</assertion>
```

In Listing 15.8 we show an example of the usage of an `example` element in OMDoc: We declare constructor symbols `strings-over`, that takes an alphabet A as an argument and returns the set A^* of stringss over A, `concat` for strings concatenation (which we will denote by ::), and `empty-string` for the empty string ϵ. Then we state that $\mathcal{W} = (A^*, ::, \epsilon)$ is a monoid in an `assertion` with `xml:id="string.struct.monoid"`. The `example` element with `xml:id="mon.ex1"` in Listing 15.8 is an example for the concept of a monoid, since it encodes the pair $(\mathcal{W}, \mathbf{A})$ where $\mathbf{A}$ is given by reference to the assertion `string.struct.monoid` in the `assertion` attribute. Example `mon.ex2` uses the pair $(\mathcal{W}, \mathbf{A}')$ as a counter-example to the false conjecture `monoids.are.groups` using the assertion `strings.isnt.group` for $\mathbf{A}'$.

15.5 Inline Statements

Note that the infrastructure for statements introduced so far does its best to mark up the interplay of formal and informal elements in mathematical documents, and make explicit the influence of the context and their contribution to it. However, not all statements in mathematical documents can be adequately captured directly. Consider for instance the following situation, which we might find in a typical mathematical textbook.

> **Theorem 3.12**: *In a monoid M the left unit and the right unit coincide, we call it the* **unit** *of M.*

The overt role of this text fragment is that of a mathematical theorem — as indicated by the cue word "**Theorem**", therefore we would be tempted represent it as an `omtext` element with the value `theorem` for the `type` attribute. But the relative clause is clearly a definition (the definiens is even marked in boldface). What we have here is an aggregated verbalization of two mathematical statements. In a simple case like this one, we could represent this as follows:

Listing 15.9. A Simple-Minded Representation of **Theorem 3.12**

```
<assertion type="theorem" style="display=flow">
  <CMP><xhtml:p>In a monoid M, the left unit and the right unit coincide,</xhtml:p></CMP>
</assertion>
<definition for="unit" style="display:flow">
   <CMP><xhtml:p>we call it the <term role="definiendum" name="unit">unit</term> of M</xhtml:p></CMP>
</definition>
```

But this representation remains unsatisfactory: the definition is not part of the theorem, which would really make a difference if the theorem continued after the inline definition. The real problem is that the inline definition is linguistically a phrase-level construct, while the `omtext` element is a discourse-level construct. However, as a phrase-level construct, the inline definition cannot really be gren the context it is presented in (which is the beauty of it; the re-use of context). With the `phrase` element and its `verbalizes`, we can do the following:

Listing 15.10. An Inline Definition

```
<assertion xml:id='unit-unique' type="theorem" >
  <CMP><xhtml:p>In a monoid M, the left unit and the right unit coincide,
    <xhtml:span verbalizes="#unit-def">we call it the unit of M</xhtml:span>.</xhtml:p></CMP>
</assertion>
<symbol name="unit"/>
<definition xml:id="unit-def" for="unit" just-by='#unit-unique'>
  <CMP><xhtml:p>We call the (unique) element of a monoid M that acts as a left
    and right unit the <term role="definiendum" name="unit">unit</term> of M.</xhtml:p></CMP>
</definition>
```

thus we would have the phrase-level markup in the proper place, and we would have an explicit version of the definition which is standalone[5], and we would have the explicit relation that states that the inline definition is an “abbreviation” of the standalone definition.

15.6 Theories as Structured Contexts

OMDoc provides an infrastructure for mathematical theories as first-class objects that can be used to structure larger bodies of mathematics by functional aspects, to serve as a framework for semantically referencing mathematical objects, and to make parts of mathematical developments reusable in multiple contexts. The module ST presented in this chapter introduces a part of this infrastructure, which can already address the first two concerns. For the latter, we need the machinery for complex theories introduced in Chapter 18.

`theory`

Theories are specified by the `theory` element in OMDoc, which has an optional `xml:id` attribute for referencing the theory. Furthermore, the `theory` element can have the `cdbase` attribute that allows to specify the `cdbase`

[5] Purists could use the CSS attribute `style` on the `definition` element with value `display:none` to hides it from the document; it might also be placed into another document altogether

this theory uses for disambiguation on `om:OMS` elements (see Section 13.1 for a discussion). Additional information about the theory like a title or a short description can be given in the `metadata` element. After this, any top-level OMDoc element can occur, including the theory-constitutive elements introduced in Sections 15.1 and 15.2, even `theory` elements themselves. Note that theory-constitutive elements may *only* occur in `theory` elements.

omgroup

Note that theories can be structured like documents e.g. into sections and the like (see Section 11.5 for a discussion) via the `omgroup` element.

Element	Attributes		D	Content
	Req.	Optional	C	
`theory`		`xml:id, class, style, cdbase, cdversion, cdrevision, cdstatus, cdurl, cdreviewdate`	+	`(`⟪*top+thc*⟫ `| imports)*`
`imports`	`from`	`id, type, class, style`	+	
where ⟪*top+thc*⟫ stands for top-level and theory-constitutive elements				

Fig. 15.8. Theories in OMDoc

15.6.1 Simple Inheritance

`theory` elements can contain `imports` elements (mixed in with the top-level ones) to specify inheritance: The main idea behind structured theories and specification is that not all theory-constitutive elements need to be explicitly stated in a theory; they can be inherited from other theories. Formally, the set of theory-constitutive elements in a theory is the union of those that are explicitly specified and those that are imported from other theories. This has consequences later on, for instance, these are available for use in proofs. See Section 17.2 for details on availability of assertional statements in proofs and justifications.

imports

The meaning of the `imports` element is determined by two attributes:

`from` The value of this attribute is a URI reference that specifies the **source theory**, i.e. the theory we import from. The current theory (the one specified in the parent of the `imports` element, we will call it the **target theory**) inherits the constitutive elements from the source theory.

`type` This optional attribute can have the values `global` and `local` (the former is assumed, if the attribute is absent): We call constitutive elements **local** to the current theory, if they are explicitly defined as children, and else **inherited**. A **local import** (an `imports` element with `type="local"`) only imports the local elements of the source theory, a global import also the inherited ones.

The meaning of nested `theory` elements is given in terms of an implicit imports relation: The inner theory imports from the outer one. Thus

```
<theory xml:id="a.thy">
  <symbol name="aa"/>
  <theory xml:id="b.thy">
    <symbol name="cc"/>
    <definition xml:id="cc.def" for="cc" type="simple">
      <OMOBJ><OMS cd="a.thy" name="aa"/></OMOBJ>
    </definition>
  </theory>
</theory>
```

is equivalent to

```
<theory xml:id="a.thy"><symbol name="aa"/></theory>
<theory xml:id="b.thy">
  <imports from="#a.thy" type="global"/>
  <symbol name="cc"/>
  <definition xml:id="cc.def" for="cc" type="simple">
    <OMOBJ><OMS cd="a.thy" name="aa"/></OMOBJ>
  </definition>
</theory>
```

In particular, the symbol `cc` is visible only in theory `b.thy`, not in the rest of theory `a.thy` in the first representation. Note that the inherited elements of the current theory can themselves be inherited in the source theory. For instance, in the Listing 15.12 the `left-inv` is the only local axiom of the theory `group`, which has the inherited axioms `closed`, `assoc`, `left-unit`.

In order for this import mechanism to work properly, the inheritance relation, i.e. the relation on theories induced by the `imports` elements, must be acyclic. There is another, more subtle constraint on the inheritance relation concerning multiple inheritance. Consider the situation in Listing 15.11: here theories `A` and `B` import theories with `xml:id="mythy"`, but from different URIs. Thus we have no guarantee that the theories are identical, and semantic integrity of the theory `C` is at risk. Note that this situation might in fact be totally unproblematic, e.g. if both URIs point to the same document, or if the referenced documents are identical or equivalent. But we cannot guarantee this by content markup alone, we have to forbid it to be safe.

Listing 15.11. Problematic Multiple Inheritance

```
<theory xml:id="A">
  <imports from="http://red.com/theories.omdoc#mythy"/>
</theory>
<theory xml:id="B">
  <imports from="http://blue.org/cd/all.omdoc#mythy"/>
</theory>
<theory xml:id="C"><imports from="#A"/><imports from="#B"/></theory>
```

Let us now formulate the constraint carefully, the **base URI** of an XML document is the URI that has been used to retrieve it. We adapt this to OMDoc theory elements: the base URI of an imported theory is the URI declared in the `cdbase` attribute of the `theory` element (if present) or the base URI of the document which contains it[6]. For theories that are imported

[6] Note that the base URI of the document is sufficient, since a valid OMDoc document cannot contain more than one `theory` element for a given `xml:id`

along a chain of global imports, which include relative URIs, we need to employ URI normalization to compute the effective URI. Now the constraint is that any two imported theories that have the same value of the `xml:id` attribute must have the same base URI. Note that this does not imply a global unicity constraint for `xml:id` values of `theory` elements, it only means that the mapping of theory identifiers to URIs is unambiguous in the dependency cone of a theory.

In Listing 15.12 we have specified three algebraic theories that gradually build up a theory of groups importing theory-constitutive statements (symbols, axioms, and definitions) from earlier theories and adding their own content. The theory `semigroup` provides symbols for an operation `op` on a base set `set` and has the axioms for closure and associativity of `op`. The theory of monoids imports these without modification and uses them to state the `left-unit` axiom. The theory `monoid` then proceeds to add a symbol `neut` and an axiom that states that it acts as a left unit with respect to `set` and `op`. The theory `group` continues this process by adding a symbol `inv` for the function that gives inverses and an axiom that states its meaning.

Listing 15.12. A Structured Development of Algebraic Theories in OMDOC

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

<theory xml:id="monoid">
  <imports from="#semigroup"/>
  <symbol name="neut"/><symbol name="setstar"/>
  <axiom xml:id="left−unit">
    <CMP><xhtml:p>neut is a left unit for op.</xhtml:p></CMP><FMP>∀x ∈ set.op(x, neut) = x</FMP>
  </axiom>
  <definition xml:id="setstar.def" for="setstar" type="implicit">
    <CMP><xhtml:p>·* subtracts the unit from a set </xhtml:p></CMP><FMP>∀S.S* = S\{unit}</FMP>
  </definition>
</theory>

<theory xml:id="group">
  <imports from="#monoid"/>
  <symbol name="inv"/>
  <axiom xml:id="left−inv">
    <CMP><xhtml:p>For every X ∈ set there is an inverse inv(X) wrt. op.</xhtml:p></CMP>
  </axiom>
</theory>
```

The example in Listing 15.12 shows that with the notion of theory inheritance it is possible to re-use parts of theories and add structure to specifications. For instance it would be very simple to define a theory of Abelian semigroups by adding a commutativity axiom.

The set of symbols, axioms, and definitions available for use in proofs in the importing theory consists of the ones directly specified as `symbol`, `axiom`, and `definition` elements in the target theory itself (we speak of **local** axioms and definitions in this case) and the ones that are inherited from the source theories via `imports` elements. Note that these symbols, axioms, and definitions (we call them **inherited**) can consist of the local ones in the source theories and the ones that are inherited there.

The local and inherited symbols, definitions, and axioms are the only ones available to mathematical statements and proofs. If a symbol is not available in the home theory (the one given by the dominating `theory` element or the one specified in the `theory` attribute of the statement), then it cannot be used since its semantics is not defined.

15.6.2 OMDoc Theories as Content Dictionaries

In Chapter 13, we have introduced the OpenMath and Content-MathML representations for mathematical objects and formulae. One of the central concepts there was the notion that the representation of a symbol includes a pointer to a document that defines its meaning. In the original OpenMath standard, these documents are identified as OpenMath content dictionaries, the MathML recommendation is not specific. In the examples above, we have seen that OMDoc documents can contain definitions of mathematical concepts and symbols, thus they are also candidates for "defining documents" for symbols. By the OpenMath2 standard [Bus+04] suitable classes of OMDoc documents can act as OpenMath content dictionaries (we call them **OMDoc content dictionaries**; see Subsection 22.3.2). The main distinguishing feature of OMDoc content dictionaries is that they include `theory` elements with symbol declarations (see Section 15.2) that act as the targets for the pointers in the symbol representations in OpenMath and Content-MathML. The theory name specified in the `xml:id` attribute of the `theory` element takes the place of the `CDname` defined in the OpenMath content dictionary.

Furthermore, the URI specified in the `cdbase` attribute is the one used for disambiguation on `om:OMS` elements (see Section 13.1 for a discussion).

For instance the symbol declaration in Listing 15.1 can be referenced as

```
<OMS cd="elAlg" name="monoid" cdbase="http://omdoc.org/algebra.omdoc"/>
```

if it occurs in a theory for elementary algebra whose `xml:id` attribute has the value `elAlg` and which occurs in a resource with the URI `http://omdoc.org/algebra.omdoc` or if the `cdbase` attribute of the `theory` element has the value `http://omdoc.org/algebra.omdoc`.

To be able to act as an OpenMath2 content dictionary format, OMDoc must be able to express content dictionary metadata (see Listing 5.1 for an example). For this, the `theory` element carries some optional attributes that allow to specify the administrative metadata of OpenMath content dictionaries.

The `cdstatus` attribute specifies the **content dictionary status**, which can take one of the following values: `official` (i.e. approved by the OpenMath Society), `experimental` (i.e. under development and thus liable to change), `private` (i.e. used by a private group of OpenMath users) or `obsolete` (i.e. only for archival purposes). The attributes `cdversion` and `cdrevision` jointly specify the **content dictionary version number**, which consists of two parts, a major **version** and a **revision**, both of which are non-negative integers. For details between the relation between content dictionary status and versions consult the OpenMath standard [Bus+04].

Furthermore, the `theory` element can have the following attributes:

`cdbase` for the content dictionary base which, when combined with the content dictionary name, forms a unique identifier for the content dictionary. It may or may not refer to an actual location from which it can be retrieved.

`cdurl` for a valid URL where the source file for the content dictionary encoding can be found.

`cdreviewdate` for the **review date** of the content dictionary, i.e. the date until which the content dictionary is guaranteed to remain unchanged.

16

Abstract Data Types (Module ADT)

Most specification languages for mathematical theories support definition mechanisms for sets that are inductively generated by a set of constructors and recursive functions on these under the heading of **abstract data type**s. Prominent examples of abstract data types are natural numbers, lists, trees, etc. The module ADT presented in this chapter extends OMDoc by a concise syntax for abstract data types that follows the model used in the Casl (Common Abstract Specification Language [Mos04]) standard.

Conceptually, an abstract data type declares a collection of symbols and axioms that can be used to construct certain mathematical objects and to group them into sets. For instance, the Peano axioms (see Figure 15.1) introduce the symbols 0 (the number zero), s (the successor function), and $\mathbb{N}$ (the set of natural numbers) and fix their meaning by five axioms. These state that the set $\mathbb{N}$ contains exactly those objects that can be constructed from 0 and s alone (these symbols are called **constructor symbol**s and the representations **constructor term**s). Optionally, an abstract data type can also declare **selector symbol**s, for (partial) inverses of the constructors. In the case of natural numbers the predecessor function is a selector for s: it "selects" the argument n, from which a (non-zero) number $s(n)$ has been constructed.

Following Casl we will call sets of objects that can be represented as constructor terms **sort**s. A sort is called **free**, iff there are no identities between constructor terms, i.e. two objects represented by different constructor terms can never be equal. The sort $\mathbb{N}$ of natural numbers is a free sort. An example of a sort that is not free is the theory of finite sets given by the constructors $\emptyset$ and the set insertion function ι , since the set $\{a\}$ can be obtained by inserting a into the empty set an arbitrary (positive) number of times; so e.g. $\iota(a,\emptyset) = \iota(a,\iota(a,\emptyset))$. This kind of sort is called **generated**, since it only contains elements that are expressible in the constructors. An abstract data type is called **loose**, if it contains elements besides the ones generated by the constructors. We consider free sorts more **strict** than generated ones, which in turn are more strict than loose ones.

In OMDoc, we use the `adt` element to specify abstract data types possibly `adt`

Element	Attributes		D	Content
	Req.	Optional	C	
adt		xml:id, class, style, parameters	+	sortdef+
sortdef	name	type, role, scope, class, style	+	(constructor | insort)*, recognizer?
constructor	name	type, scope, class, style	+	argument*
argument			+	type, selector?
insort	for		–	
selector	name	type, scope, role, total, class, style	+	EMPTY
recognizer	name	type, scope, role, class, style	+	

Fig. 16.1. Abstract data types in OMDoc

consisting of multiple sorts. It is a theory-constitutive statement and can only occur as a child of a `theory` element (see Section 15.1 for a discussion). An `adt` element contains one or more `sortdef` elements that define the sorts and specify their members and it can carry a `parameters` attribute that contains a whitespace-separated list of parameter variable names. If these are present, they declare type variables that can be used in the specification of the new sort and constructor symbols see Section **??** for an example.

We will use an augmented representation of the abstract data type of natural numbers as a running example for introduction of the functionality added by the ADT module; Listing 16.1 contains the listing of the OMDoc encoding. In this example, we introduce a second sort $\mathbb{P}$ for positive natural numbers to make it more interesting and to pin down the type of the predecessor function.

`sortdef`

A `sortdef` element is a highly condensed piece of syntax that declares a sort symbol together with the constructor symbols and their selector symbols of the corresponding sort. It has a required `name` attribute that specifies the symbol name, an optional `type` attribute that can have the values `free`, `generated`, and `loose` with the meaning discussed above. A `sortdef` element contains a set of `constructor` and `insort` elements. The latter are empty elements which refer to a sort declared elsewhere in a `sortdef` with their `for` attribute: An `insort` element with `for="`⟪*URI*⟫`#`⟪*name*⟫`"` specifies that all the constructors of the sort ⟪*name*⟫ are also constructors for the one defined in the parent `sortdef`. Furthermore, the type of a sort given by a `sortdef` element can only be as strict as the types of any sorts included by its `insort` children.

`constructor`

`insort`

Listing 16.1 introduces the sort symbols `pos-nats` (positive natural numbers) and `nats` (natural numbers) , the symbol names are given by the required `name` attribute. Since a constructor is in general an n-ary function, a `constructor` element contains n `argument` children that specify the argument sorts of this function along with possible selector functions. The argument sort is given as the first child of the `argument` element: a `type` element as described in Subsection 15.2.3. Note that n may be 0 and thus the constructor element may not have `argument` children (see for instance the `constructor` for `zero`

`argument`

in Listing 16.1). The first `sortdef` element there introduces the constructor symbol `succ@Nat` for the successor function. This function has one argument, which is a natural number (i.e. a member of the sort `nats`).

`selector`

Sometimes it is convenient to specify the inverses of a constructors that are functions. For this OMDoc offers the possibility to add an empty `selector` element as the second child of an `argument` child of a `constructor`. The required attribute `name` specifies the symbol name, the optional `total` attribute of the `selector` element specifies whether the function represented by this symbol is total (value `yes`) or partial (value `no`). In Listing 16.1 the `selector` element in the first `sortdef` introduces a selector symbol for the successor function `succ`. As `succ` is a function from `nats` to `pos-nats`, `pred` is a total function from `pos-nats` to `nats`.

`recognizer`

Finally, a `sortdef` element can contain a `recognizer` child that specifies a symbol for a predicate that is true, iff its argument is of the respective sort. The name of the predicate symbol is specified in the required `name` attribute. Listing 16.1 introduces such a **recognizer predicate** as the last child of the `sortdef` element for the sort `pos-nats`.

Note that the `sortdef`, `constructor`, `selector`, and `recognizer` elements define symbols of the name specified by their `name` element in the theory that contains the `adt` element. To govern the visibility, they carry the attribute `scope` (with values `global` and `local`) and the attribute `role` (with values `type`, `sort`, `object`).

Listing 16.1. The natural numbers using `adt` in OMDoc

```
<theory xml:id="Nat">
  <adt xml:id="nat-adt">
    <metadata>
      <dc:title>Natural Numbers as an Abstract Data Type.</dc:title>
      <dc:description>The Peano axiomatization of natural numbers.</dc:description>
    </metadata>

    <sortdef name="pos-nats" type="free">
      <metadata>
        <dc:description>The set of positive natural numbers.</dc:description>
      </metadata>
      <constructor name="succ">
        <metadata><dc:description>The successor function.</dc:description></metadata>
        <argument>
          <type><OMOBJ><OMS cd='Nat' name="nats"/></OMOBJ></type>
          <selector name="pred" total="yes">
            <metadata><dc:description>The predecessor function.</dc:description></metadata>
          </selector>
        </argument>
      </constructor>
      <recognizer name="positive">
        <metadata>
          <dc:description>
            The recognizer predicate for positive natural numbers.
          </dc:description>
        </metadata>
      </recognizer>
    </sortdef>

    <sortdef name="nats" type="free">
      <metadata><dc:description>The set of natural numbers</dc:description></metadata>
```

```
    <constructor name=”zero”>
      <metadata><dc:description>The number zero.</dc:description></metadata>
    </constructor>
    <insort for=”#pos−nats”/>
  </sortdef>
 </adt>
</theory>
```

To summarize Listing 16.1: The abstract data type `nat-adt` is free and defines two sorts `pos-nats` and `nats` for the (positive) natural numbers. The positive numbers (`pos-nats`) are generated by the successor function (which is a constructor) on the natural numbers (all positive natural numbers are successors). On `pos-nats`, the inverse `pred` of `succ` is total. The set `nats` of all natural numbers is defined to be the union of `pos-nats` and the constructor `zero`. Note that this definition implies the five well-known Peano Axioms: the first two specify the constructors, the third and fourth exclude identities between constructor terms, while the induction axiom states that `nats` is generated by `zero` and `succ`. The document that contains the `nat-adt` could also contain the symbols and axioms defined implicitly in the `adt` element explicitly as `symbol` and `axiom` elements for reference. These would then carry the `generated-from` attribute with value `nat-adt`.

17

Representing Proofs (Module PF)

Proofs form an essential part of mathematics and modern sciences. Conceptually, a **proof** is a representation of uncontroversial evidence for the truth of an assertion.

The question of what exactly constitutes a proof has been controversially discussed (see e.g. [BC01]). The clearest (and most radical) definition is given by theoretical logic, where a proof is a sequence, or tree, or directed acyclic graph (DAG) of applications of inference rules from a formally defined logical calculus, that meets a certain set of well-formedness conditions. There is a whole zoo of logical calculi that are optimized for various applications. They have in common that they are extremely explicit and verbose, and that the proofs even for simple theorems can become very large. The advantage of having formal and fully explicit proofs is that they can be very easily verified, even by simple computer programs. We will come back to this notion of proof in Section 17.4.

In mathematical practice the notion of a proof is more flexible, and more geared for consumption by humans: any line of argumentation is considered a proof, if it convinces its readers that it could in principle be expanded to a formal proof in the sense given above. As the expansion process is extremely tedious, this option is very seldom carried out explicitly. Moreover, as proofs are geared towards communication among humans, they are given at vastly differing levels of abstraction. From a very informal proof idea for the initiated specialist of the field, who can fill in the details herself, down to a very detailed account for skeptics or novices which will normally be still well above the formal level. Furthermore, proofs will usually 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 (or mathematical vernacular).

Let us consider a proof and its context (Figure 17.1) as it could be found in a typical elementary math. textbook, only that we have numbered the proof steps for referencing convenience. Figure 17.1 will be used as a running example throughout this chapter.

Theorem: *There are infinitely many prime numbers.*
Proof: We need to prove that the set P of all prime numbers is not finite.

1. We proceed by assuming that P is finite and reaching a contradiction.
2. Let P be finite.
3. Then $P = \{p_1, \ldots, p_n\}$ for some p_i.
4. Let $q \stackrel{def}{=} p_1 \cdots p_n + 1$.
5. Since for each $p_i \in P$ we have $q > p_i$, we conclude $q \notin P$.
6. We prove the absurdity by showing that q is prime:
7. For each $p_i \in P$ we have $q = p_i k + 1$ for some natural number k, so p_i can not divide q;
8. q must be prime as P is the set of all prime numbers.
9. Thus we have contradicted our assumption (2)
10. and proven the assertion. □

Fig. 17.1. A Theorem with a Proof.

Since proofs can be marked up on several levels, we will introduce the OMDoc markup for proofs in stages: We will first concentrate on proofs as structured texts, marking up the discourse structure in example Figure 17.1. Then we will concentrate on the justifications of proof steps, and finally we will discuss the scoping and hierarchical structure of proofs.

The development of the representational infrastructure in OMDoc has a long history: From the beginning the format strived to allow structural semantic markup for textbook proofs as well as accommodate a wide range of formal proof systems without over-committing to a particular system. However, the proof representation infrastructure from Version 1.1 of OMDoc turned out not to be expressive enough to represent the proofs in the Helm library [Asp+01]. As a consequence, the PF module has been redesigned [AKSC03] as part of the MoWGLI project [AK02]. The current version of the PF module is an adaptation of this proposal to be as compatible as possible with earlier versions of OMDoc. It has been validated by interpreting it as an implementation of the $\overline{\lambda}\mu\tilde{\mu}$ calculus [SC06] proof representation calculus.

17.1 Proof Structure

In this section, we will concentrate on the structure of proofs apparent in the proof text and introduce the OMDoc infrastructure needed for marking up this aspect. Even if the proof in Figure 17.1 is very short and simple, we can observe several characteristics of a typical mathematical proof. The proof starts with the thesis that is followed by nine main "steps" (numbered from 1 to 10). A very direct representation of the content of Figure 17.1 is given in Listing 17.1.

Listing 17.1. An OMDoc Representation of Figure 17.1.

```
<assertion xml:id="a1">
  <CMP><xhtml:p>There are infinitely many prime numbers.</xhtml:p></CMP>
</assertion>
<proof xml:id="p" for="#a1">
  <omtext xml:id="intro">
    <CMP><xhtml:p>We need to prove that the set P of all prime numbers is not finite.</xhtml:p></CMP>
  </omtext>
  <derive xml:id="d1">
    <CMP><xhtml:p>We proceed by assuming that P is finite and reaching a contradiction.</xhtml:p></CMP>
    <method>
      <proof xml:id="p1">
        <hypothesis xml:id="h2"><CMP><xhtml:p>Let P be finite.</xhtml:p></CMP></hypothesis>
        <derive xml:id="d3">
          <CMP><xhtml:p>Then P = {p_1, ..., p_n} for some p_i.</xhtml:p></CMP>
          <method><premise xref="#h2"/></method>
        </derive>
        <symbol name="q"/>
        <definition xml:id="d4" for="q" type="informal">
          <CMP><xhtml:p>Let q =def p_1 ··· p_n + 1</xhtml:p></CMP>
        </definition>
        <derive xml:id="d5">
          <CMP><xhtml:p> Since for each p_i ∈ P we have q > p_i, we conclude q ∉ P.</xhtml:p></CMP>
        </derive>
        <omtext xml:id="c6">
          <CMP><xhtml:p>We prove the absurdity by showing that q is prime:</xhtml:p></CMP>
        </omtext>
        <derive xml:id="d7">
          <CMP><xhtml:p>For each p_i ∈ P we have q = p_i k + 1 for some
            natural number k, so p_i can not divide q;</xhtml:p></CMP>
          <method><premise xref="#d4"/></method>
        </derive>
        <derive xml:id="d8">
          <CMP><xhtml:p>q must be prime as P is the set of all prime numbers.</xhtml:p></CMP>
          <method><premise xref="#d7"/></method>
        </derive>
        <derive xml:id="d9">
          <CMP><xhtml:p>Thus we have contradicted our assumption</xhtml:p></CMP>
          <method><premise xref="#d5"/><premise xref="#d8"/></method>
        </derive>
      </proof>
    </method>
  </derive>
  <derive xml:id="d10" type="conclusion">
    <CMP><xhtml:p>This proves the assertion.</xhtml:p></CMP>
  </derive>
</proof>
```

Proofs are specified by `proof` elements in OMDoc that have the optional attributes `xml:id` and `theory` and the required attribute `for`. The `for` at-

`proof`

tribute points to the assertion that is justified by this proof (this can be an `assertion` element or a `derive` proof step (see below), thereby making it possible to specify expansions of justifications and thus hierarchical proofs). Note that there can be more than one proof for a given assertion.

Element	Attributes		D	Content
	Req.	Optional	C	
proof	for	theory, xml:id, class, style	+	(omtext | derive | hypothesis | symbol | definition)*
proofobject	for	xml:id, class, style, theory	+	CMP*, (OMOBJ |m:math |legacy)
hypothesis		xml:id, class, style, inductive	–	CMP*, FMP*
derive		xml:id, class, style, type	–	CMP*, FMP*, method?
method		xref	–	(OMOBJ |m:math |legacy | premise | proof | proofobject)*
premise	xref	rank	–	EMPTY

Fig. 17.2. The OMDoc Proof Elements

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

`omtext` OMDoc allows this element to allow for intermediate text in proofs that does not have to have a logical correspondence to a proof step, but e.g. guides the reader through the proof. Examples for this are remarks by the proof author, e.g. an explanation why some other proof method will not work. We can see another example in Listing 17.1 in lines 5-7, where the comment gives a preview over the course of the proof.

`derive` elements specify normal proof steps that derive a new claim from already known ones, from assertions or axioms in the current theory, or from the assumptions of the assertion that is under consideration in the proof. See for example lines $12ff$ in Listing 17.1 for examples of `derive` proof steps that only state the local assertion. We will consider the specification of justifications in detail in Section 17.2 below. The `derive` element carries an optional `xml:id` attribute for identification and an optional `type` to single out special cases of proofs steps.

`derive`

The value `conclusion` is reserved for the concluding step of a proof[1], i.e. the one that derives the assertion made in the corresponding theorem.

The value `gap` is used for proof steps that are not justified (yet): we call them **gap steps**. Note that the presence of gap steps allows OMDoc to specify incomplete proofs as proofs with gap steps.

[1] As the argumentative structure of the proof is encoded in the justification structure to be detailed in Section 17.2, the concluding step of a proof need not be the last child of a proof element.

`hypothesis` elements allow to specify local assumptions that allow the hypothetical reasoning discipline needed for instance to specify proof by contradiction, by case analysis, or simply to show that A implies B, by assuming A and then deriving B from this local hypothesis. The scope of an hypothesis extends to the end of the `proof` element containing it. In Listing 17.1 the classification of step 2 from Figure 17.1 as the `hypothesis` element `h2` forces us to embed it into a `derive` element with a `proof` grandchild, making a structure apparent that was hidden in the original. `hypothesis`

An important special case of hypothesis is the case of "inductive hypothesis", this can be flagged by setting the value of the attribute `inductive` to `yes`; the default value is `no`.

`symbol`/`definition` These elements allow to introduce new local symbols that are local to the containing `proof` element. Their meaning is just as described in Section 15.2, only that the role of the `axiom` element described there is taken by the `hypothesis` element. In Listing 17.1 step 4 in the proof is represented by a `symbol`/`definition` pair. Like in the `hypothesis` case, the scope of this symbol extends to the end of the `proof` element containing it.

These elements contain an informal (natural language) representation of the proof step in a multilingual `CMP` group and possibly an `FMP` element that gives a formal representation of the claim made by this proof step. A `derive` element can furthermore contain a `method` element that specifies how the assertion is derived from already-known facts (see the next section for details). All of the proof step elements have an optional `xml:id` attribute for identification and the CSS attributes.

As we have seen above, the content of any proof step is essentially a Gentzen-style sequent; see Listing 17.3 for an example. 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.

17.2 Proof Step Justifications

So far we have only concerned ourselves with the linear structure of the proof, we have identified the proof steps and classified them by their function in the proof. A central property of the `derive` elements is that their content (the local claim) follows from statements that we consider true. These can be earlier steps in the proof or general knowledge. To convince the reader of a proof, the steps are often accompanied with a **justification**. This can be given either by a logical inference rule or higher-level evidence for the truth of the claim. The evidence can consist in a proof method that can be used to prove the assertion, or in a separate subproof, that could be presented if the consumer was unconvinced. Conceptually, both possibilities are equivalent, since the method can be used to compute the subproof (called its **expansion**). Justifications are represented in OMDoc by the `method` children of `derive` elements[2] (see Listing 17.2 for an example):

`method`

The `method` element contains a structural specification of the justification of the claim made in the `FMP` of a `derive` element. So the `FMP` together with the `method` element jointly form the counterpart to the natural language content of the `CMP` group, they are sibling to: The `FMP` formalizes the local claim, and the `method` stands for the justification. In Listing 17.2 the formula in the `CMP` element corresponds to the claim, whereas the part "By ..., we have" is the justification. In other words, a `method` element specifies a proof method or inference rule with its arguments that justifies the assertion made in the `FMP` elements. It has an optional `xref` attribute whose target is an OMDoc definition of an inference rule or proof method.[3] A method may have `om:OMOBJ`, `m:math`, `legacy`, `premise`, `proof`, and `proofobject`[4] children. These act as parameters to the method, e.g. for the repeated universal instantiation method in Listing 17.2 the parameters are the terms to instantiate the bound variables.

`premise`

The `premise` elements are used to refer to already established assertions: other proof steps or statements (given as `assertion`, `definition`, or `axiom` elements) the method was applied to to obtain the local claim of the proof step. The `premise` elements are empty and carry the required attribute `xref`, which contains the URI of the assertion. Thus the `premise` elements specify the DAG structure of the proof. Note that even if we do not mark up the method in a justification (e.g. if it is unknown or obvious) it can still make sense to structure the argument in `premise` elements. We have done so in Listing 17.1 to make the dependencies of the argumentation explicit.

If a `derive` step is a logically (or even mathematically) complex step, an expansion into sub-steps can be specified in a `proof` or `proofobject` element embedded into the justifying `method` element. An embedded proof allows us to specify generic markup for the hierarchic structure of proofs. Expansions

---

[2] The structural and formal justification elements discussed in this section are derived from hierarchical data structures developed for semi-automated theorem proving (satisfying the logical side). They allow natural language representations at every level (allowing for natural representation of mathematical vernacular at multiple levels of abstraction). This proof representation (see [Ben+97] for a discussion and pointers) is a DAG of nodes which represent the proof steps.

[3] At the moment OMDoc does not provide markup for such objects, so that they should best be represented by `symbol`s with `definition` where the inference rule is explained in the `CMP` (see the lower part of Listing 17.2), and the `FMP` holds a content representation for the inference rule, e.g. using the content dictionary [**CD:inference-rules**]. A good enhancement is to encapsulate system-specific encodings of the inference rules in `private` or `code` elements and have the `xref` attribute point to these.

[4] This object is an alternative representation of certain proofs, see Section 17.4.

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 [Pau94] or NuPrL [Con+86]. Thus, proof nodes may have justifications at multiple levels of abstraction in an hierarchical proof data structure. Thus the `method` elements allow to augment the linear structure of the proof by a tree/DAG-like secondary structure given by the `premise` links. Due to the complex hierarchical structure of proofs, we cannot directly utilize the tree-like structure provided by XML, but use cross-referencing. The `derive` step in Listing 17.2 represents an inner node of the proof tree/DAG with three children (the elements with identifiers `A2`, `A4`, and `A5`).

Listing 17.2. A `derive` Proof Step

```
<proof xml:id="proof.2.1.2.proof.D2.1" for="#assertion.2.1.2">
  ...
  <derive xml:id="D2.1">
    <CMP><xhtml:p>By <oref xref="#A2"/>, <ref type="cite" xref="#A4"/>, and
        <oref xref="#A5"/> we have z + (a + (−a)) = (z + a) + (−a).</xhtml:p></CMP>
    <FMP>z + (a + (−a)) = (z + a) + (−a)</FMP>
    <method xref="nk−sorts.omdoc#NK−Sorts.forallistar">
      <OMOBJ><OMV name="z"/></OMOBJ>
      <OMOBJ><OMV name="a"/></OMOBJ>
      <OMOBJ>−a</OMOBJ>
      <premise xref="#A2"/><premise xref="#A4"/><premise xref="#A5"/>
    </method>
  </derive>
  ...
</proof>
...
<theory xml:id="NK−Sorts">
  <metadata>
    <dc:title>Natural Deduction for Sorted Logic</dc:title>
  </metadata>

  <symbol name="forallistar">
    <metadata>
      <dc:description>Repeated Universal Instantiation></dc:description>
    </metadata>
  </symbol>
  <definition xml:id="forallistar.def" for="forallistar" type="informal">
    <CMP><xhtml:p>Given n parameters, the inference rule ∀I* instantiates
      the first n universal quantifications in the antecedent with them.</xhtml:p></CMP>
  </definition>
  ...
</theory>
```

In OMDoc the `premise` elements must reference proof steps in the current proof or statements (`assertion` or `axiom` elements) in the scope of the current theory: A statement is **in scope of** the current theory, if its home theory is the current theory or imported (directly or indirectly) by the current theory.

Furthermore note that a proof containing a `premise` element is not self-contained evidence for the validity of the `assertion` it proves. Of course it is only evidence for the validity at all (we call such a proof grounded), if all the statements that are targets of `premise` references have grounded proofs themselves[5] and the reference relation does not contain cycles. A grounded proof can be made self-contained by inserting the target statements as `derive` elements before the referencing `premise` and embedding at least one `proof` into the `derive` as a justification.

Let us now consider another proof example (Listing 17.3) to fortify our intuition.

Listing 17.3. An OMDoc Representation of a Proof by Cases

```
<assertion xml:id="t1" theory="sets">
  <CMP><xhtml:p>If a ∈ U or a ∈ V, then a ∈ U ∪ V.</xhtml:p></CMP>
  <FMP>
    <assumption xml:id="t1_a">a ∈ U ∨ a ∈ V</assumption>
    <conclusion xml:id="t1_c">a ∈ U ∪ V</conclusion>
  </FMP>
</assertion>
<proof xml:id="t1_p1" for="#t1" theory="sets">
  <omtext xml:id="t1_p1_m1">
    <CMP><xhtml:p> We prove the assertion by a case analysis.</xhtml:p></CMP>
  </omtext>
  <derive xml:id="t1_p1_l1">
    <CMP><xhtml:p>If a ∈ U, then a ∈ U ∪ V.</xhtml:p></CMP>
    <FMP>
      <assumption xml:id="t1_p1_l1_a">a ∈ U</assumption>
      <conclusion xml:id="t1_p1_l1_c">a ∈ U ∪ V</conclusion>
    </FMP>
    <method xref="sk.omdoc#SK.by_definition">∪</method>
  </derive>
  <derive xml:id="t1_p1_l2">
    <CMP><xhtml:p>If a ∈ V, then a ∈ U ∪ V.</xhtml:p></CMP>
    <FMP>
      <assumption xml:id="t1_p1_l2_a">a ∈ V</assumption>
      <conclusion xml:id="t1_p1_l2_c">a ∈ U ∪ V</conclusion>
    </FMP>
    <method xref="sk.omdoc#SK.by_definition">∪</method>
  </derive>
  <derive xml:id="t1_p1_c">
    <CMP><xhtml:p> We have considered both cases, so we have a ∈ U ∪ V.</xhtml:p></CMP>
  </derive>
</proof>
```

This proof is in **sequent style**: The statement of all local claims is in self-contained `FMP`s that mark up the statement in `assumption`/`conclusion` form, which makes the logical dependencies explicit. In this example we use inference rules from the calculus "SK",Gentzen's sequent calculus for classical first-order logic [Gen35], which we assume to be formalized in a theory `SK`. Note that local assumptions from the `FMP` should not be referenced outside the `derive` step they were made in. In effect, the `derive` element serves as a grouping device for local assumptions.

Note that the same effect as embedding a `proof` element into a `derive` step can be obtained by specifying the `proof` at top-level and using the optional `for` attribute to refer to the identity of the enclosing proof step (given by its

[5] For `assertion` targets this requirement is obvious. Obviously, `axiom`s do not need proofs, but certain forms of definitions need well-definedness proofs (see Subsection 15.2.4). These are included in the definition of a grounded proof.

optional `xml:id` attribute), we have done this in the proof in Listing 17.4, which expands the `derive` step with identifier `t1_p1_l1` in Listing 17.3.

Listing 17.4. An External Expansion of Step `t_1_p1_l1` in Listing 17.3

```
<definition xml:id="union.def" for="union">
  <OMOBJ>∀P, Q, x.x ∈ P ∪ Q ⇔ x ∈ P ∨ x ∈ Q</OMOBJ>
</definition>

<proof xml:id="t1_p1_l1.exp" for="#t1_p1_l1">
  <derive xml:id="t1_p1_l1.d1">
    <FMP>
      <assumption xml:id="t1_p1_l1.d1.a">a ∈ U</assumption>
      <conclusion xml:id="t1_p1_l1.d1.c">a ∈ U</conclusion>
    </FMP>
    <method xref="sk.omdoc#SK.axiom"/>
  </derive>
  <derive xml:id="t1_p1_l1.l1.d2">
    <FMP>
      <assumption xml:id="t1_p1_l1.d2.a">a ∈ U</assumption>
      <conclusion xml:id="t1_p1_l1.d2.c">a ∈ U ∨ a ∈ V</conclusion>
    </FMP>
    <method xref="sk.omdoc#SK.orR"><premise xref="#t1_p1_l1.d1"/></method>
  </derive>
  <derive xml:id="t1_p1_l1.d3">
    <FMP>
      <assumption xml:id="t1_p1_l1.d3.a">a ∈ U ∨ a ∈ V</assumption>
      <conclusion xml:id="t1_p1_l1.d3.c">a ∈ U ∪ V</conclusion>
    </FMP>
    <method xref="sk.omdoc#SK.definition−rl">U, V, a
      <premise xref="#unif.def"/>
    </method>
  </derive>
  <derive xml:id="t1_p1_l1.d4">
    <FMP>
      <assumption xml:id="t1_p1_l1.d3.a">a ∈ U</assumption>
      <conclusion xml:id="t1_p1_l1.d3.c">a ∈ U ∪ V</conclusion>
    </FMP>
    <method xref="sk.omdoc#SK.cut">
      <premise xref="#t1_p1_l1.d2"/>
      <premise xref="#t1_p1_l1.d3"/>
    </method>
  </derive>
</proof>
```

17.3 Scoping and Context in a Proof

Unlike the sequent style proofs we discussed in the last section, many informal proofs use the **natural deduction style** [Gen35], which allows to reason from local assumptions. We have already seen such hypotheses as `hypothesis` elements in Listing 17.1. The main new feature is that hypotheses can be introduced at some point in the proof, and are discharged later. As a consequence, they can only be used in certain parts of the proof. The hypothesis is inaccessible for inference outside the nearest ancestor `proof` element of the `hypothesis`.

Let us now reconsider the proof in Figure 17.1. Some of the steps (2, 3, 4, 5, 7) leave the thesis unmodified; these are called **forward reasoning** or

bottom-up proof steps, since they are used to derive new knowledge from the available one with the aim of reaching the conclusion. Some other steps (1, 6) are used to conclude the (current) thesis by opening new subproofs, each one characterized with a new local thesis. These steps are called **backward reasoning** or **top-down proof step**s steps, since they are used to reduce a complex problem (proving the thesis) to several simpler problems (the subproofs). In our example, both backward reasoning steps open just one new subproof: Step 1 reduces the goal to proving that the finiteness of P implies a contradiction; step 5 reduces the goal to proving that q is prime.

Step 2 is used to introduce a new hypothesis, whose scope extends from the point where it is introduced to the end of the current subproof, covering also all the steps inbetween and in particular all subproofs that are introduced in these. In our example the scope of the hypothesis that P is finite (step 2 in Figure 17.1) are steps 3 – 8. In an inductive proof, for instance, the scope of the inductive hypothesis covers only the proof of the inductive step and not the proof of the base case (independently from the order adopted to present them to the user).

Step 4 is similar, it introduces a new symbol q, which is a local declaration that has scope over lines 4 – 9. The difference between a hypothesis and a local declaration is that the latter is used to introduce a variable as a new element in a given set or type, whereas the former, is used to locally state some property of the variables in scope. For example, *"let n be a natural number"* is a declaration, while *"suppose n to be a multiple of 2"* is a hypothesis. The introduction of a new hypothesis or local declaration should always be justified by a proof step that discharges it. In our example the declaration P is discharged in step 10. Note that in contrast to the representation in Listing 17.1 we have chosen to view step 6 in Figure 17.1 as a top-down proof step rather than a proof comment.

To sum up, every proof step is characterized by a current thesis and a *context*, which is the set of all the local declarations, hypotheses, and local definitions in scope. Furthermore, a step can either introduce a new hypothesis, definition, or declaration or can just be a forward or backward reasoning step. It is a forward reasoning `derive` step if it leaves the current thesis as it is. It is a backward reasoning `derive` step if it opens new subproofs, each one characterized by a new thesis and possibly a new context.

Listing 17.5. A top-down Representation of the Proof in Figure 17.1.

```
<assertion xml:id="a1">
  <CMP><xhtml:p>There are infinitely many prime numbers.</xhtml:p></CMP>
</assertion>
<proof for="#a1">
  <omtext xml:id="c0">
    <CMP><xhtml:p>We need to prove that the set P of all prime numbers is not finite.</xhtml:p></CMP>
  </omtext>
  <derive xml:id="d1">
    <CMP><xhtml:p> We proceed by assuming that P is finite and reaching a contradiction.</xhtml:p></CMP>
    <method xref="nk.omdoc#NK.by-contradiction">
      <proof>
        <hypothesis xml:id="h2"><CMP><xhtml:p>Let P be finite.</xhtml:p></CMP></hypothesis>
```

```
        <derive xml:id="d3"><CMP><xhtml:p>Then $P = \{p_1, \ldots, p_n\}$ for some $n$</xhtml:p></CMP></derive>
        <symbol name="q"/>
        <definition xml:id="d4" for="q" type="informal">
          <CMP><xhtml:p>Let $q \stackrel{def}{=} p_1 \cdots p_n + 1$</xhtml:p></CMP>
        </definition>
        <derive xml:id="d5a">
          <CMP><xhtml:p>For each $p_i \in P$ we have $q > p_i$</xhtml:p></CMP>
          <method xref="#Trivial"><premise xref="#d4"/></method>
        </derive>
        <derive xml:id="d5b">
          <CMP><xhtml:p>$q \notin P$</xhtml:p></CMP>
          <method xref="#Trivial"><premise xref="#d5"/></method>
        </derive>
        <derive xml:id="d6">
          <CMP><xhtml:p>We show absurdity by showing that $q$ is prime</xhtml:p></CMP>
          <FMP>⊥</FMP>
          <method xref="#Contradiction">
            <premise xref="#d5b"/>
            <proof>
              <derive xml:id="d7a">
                <CMP><xhtml:p>
                  For each $p_i \in P$ we have $q = p_i k + 1$ for a given natural number $k$.
                </xhtml:p></CMP>
                <method xref="#By_Definition"><premise xref="#d1"/></method>
              </derive>
              <derive xml:id="d7b">
                <CMP><xhtml:p>Each $p_i \in P$ does not divide $q$</xhtml:p></CMP>
              </derive>
              <derive xml:id="d8">
                <CMP><xhtml:p>$q$ is prime</xhtml:p></CMP>
                <method xref="#Trivial">
                  <premise xref="#h2"/>
                  <premise xref="#p4"/>
                </method>
              </derive>
            </proof>
          </method>
        </derive>
      </proof>
    </method>
  </derive>
</proof>
```

`proof` elements are considered to be non-assertional in OMDoc, since they do not make assertions about mathematical objects themselves, but only justify such assertions. The assertional elements inside the proofs are governed by the scoping mechanisms discussed there, so that using them in a context where assertional elements are needed, can be forbidden.

17.4 Formal Proofs as Mathematical Objects

`proofobject`

In OMDoc, the notion of fully formal proofs is accommodated by the `proofobject` element. In logic, the term **proof object** is used for term representations for formal proofs via the Curry/Howard/DeBruijn Isomorphism (see e.g. [Tho91] for an introduction and Figure 17.3 for an example). λ-terms are among the most succinct representations of calculus-level proofs as they only document the inference rules. Since they are fully formal, they are very difficult to read and need specialized proof presentation systems for human consumption. In proof objects inference rules are represented as mathematical symbols, in our example in Figure 17.3 we have assumed a theory `PL0ND` for the calculus of natural deduction in propositional logic which provides the necessary symbols (see Listing 17.6).

The `proofobject` element contains an optional multilingual group of `CMP` elements which describes the formal proof as well as a proof object which can be an `om:OMOBJ`, `m:math`, or `legacy` element.

$$\dfrac{\dfrac{\dfrac{[A \wedge B]}{B}\wedge E_r \qquad \dfrac{[A \wedge B]}{A}\wedge E_l}{B \wedge A}\wedge I}{A \wedge B \Rightarrow B \wedge A}\Rightarrow I$$

```
<proofobject xml:id="ac.p" for="#and−comm">
 <metadata>
  <dc:description>
   Assuming A ∧ B we have B and A
   from which we can derive B ∧ A.
  </dc:description>
 </metadata>
 <OMOBJ>
  <OMBIND id="andcom.pf">
   <OMS cd="PL0ND" name="impliesI"/>
   <OMBVAR>
    <OMATTR>
     <OMATP>
      <OMS cd="PL0ND" name="type"/>
      A ∧ B
     </OMATP>
     <OMV name="X"/>
    </OMATTR>
   </OMBVAR>
   <OMA>
    <OMS cd="PL0ND" name="andI"/>
    <OMA>
     <OMA>
      <OMS cd="PL0ND" name="andEr"/>
      <OMV name="X"/>
     </OMA>
     <OMA>
      <OMS cd="PL0ND" name="andEl"/>
      <OMV name="X"/>
     </OMA>
    </OMA>
   </OMA>
  </OMBIND>
 </OMOBJ>
</proofobject>
```

The schema on the left shows the proof as a natural deduction proof tree, the OMDoc representation gives the proof object as a λ term. This term would be written as the following term in traditional (mathematical) notation: $\Rightarrow I(\lambda X : A \wedge B. \wedge I(\wedge E_r(X), \wedge E_l(X)))$

Fig. 17.3. A Proof Object for the Commutativity of Conjunction

Note that using OMDoc symbols for inference rules and mathematical objects for proofs reifies them to the object level and allows us to treat them at par with any other mathematical objects. We might have the following theory for natural deduction in propositional logic as a reference target for the second inference rule in Figure 17.3.

Listing 17.6. A Theory for Propositional Natural Deduction

```
<theory xml:id=”PL0ND”>
  <metadata>
    <dc:description>The Natural Deduction Calculus for Propositional Logic</dc:description>
  </metadata>
  ...
  <symbol name=”andI”>
    <metadata><dc:subject>Conjunction Introduction</dc:subject></metadata>
    <type system=”prop−as−types”>A → B → (A ∧ B)</type>
  </symbol>

  <definition xml:id=”andI.def” for=”andi”>
    <CMP><xhtml:p>Conjunction introduction, if we can derive A and B,
      then we can conclude A ∧ B.</xhtml:p></CMP>
  </definition>
  ...
</theory>
```

In particular, it is possible to use a `definition` element to define a derived inference rule by simply specifying the proof term as a definiens:

```
<symbol name=”andcom”>
  <metadata><dc:description>Commutativity for ∧</dc:description></metadata>
  <type system=”prop−as−types”>(A ∧ B) → (B ∧ A)</type>
</symbol>
<definition xml:id=”andcom.def” for=”#andcom” type=”simple”>
  <OMOBJ><OMR href=”#andcom.pf”/></OMOBJ>
</definition>
```

Like `proof`s, `proofobject`s elements are considered to be non-assertional in OMDoc, since they do not make assertions about mathematical objects themselves, but only justify such assertions.

18

Complex Theories (Module CTH)

In Section 15.6 we have presented a notion of theory and inheritance that is sufficient for simple applications like content dictionaries that informally (though presumably rigorously) define the static meaning of symbols. Experience in e.g. program verification has shown that this infrastructure is insufficient for large-scale developments of formal specifications, where reusability of formal components is the key to managing complexity. For instance, for a theory of rings we cannot simply inherit the same theory of monoids as both the additive and multiplicative structure.

In this chapter, we will generalize the inheritance relation from Section 15.6 to that of "structures", also called "theory morphisms" or "theory interpretations" elsewhere [Far93]. This infrastructure allows to structure a collection of theories into a complex theory graph that particularly supports modularization and reuse of parts of specifications and theories. This gives rise to the name "complex theories" of the OMDoc module.

Element	Attributes		D	Content
	Required	Optional	C	
`theory`		`xml:id, class, style`	+	`(`《*top-level*》` | imports | structure)*`
`structure`	`from`	`xml:id, class, style`	+	`morphism?`
`morphism`		`xml:id, base, class, style, hiding, type, consistency, exhaustivity`	−	`requation+, measure?, ordering?`

Fig. 18.1. Complex Theories in OMDoc

18.1 Structures: Inheritance via Translations

Literal inheritance of symbols is often insufficient to re-use mathematical structures and theories efficiently. Consider for instance the situation in the elementary algebraic hierarchy: for a theory of rings, we should be able to inherit the additive group structure from the theory `group` of groups and the structure of a multiplicative monoid from the theory `monoid`: A ring is a set R together with two operations $+$ and $*$, such that $(R,+)$ is a group with unit 0 and inverse operation $-$ and $(R^*,*)$ is a monoid with unit 1 and base set $R^* := \{r \in R \middle| r \neq 0\}$. Using the literal inheritance regime introduced so far, would lead us into a duplication of efforts as we have to define theories for semigroups and monoids for the operations $+$ and $*$ (see Figure 18.2).

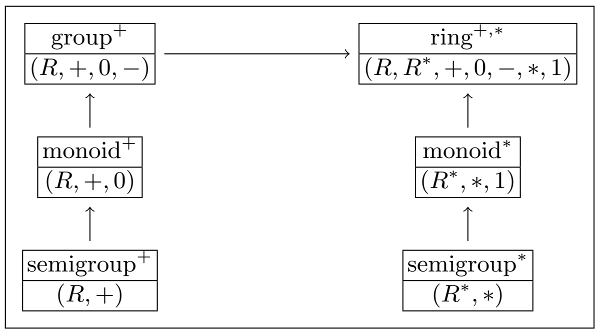


Fig. 18.2. A Theory of Rings via Simple Inheritance

This problem[1] can be alleviated by allowing theory inheritance via translations. Instead of literally inheriting the symbols and axioms from the source theory, we involve a symbol mapping function (we call this a **morphism**) in the process. This function maps source formulae (i.e. built up exclusively from symbols visible in the source theory) into formulae in the target theory by translating the source symbols.

Figure 18.3 shows a theory graph that defines a theory of rings by importing the monoid axioms via the morphism σ. With this translation, we do not have to duplicate the `monoid` and `semigroup` theories and can even move the definition of $\cdot^*$ operator into the theory of monoids, where it intuitively belongs[2].

Formally, we extend the notion of inheritance given in Section 15.6 by allowing a target theory to import another a source theory **via a morphism**: Let $\mathcal{S}$ be a theory with theory-constitutive elements[3] $t_1, \ldots, t_n$ and $\sigma: \mathcal{S} \to \mathcal{T}$

[1] which seems negligible in this simple example, but in real life, each instance of multiple inheritance leads to a *multiplication* of all dependent theories, which becomes an exponentially redundant management nightmare.

[2] On any monoid $M = (S, \circ, e)$, we have the $\cdot^*$ operator, which converts a set $S \subseteq M$ in to $S^* := \{r \in S \middle| r \neq e\}$

[3] which may in turn be inherited from other theories

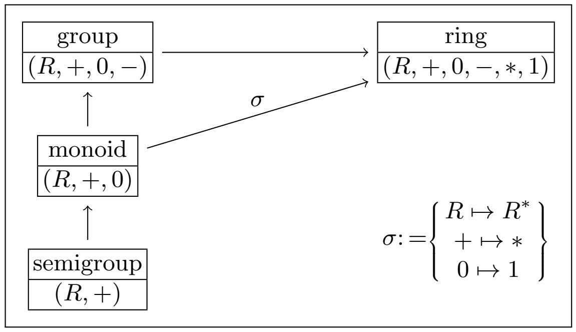


Fig. 18.3. A Theory of Rings via Morphisms

a morphism, if we declare that $\mathcal{T}$ imports $\mathcal{S}$ via σ, then $\mathcal{T}$ **inherit**s the theory-constitutive statements $\sigma(t_i)$ from $\mathcal{S}$. For instance, the theory of rings inherits the axiom $\forall x.x + 0 = x$ from the theory of monoids as $\sigma(\forall x.x + 0 = x) = \forall x.x * 1 = x$.

`morphism`

To specify the formula mapping function, module CTH extends the `imports` element by allowing it to have a child element `morphism`, which specifies a formula mapping by a set of recursive equations using the `requation` element described in Section 15.2. The optional attribute `type` allows to specify whether the function is really recursive (value `recursive`) or pattern-defined (value `pattern`). As in the case of the `definition` element, termination of the defined function can be specified using the optional child elements `measure` and `ordering`, or the optional attributes `uniqueness` and `existence`, which point to uniqueness and existence assertions. Consistency and exhaustivity of the recursive equations are specified by the optional attributes `consistency` and `exhaustivity`.

Listing 18.1 gives the OMDoc representation of the theory graph in Figure 18.3, assuming the theories in Listing 15.12.

Listing 18.1. A Theory of Rings by Inheritance Via Renaming

```
<theory xml:id=”ring”>
  <symbol name=”times”/><symbol name=”one”/>
  <structure name=”add” from=”?group”/>
  <structure name=”mult” from=”?monoid”>
    <morphism>
      <requation>
        <OMOBJ><OMS cd=”monoid” name=”set”/></OMOBJ>
        <OMOBJ>
          <OMA><OMS cd=”monoid” name=”setstar”/>
            <OMS cd=”semigroup” name=”set”/>
          </OMA>
        </OMOBJ>
      </requation>
      <requation>
        <OMOBJ><OMS cd=”monoid” name=”op”/></OMOBJ>
        <OMOBJ><OMS cd=”ring” name=”times”/></OMOBJ>
      </requation>
      <requation>
```

```
      <OMOBJ><OMS cd=”monoid” name=”neut”/></OMOBJ>
      <OMOBJ><OMS cd=”ring” name=”one”/></OMOBJ>
    </requation>
  </morphism>
</imports>
<axiom xml:id=”ring.distribution”>
  <CMP><xthml:p><OMOBJ><OMS cd=”semigroup” name=”op”/></OMOBJ> distributes over
    <OMOBJ><OMS cd=”ring” name=”times”/></OMOBJ> </xhtml:p>
  </CMP>
</axiom>
</theory>
```

To conserve space and avoid redundancy, OMDoc morphisms need only specify the values of symbols that are translated; all other symbols are inherited literally. Thus the set of symbols inherited by an `imports` element consists of the symbols of the source theory that are not in the domain of the morphism. In our example, the symbols R, $+$, 0, $-$, $*$, 1 are visible in the theory of rings (and any other symbols the theory of semigroups may have inherited). Note that we do not have a name clash from multiple inheritance.

Finally, it is possible to hide symbols from the source theory by specifying them in the `hiding` attribute. The intended meaning is that the underlying signature mapping is defined (total) on all symbols in the source theory except on the hidden ones. This allows to define symbols that are local to a given theory, which helps achieve data protection. Unfortunately, there is no simple interpretation of hiding in the general case in terms of formula translations, see [Mos04; MAH06] for details. The definition of hiding used there is more general. The variant used here arises as the special case where the hiding morphism, which goes against the import direction, is an inclusion; then the symbols that are not in the image are the hidden ones. If we restrict ourselves to hiding defined symbols, then the situation becomes simpler to understand: A morphism that hides a (defined) symbol s will translate the theory-constitutive elements of the source theory by expanding definitions. Thus s will not be present in the target theory, but all the contributions of the theory-constitutive elements of the source theory will have been inherited. Say, we want to define the concept of a sorting function, i.e. a function that — given a list L as input — returns a returns a permutation L' of L that is ordered. In the situation depicted in Figure 18.4, we would the concept of an ordering function (a function that returns a permutation of the input list that is ordered) with the help of predicates `perm` and `ordered`. Since these are only of interest in the context of the definition of the latter, they would typically be hidden in order to refrain from polluting the name space.

As morphisms often contain common prefixes, the `morphism` element has an optional `base` attribute, which points to a chain of morphisms, whose composition is taken to be the base of this morphism. The intended meaning is that the new morphism coincides as a function with the base morphism, wherever the specified pattern do not match, otherwise their corresponding values take precedence over those in the base morphism. Concretely, the `base` contains a whitespace-separated list of URI references to `structure` elements.

Note that the order of the references matters: they are ordered in order of the path in the local chain, i.e if we have `base="#⟪ref1⟫...#⟪refn⟫"` there must be theory inclusions σ_i with `xml:id="⟪refi⟫"`, such that the target theory of σ_{i-1} is the source theory of σ_i, and such that the source theory of σ_1 and the target theory of σ_n are the same as those of the current view.

Finally, the CTH module adds two the optional attributes `conservativity` and `conservativity-just` to the `imports` element for stating and justifying conservativity (see the discussion below).

18.2 Views

We have seen that inheritance via morphisms provides a powerful mechanism for structuring and re-using theories and contexts. It turns out that the distinguishing feature of theory morphisms is that all theory-constitutive elements of the source theory are valid in the target theory (possibly after translation). This can be generalized to obtain even more structuring relations and thus possibilities for reuse among theories. Before we go into the OMDoc infrastructure, we will briefly introduce the mathematical model (see e.g. [Hut00] for details).

A **view** from a **source theory** $\mathcal{S}$ to a **target theory** $\mathcal{T}$ is a mapping σ from $\mathcal{S}$ objects[4] to those of $\mathcal{T}$, such that for every theory-constitutive statement $\mathbf{S}$ of $\mathcal{S}$, $\sigma(\mathbf{S})$ is provable in $\mathcal{T}$ (we say that $\sigma(\mathbf{S})$ is a **$\mathcal{T}$-theorem**).

In OMDoc, we weaken this logical property to a structural one: We say that a theory-constitutive statement $\mathbf{S}$ in theory $\mathcal{S}$ is **structurally included** in theory $\mathcal{T}$ via σ, if there is an assertional statement $\mathbf{T}$ in $\mathcal{T}$, such that the content of $\mathbf{T}$ is $\sigma(\mathbf{S})$. Note that strictly speaking, σ is only defined on formulae, so that if a statement $\mathbf{S}$ is only given by a `CMP`, $\sigma(\mathbf{S})$ is not defined. In such cases, we assume $\sigma(\mathbf{S})$ to contain a `CMP` element containing suitably translated mathematical vernacular. In this view, a **structural theory inclusion** from $\mathcal{S}$ to $\mathcal{T}$ is a morphism $\sigma\colon\mathcal{S}\to\mathcal{T}$, such that every theory-constitutive element is structurally included in $\mathcal{T}$.

Note that an `imports` element in a theory $\mathcal{T}$ with source theory $\mathcal{S}$ as discussed in Section 18.1 induces a theory inclusion from $\mathcal{S}$ into $\mathcal{T}$[5] (the theory-constitutive statements of $\mathcal{S}$ are accessible in $\mathcal{T}$ after translation and are therefore structurally included trivially). We call this kind of theory inclusion **definitional**, since it is a theory inclusion by virtue of the definition of the target theory. For all other theory inclusions (we call them **postulated theory inclusions**), we have to establish the theory inclusion property by

[4] Mathematical objects that can be represented using the only symbols of the source theory $\mathcal{S}$.

[5] Note that in contrast to the inheritance relation induced by the `imports` elements the relation induced by general theory inclusions may be cyclic. A cycle just means that the theories participating in it are semantically equivalent.

proving the translations of the theory-constitutive statements of the source theory (we call these translated formulae **proof obligation**).

The benefit of a theory inclusion is that all theorems, proofs, and proof methods of the source theory can be used (after translation) in the target theory (see Section 18.4). Obviously, the transfer approach only depends on the theorem inclusion property, and we can extend its utility by augmenting the theory graph by more theory morphisms than just the definitional ones (see [FGT93] for a description of the IMPS theorem proving system that makes heavy use of this idea). We use the infrastructure presented in this chapter to structure a collection of theories as a graph — the **theory graph** — where the nodes are theories and the links are theory inclusions (definitional and postulated ones).

We call a theory inclusion $\sigma{:}\mathcal{S} \to \mathcal{T}$ **conservative**, iff $\mathbf{A}$ is already a $\mathcal{S}$-theorem for all $\mathcal{T}$-theorems of the from $\sigma(\mathbf{A})$. If the morphism σ is the identity, then this means the local axioms in $\mathcal{T}$ only affect the local symbols of $\mathcal{T}$, and do not the part inherited from $\mathcal{S}$. In particular, conservative extensions of consistent theories cannot be inconsistent. For instance, if all the local theory-constitutive elements in $\mathcal{T}$ are symbol declarations with definitions, then conservativity is guaranteed by the special form of the definitions. We can specify conservativity of a theory inclusion via the `conservativity` attribute. The values `conservative` and `definitional` are used for the two cases discussed above. There is a third value: `monomorphism`, which we will not explain here, but refer the reader to [MAH06].

`view`

OMDOC implements the concept of view in the top-level `view` element. It has the required attributes `from` and `to`, which point to the source- and target theories and contains a `morphism` child element as described above to define the translation function. A subsequent (possibly empty) set of `obligation` elements can be used to mark up proof obligations for the theory-constitutive elements of the source theory.

`obligation`

An `obligation` is an empty element whose `assertion` attribute points to an `assertion` element that states that the theory-constitutive statement specified by the `induced-by` (translated by the morphism in the parent `view`) is provable in the target theory. Note that a `view` element must contain `obligation` elements for all theory-constitutive elements (inherited or local) of the source theory to be correct.

Listing 18.2 shows a theory inclusion from the theory `group` defined in Listing 15.12 to itself. The morphism just maps each element of the base set to its inverse. A good application for this kind of theory morphism is to import claims for symmetric (e.g. with respect to the function `inv`, which serves as an involution in `group`) cases via this theory morphism to avoid explicitly having to prove them (see Section 18.4).

Listing 18.2. A Theory Inclusion for Groups

```
<assertion xml:id="conv.assoc">∀x, y, z ∈ M.z ∘ (y ∘ x) = (z ∘ y) ∘ x</assertion>
<assertion xml:id="conv.closed" theory="semigroup">∀x, y ∈ M.y ∘ x ∈ M</assertion>
<assertion xml:id="left.unit" theory="monoid">∀x ∈ M.e ∘ x = x</assertion>
```

```
<assertion xml:id="conv.inv" theory="group">∀x, y ∈ M.x ∘ x^{−1} = e</assertion>
<view xml:id="grp−conv−grp" from="#group" to="#group">
  <morphism><requation>X ∘ Y ⇝ Y ∘ X</requation></morphism>
  <obligation assertion="#conv.closed" induced−by="#closed.ax"/>
  <obligation assertion="#conv.assoc" induced−by="#assoc.ax"/>
  <obligation assertion="#left.unit" induced−by="#unit.ax"/>
  <obligation assertion="#conv.inv" induced−by="#inv.ax"/>
</view>
```

18.3 Local- and Required Theory Inclusions

In some situations, we need to pose well-definedness conditions on theories, e.g. that a specification of a program follows a certain security model, or that a parameter theory used for actualization satisfies the assumptions made in the formal parameter theory; (see Chapter 6 for a discussion). If these conditions are not met, the theory intuitively does not make sense. So rather than simply stating (or importing) these assumptions as theory-constitutive statements — which would make the theory inconsistent, when they are not met — they can be stated as well-definedness conditions. Usually, these conditions can be posited as theory inclusions, so checking these conditions is a purely structural matter, and comes into the realm of OMDoc's structural methods.

OMDoc provides the empty `inclusion` element for this purpose. It can occur anywhere as a child of a `theory` element and its `via` attribute points to a theory inclusion, which is required to hold in order for the parent theory to be well-defined. `inclusion`

If we consider for instance the situation in Figure 18.4[6]. There we have a theory `OrdList` of lists that is generic in the elements (which is assumed to be a totally ordered set, since we want to talk about ordered lists). We want to to instantiate `OrdList` by applying it to the theory `NatOrd` of natural numbers and obtain a theory `NatOrdList` of lists of natural numbers by importing the theory `OrdList` in `NatOrdList`. This only makes sense, if `NatOrd` is a totally ordered set, so we add an `inclusion` element in the statement of theory `NatOrdList` that points to a theory inclusion of `TOSet` into `OrdNat`, which forces us to verify the axioms of `TOSet` in `OrdNat`.

Furthermore note, that the inclusion of `OrdList` into `NatOrdList` should not include the `TOSet` axioms on orderings, since this would defeat the purpose of making them a precondition to well-definedness of the theory `NatOrdList`. Therefore OMDoc follows the "development graph model" put forward in [Hut00] and generalizes the notion of theory inclusions even further: A formula mapping between theories $\mathcal{S}$ and $\mathcal{T}$ is called a **local theory inclusion** or **axiom inclusion**, if the theory inclusion property holds for the local theory-constitutive statements of the source theory. To distinguish this from the notion of a proper theory inclusion — where the theory inclusion property holds for all theory constitutive statements of $\mathcal{S}$ (even the inherited

[6] This example is covered in detail in Chapter 6.

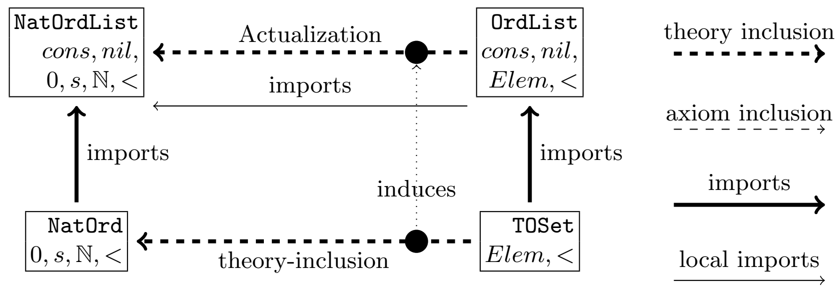


Fig. 18.4. A Structured Specification of Lists (of Natural Numbers)

ones) — we call the latter one **global**. Of course all global theory inclusions are also local ones, so that the new notion is a true generalization. Note that the structural inclusions of an axiom inclusion are not enough to justify translated source theorems in the target theory.

To allow for a local variant of inheritance, the CTH module adds an attribute `type` to the `imports` element. This can take the values `global` (the default) and `local`. In the latter case, only the theory-constitutive statements that are local to the source theory are imported.

`axiom-inclusion`

Furthermore, the CTH module introduces the `axiom-inclusion` element for local theory inclusions. This has the same attributes as `view`: `from` to specify source theory, `to` for the target theory. It also allows `obligation` elements as children.

18.4 Induced Assertions and Expositions

The main motivation of theory inclusions is to be able to transport mathematical statements from the source theory to the target theory. In OMDoc, this operation can be made explicit by the attributes `generated-from` and `generated-via` that the module CTH adds to all mathematical statements. On a statement **T**, the second attribute points to a theory inclusion σ whose target is (imported into the) current theory, the first attribute points to a statement **S** in that theory which is of the same type (i.e. has the same OMDoc element name) as **T**. The content of **T** must be (equivalent to) the content of **S** translated by the morphism of σ.

In the context of the theory inclusion in Listing 18.2, we might have the following situation:

Listing 18.3. Translating a Statement via a Theory Inclusion

```
<assertion xml:id="foo" type="theorem">...</assertion>
<proof xml:id="foo.pf" for="#foo">...</proof>
<assertion xml:id="target" induced-by="#foo" induced-via="#grp-conv-grp">
  ...
</assertion>
```

Here, the second assertion is induced by the first one via the theory inclusion in Listing 18.2, the statement of the theorem is about the inverses. In particular, the proof of the second theorem comes for free, since it can also be induced from the proof of the first one.

In particular we see that in OMDoc documents, not all statements are automatically generated by translation e.g. the proof of the second assertion is not explicitly stated. Mathematical knowledge management systems like knowledge bases might choose to do so, but at the document level we do not mandate this, as it would lead to an explosion of the document sizes. Of course we could cache the transformed proof giving it the same "cache attribute state".

Note that not only statements like assertions and proofs can be translated via theory inclusions, but also whole documents: Say that we have course materials for elementary algebra introducing monoids and groups via left units and left inverses, but want to use examples and exercises from a book that introduces them using right units and right inverses. Assuming that both are formalized in OMDoc, we can just establish a theory morphism much like the one in Listing 18.2. Then we can automatically translate the exercises and examples via this theory inclusion to our own setting by just applying the morphism to all formulae in the text[7] and obtain exercises and examples that mesh well with our introduction. Of course there is also a theory inclusion in the other direction, which is an inverse, so our colleague can reuse our course materials in his right-leaning setting.

Another example is the presence of different normalization factors in physics or branch cuts in elementary complex functions. In both cases there is a plethora of definitions, which all describe essentially the same objects (see e.g. [Bra+02] for an overview over the branch cut situation). Reading materials that are based on the "wrong" definition is a nuisance at best, and can lead to serious errors. Being able to adapt documents by translating them from the author theory to the user theory by a previously established theory morphism can alleviate both.

Mathematics and science are full of such situations, where objects can be viewed from different angles or in different representations. Moreover, no single representation is "better" than the other, since different views reveal or highlight different aspects of the object (see [KK06a] for a systematic account). Theory inclusions seem uniquely suited to formalize the structure of different views in mathematics and their interplay, and the structural markup for theories in OMDoc seems an ideal platform for offering added-value services that

[7] There may be problems, if mathematical statements are verbalized; this can currently not be translated directly, since it would involve language processing tools much beyond the content processing tools described in this book. For the moment, we assume that the materials are written in a controlled subset of mathematical vernacular that avoids these problems.

feed on these structures without committing to a particular formalization or foundation of mathematics.

19

Notation and Presentation (Module PRES)

The main difference of OMDoc 1.3 is that it uses the notation system developed in [Mül10; KMR08]. This system is already supported by the JOMDoc system [Jom].

20

Auxiliary Elements (Module EXT)

Up to now, we have been mainly concerned with providing elements for marking up the inherent structure of mathematical knowledge in mathematical statements and theories. Now, we interface OMDoc documents with the Internet in general and mathematical software systems in particular. We can thereby generate presentations from OMDoc documents where formulae, statements or even theories that are active components that can directly be manipulated by the user or mathematical software systems. We call these documents **active documents**. For this we have to solve two problems: an abstract interface for calls to external (web) services[1] and a way of storing application-specific data in OMDoc documents (e.g. as arguments to the system calls).

The module EXT provides a basic infrastructure for these tasks in OMDoc. The main purpose of this module is to serve as an initial point of entry. We envision that over time, more sophisticated replacements will be developed driven by applications.

Element	Attributes		D	Content
	Req.	Optional	C	
`private`		`xml:id, for, theory, requires, type, reformulates, class, style`	+	`CMP*, data+`
`code`		`xml:id, for, theory, requires, type, class, style`	+	`CMP*, input?, output?, effect?, data+`
`input`		`xml:id, style, class`	+	`CMP*, FMP*`
`output`		`xml:id, style, class`	+	`CMP*, FMP*`
`effect`		`xml:id, style, class`	+	`CMP*, FMP*`
`data`		`format, href, size, original, pto, pto-version`	−	`<![CDATA[...]]>`

Fig. 20.1. The OMDoc Auxiliary Elements for Non-XML Data

[1] Compare Chapter 9 in the OMDoc Primer.

20.1 Non-XML Data and Program Code in OMDoc

The representational infrastructure for mathematical knowledge provided by OMDoc is sufficient as an output- and library format for mathematical software systems like computer algebra systems, theorem provers, or theory development systems. In particular, having a standardized output- and library format like OMDoc will enhance system interoperability, and allows to build and deploy general storage and library management systems (see Section **??** for an OMDoc example). In fact this was one of the original motivations for developing the format.

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

`private`

`code`

For this kind of data, OMDoc provides the `private` and `code` elements. As the name suggests, the latter is intended for program code[2] and the former for system-specific data that is not program code.

The attributes of these elements are almost identical and contain metadata information identifying system requirements and relations to other OMDoc elements. We will first describe the shared attributes and then describe the elements themselves.

`xml:id` for identification.

`theory` specifies the mathematical theory (see Section 15.6) that the data is associated with.

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

`requires` specifies other data this element depends upon as a whitespace-separated list of URI references. This allows to factor private data into smaller parts, allowing more flexible data storage and retrieval which is useful for program code or private data that relies on program code. Such data can be broken up into procedures and the call-hierarchy can be encoded in `requires` attributes. With this information, a storage application based on OMDoc can always communicate a minimal complete code set to the requesting application.

[2] There is a more elaborate proposal for treating program code in the OMDoc arena at [Koh], which may be integrated into OMDoc as a separate module in the future, for the moment we stick to the basic approach.

`reformulates` (`private` only) specifies a set of OMDoc elements whose knowledge content is reformulated by the `private` element as a whitespace-separated list of URI references. For instance, the knowledge in the assertion in Listing 20.1 can be used as an algebraic simplification rule in the Analytica theorem prover [Cla+03] based on the Mathematica computer algebra system.

The `private` and `code` elements contain an optional `metadata` element and a set of `data` elements that contain or reference the actual data.

Listing 20.1. Reformulating Mathematical Knowledge

```
<assertion xml:id="ALGX0">
  <CMP><xhtml:p>If a, b, c, d are numbers, then we have a + b(c + d) = a + bc + bd.</xhtml:p></CMP>
</assertion>
<private xml:id="alg-expr-1" pto="Analytica" reformulates="ALGX0">
 <data format="mathematica-5.0">
   <![CDATA[SIMPLIFYRULES[a_ + b_*(c_ + d_) :> a + b*c + b*d /; NumberQ[b]]]]>
 </data>
</private>
```

data

The `data` element contains the data in a `CDATA` section. Its `pto` attribute contains a whitespace-separated list of URI references which specifies the set of systems to which the data are related. The intention of this field is that the data is visible to all systems, but should only manipulated by a system that is mentioned here. The `pto-version` attribute contains a whitespace-separated list of version number strings; this only makes sense, if the value of the corresponding `pto` is a singleton. Specifying this may be necessary, if the data or even their format change with versions.

If the content of the `data` element is too large to store directly in the OMDoc or changes often, then the `data` element can be augmented by a link, specified by a URI reference in the `href` attribute. If the `data` element is non-empty and there is a `href`[3], then the optional attribute `original` specifies whether the `data` content (value `local`) or the external resource (value `external`) is the original. The optional `size` attribute can be used to specify the content size (if known) or the resource identified in the `href` attribute. The `data` element has the (optional) attribute `format` to specify the format the data are in, e.g. `image/jpeg` or `image/gif` for image data, `text/plain` for text data, `binary` for system-specific binary data, etc. It is good practice to use the MIME types [FB96] for this purpose whenever applicable. Note that in a `private` or `code` element, the `data` elements must differ in their `format` attribute. Their order carries no meaning.

In Listing 20.2 we use a `private` element to specify data for an image[4] in various formats, which is useful in a content markup format like OMDoc as the transformation process can then choose the most suitable one for the target.

---

[3] e.g. if the `data` content serves as a cache for the data at the URI, or the `data` content fixes a snapshot of the resource at the URI

[4] actually Figure 4.1 from Chapter 4

Listing 20.2. A `private` Element for an Image

```
<private xml:id="legacy">
  <metadata>
    <dc:title>A fragment of Bourbaki's Algebra</dc:title>
    <dc:creator role="trl">Michael Kohlhase</dc:creator>
    <dc:date action="created">2002-01-03T0703</dc:date>
    <dc:description>A fragment of Bourbaki's Algebra</dc:description>
    <dc:source>Nicolas Bourbaki, Algebra, Springer Verlag 1974</dc:source>
    <dc:type>Text</dc:type>
  </metadata>
  <data format="application/x-latex" href="legacy.tex"/>
  <data format="image/jpg" href="legacy.jpeg"/>
  <data format="application/postscript" href="legacy.ps"/>
  <data format="application/pdf" href="legacy.pdf"/>
</private>
```

`input`

`output`

`effect`

The `code` element is used for embedding pieces of program code into an OMDoc document. It contains the documentation elements `input`, `output`, and `effect` that specify the behavior of the procedure defined by the code fragment. The `input` element describes the structure and scope of the input arguments, `output` the outputs produced by calling this code on these elements, and `effect` any side effects the procedure may have. They contain a multilingual group of `CMP` elements with an optional `FMP` group for a formal description. The latter may be used for program verification purposes. If any of these elements are missing it means that we may not make any assumptions about them, not that there are no inputs, outputs or effects. For instance, to specify that a procedure has no side-effects we need to specify something like

```
<effect><CMP><xhtml:p>None.</xhtml:p></CMP></effect>
```

These documentation elements are followed by a set of `data` elements that contain or reference the program code itself. Listing 20.5 shows an example of a `code` element used to store Java code for an applet.

Listing 20.3. The Program Code for a Java Applet

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

20.2 Applets and External Objects in OMDoc

Web-based text markup formats like HTML have the concept of an external object or "applet", i.e. a program that can in some way be executed in the browser or web client during document manipulation. This is one of the primary format-independent ways used to enliven parts of the document. Other ways are to change the document object model via an embedded programming language (e.g. JavaScript). As this method (dynamic HTML) is format-dependent[5], it seems difficult to support in a content markup format like OMDoc.

The challenge here is to come up with a format-independent representation of the applet functionality, so that the OMDoc representation can be transformed into the specific form needed by the respective presentation format. Most user agents for these presentation formats have built-in mechanisms for processing common data types such as text and various image types. In some instances the user agent may pass the processing to an external application ("plug-ins"). These need information about the location of the object data, the MIME type associated with the object data, and additional values required for the appropriate processing of the object data by the object handler at run-time.

Element	Attributes		D	Content
	Req.	Optional	C	
omlet	data,	xml:id, action, show, actuate, class, style	+	(《*CMP content*》 | param)*,private*,code*
param	name	value, valuetype	-	EMPTY

Fig. 20.2. The OMDoc Elements for External Objects

In OMDoc, we use the `omlet` element for applets. It generalizes the HTML applet concept in two ways: The computational engine is not restricted to plug-ins of the browser (we do not know what the result format and presentation engine will be) and the program code can be included in the OMDoc document, making document-centered computation easier to manage. `omlet`

Like the `xhtml:object` tag, the `omlet` element can be used to wrap any text. In the OMDoc context, this means that the children of the `omlet` element can be any elements or text that can occur in the `CMP` element together with `param` elements to specify the arguments. The main presentation intuition is that the applet reserves a rectangular space of a given pre-defined size (specified in the CSS markup in the `style` attribute; see Listing 20.5) in the result document presentation, and hands off the presentation and interaction with the document in this space to the applet process. The data for the external object is referenced in two possible ways. Either via the `data` attribute, which contains a URI reference that points to an OMDoc `code` or `private` element that is accessible (e.g. in the same OMDoc) or by embedding the

[5] In particular, the JavaScript references the HTML DOM, which in our model is created by a presentation engine on the fly.

respective `code` or `private` elements as children at the end of the `omlet` element. This indirection allows us to reuse the machinery for storing code in OMDocs. For a simple example see Listing 20.5.

The behavior of the external object is specified in the attributes `action`, `show` and `actuate` attributes[6].

The `action` specified the intended action to be performed with the data. For most objects, this is clear from the MIME type. Images are to be displayed, audio formats will be played, and application-specific formats are passed on to the appropriate plug-in. However, for the latter (and in particular for program code), we might actually be interested to display the data in its raw (or suitably presented) form. The `action` addresses this need, it has the possible values `execute` (pass the data to the appropriate plug-in or execute the program code), `display` (display it to the user in audio- or visual form), and `other` (the action is left unspecified).

The `show` attribute is used to communicate the desired presentation of the ending resource on traversal from the starting resource. It has one of the values `new` (display the object in a new document), `replace` (replace the current document with the presentation of the external object), `embed` (replace the `omlet` element with the presentation of the external object in the current document), and `other` (the presentation is left unspecified).

The `actuate` attribute is used to communicate the desired timing of the action specified in the `action` attribute. Recall that OMDoc documents as content representations are not intended for direct viewing by the user, but appropriate presentation formats are derived from it by a "presentation process" (which may or may not be incorporated into the user agent). Therefore the `actuate` attribute can take the values `onPresent` (when the presentation document is generated), `onLoad` (when the user loads the presentation document), `onRequest` (when the user requests it, e.g. by clicking in the presentation document), and `other` (the timing is left unspecified).

The simplest form of an `omlet` is just the embedding of an external object like an image as in Listing 20.4, where the `data` attribute points to the `private` element in Listing 20.2. For presentation, e.g. as XHTML in a modern browser, this would be transformed into an `xhtml:object` element [The02], whose specific attributes are determined by the information in the `omlet` element here and those `data` children of the `private` element specified in the `data` attribute of the `omlet` that are chosen for presentation in XHTML. If the action specified in the `action` attribute is impossible (e.g. if the contents of the `data` target cannot be presented), then the content of the `omlet` element is processed as a fallback.

Listing 20.4. An `omlet` for an Image

```
<omlet data="#legacy" show="embed">A Fragment of Bourbaki's Algebra</omlet>
```

[6] These latter two attributes are modeled after the XLink [DeR+01] attributes `show` and `actuate`.

In Listing 20.5 we present an example of a conventional Java applet in a mathematical text: the `data` attribute points to a `code` element, which will be executed (if the value of the `action` attribute were `display`, the code would be displayed).

Listing 20.5. An `omlet` that Calls the Java Applet from Listing 20.3.

```
<omtext xml:id=”monp_1”>
  <CMP><xhtml:p>
    <p>Let practice integration!</p>
    <p><omlet data=”#callMint” action=”execute” style=”width:320;height:200”>
        No plug−in found for callMint!
      </omlet></p>
  </xhtml:p></CMP>
</omtext>
```

In this example, the Java applet did not need any parameters (compare the documentation in the `input` element in Listing 20.3).

In the applet in Listing 20.6 we assume a code fragment or plug-in (in a `code` element whose `xml:id` attribute has the value `sendtoTP`, which we have not shown) that processes a set of named arguments (parameter passing with keywords) and calls the theorem prover, e.g. via a web-service as described in Chapter 9.

Listing 20.6. An `omlet` for Connecting to a Theorem Prover

```
<CMP><xhtml:p>Let us prove it interactively:
  <omlet data=”#sendtoTP” action=”display”>
    <param name=”timeout” value=”30” valuetype=”data”/>
    <param name=”performative” value=”prove”/>
    <param name=”problem” value=”#ALGX0” valuetype=”object”/>
    <param name=”description” value=”http://example.org/prob17.html” valuetype=”ref”/>
    <param name=”instance”>
      <OMOBJ>
        <OMA><OMS name=”root” cd=”arith1”/>
          <OMI>3</OMI><OMI>3</OMI>
        </OMA>
      </OMOBJ>
    </param>
    Sorry, no theorem prover available!
  </omlet></xhtml:p>
</CMP>
```

For parameter passing, we use the `param` elements which specify a set of values that may be required to process the object data by a plug-in at run-time. Any number of `param` elements may appear in the content of an `omlet` element. Their order does not carry any meaning. The `param` element carries the attributes `param`

`name` This required attribute defines the name of a run-time parameter, assumed to be known by the plug-in. Any two `param` children of an `omlet` element must have different `name` values.

`value` This attribute specifies the value of a run-time parameter passed to the plug-in for the key `name`. Property values have no meaning to OMDoc; their meaning is determined by the plug-in in question.

`valuetype` This attribute specifies the type of the `value` attribute. The value `data` (the default) means that the value of the `value` will be passed to the plug-in as a string. The value `ref` specifies that the value of the `value` attribute is to be interpreted as a URI reference that designates a resource where run-time values are stored. Finally, the value `object` specifies that the `value` value points to a `private` or `code` element that contains a multi-format collection of `data` elements that carry the data.

If the `param` element does not have a `value` attribute, then it may contain a list of mathematical objects encoded as `om:OMOBJ`, `m:mathml`, or `legacy` elements.

21

Exercises (Module QUIZ)

Exercises and study problems are vital parts of mathematical documents like textbooks or exams, in particular, mathematical exercises contain mathematical vernacular and pose the same requirements on context like mathematical statements. Therefore markup for exercises has to be tightly integrated into the document format, so OMDoc provides a module for them.

Note that the functionality provided in this module is very limited, and largely serves as a place-holder for more pedagogically informed developments in the future (see Section ?? and [Gog+03] for an example in the OMDoc framework).

Element	Attributes		D	Content
	Req.	Optional	C	
`exercise`		`xml:id, class, style`	+	`CMP*,FMP*,hint?,(solution*|mc*)`
`hint`		`xml:id, class, style`	+	`CMP*, FMP*`
`solution`		`xml:id, for, class, style`	+	⟪*top-level element*⟫
`mc`		`xml:id, for, class, style`	–	`choice, hint?, answer`
`choice`		`xml:id, class, style`	+	`CMP*, FMP*`
`answer`	`verdict`	`xml:id, class, style`	+	`CMP*, FMP*`

Fig. 21.1. The OMDoc Auxiliary Elements for Exercises

The QUIZ module provides the top-level elements `exercise`, `hint`, and `solution`. The first one is used for exercises and assessments. The question statement is represented in the multilingual `CMP` group followed by a multilogic `FMP` group. This information can be augmented by hints (using the `hint` element) and a solution/assessment block (using the `solution` and `mc` elements). `exercise`

The `hint` and `solution` elements can occur as children of `exercise`; or outside, referencing it in their optional `for` attribute. This allows a flexible positioning of the hints and solutions, e.g. in separate documents that can be distributed separately from the `exercise` elements. The `hint` element contains a `CMP`/`FMP` group for the hint text. The `solution` element can contain `hint` `solution` any number of OMDoc top-level elements to explain and justify the solution. This is the case, where the question contains an assertion whose proof is not displayed and left to the reader. Here, the `solution` contains a proof.

Listing 21.1. An Exercise from the TEXBook

```
<exercise xml:id="TeXBook−18−22">
  <CMP><xhtml:p>
    <p>Sometimes the condition that defines a set is given as a fairly long
      English description; for example consider '{p|p and p+2 are prime}'. An
      hbox would do the job:</p>

    <p style="display:block;font−family:fixed">
      $\{\,p\mid\hbox{$p$ and $p+2$ are prime}\,\}$
    </p>

    <p>but a long formula like this is troublesome in a paragraph, since an hbox cannot
      be broken between lines, and since the glue inside the
      <xhtml:span style="font−family:fixed">\hbox</xhtml:span> does not vary with the inter−word
      glue in the line that contains it. Explain how the given formula could be
      typeset with line breaks.</p>
  </xhtml:p></CMP> <hint>
    <CMP><xhtml:p>Go back and forth between math mode and horizontal mode.</xhtml:p></CMP>
  </hint>
  <solution>
    <CMP><xhtml:p>
      <xhtml:span style="font−family:fixed">
       $\{\,p\mid p$~and $p+2$ are prime$\,\}$
      </xhtml:span>,
      assuming that <xhtml:span style="font−family:fixed">\mathsurround</xhtml:span> is
      zero. The more difficult alternative '<xhtml:span style="font−family:fixed">
      $\{\,p\mid p\\ {\rm and}\ p+2\rm\ are\ prime\,\}$</xhtml:span>'
       is not a solution, because line breaks do not occur at
      <xhtml:span style="font−family:fixed">\␣</xhtml:span> (or at glue of any
      kin) within math formulas. Of course it may be best to display a formula like
      this, instead of breaking it between lines.
    </xhtml:p></CMP>
  </solution>
</exercise>
```

Multiple-choice exercises (see Listing 21.2) are represented by a group of `mc` elements inside an `exercise` element. An `mc` element represents a single choice in a multiple choice element. It contains the elements below (in this order).

`mc`

`choice` for the description of the choice (the text the user gets to see and is asked to make a decision on). The `choice` element carries the `xml:id`, `style`, and `class` attributes and contains a `CMP`/`FMP` group for the text.

`choice`

`hint` (optional) for a hint to the user, see above for a description.

`answer` for the feedback to the user. This can be the correct answer, or some other feedback (e.g. another hint, without revealing the correct answer). The `verdict` attribute specifies the truth of the answer, it can have the values `true` or `false`. This element is required, inside a `mc`, since the `verdict` is needed. It can be empty if no feedback is available. Furthermore, the `answer` element carries the `xml:id`, `style`, and `class` attributes and contains a `CMP`/`FMP` group for the text.

`answer`

Listing 21.2. A Multiple-Choice Exercise in OMDoc

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

22

Document Models for OMDoc

In almost all XML applications, there is a tension between the document view and the object view of data; after all, XML is a document-oriented interoperability framework for exchanging data objects. The question, which view is the correct one for XML in general is hotly debated among XML theorists. In OMDoc, actually both views make sense in various ways. Mathematical documents are the objects we try to formalize, they contain knowledge about mathematical objects that are encoded as formulae, and we arrive at content markup for mathematical documents by treating knowledge fragments (statements and theories) as objects in their own right that can be inspected and reasoned about.

In Chapters 13 to 21, we have defined what OMDoc documents look like and motivated this by the mathematical objects they encode. But we have not really defined the properties of these documents as objects themselves (we will speak of the OMDoc **document object model** (OMDOM)). To get a feeling for the issues involved, let us take stock of what we mean by the object view of data. In mathematics, when we define a class of mathematical objects (e.g. vector spaces), we have to say which objects belong to this class, and when they are to be considered equal (e.g. vector spaces are equal, iff they are isomorphic). When defining the intended behavior of operations, we need to care only about objects of this class, and we can only make use of properties that are invariant under object equality. In particular, we cannot use properties of a particular realization of a vector space that are not preserved under isomorphism. For document models, we do the same, only that the objects are documents.

22.1 XML Document Models

XML supports the task of defining a particular class of documents (e.g. the class of OMDoc documents) with formal grammars such as the document type definition (DTD) or an XML schema, that can be used for mechanical document validation. Surprisingly, XML leaves the task of specifying document equality to be clarified in the (informal) specifications, such as this OMDoc specification. As a consequence, current practice for XML applications is quite varied. For instance, the OpenMath standard (see [Bus+04] and Section 13.1) gives a mathematical object model for OpenMath objects that is specified independently of the XML encoding. Other XML applications like e.g. presentation MathML [Aus+03a] or XHTML [The02] specify models in form of the intended screen presentation, while still others like the XSLT [Cla99b] give the operational semantics.

For a formal definition let $\mathcal{K}$ be a set of documents. We take a **document model** to be a partial equivalence relation[1] $\mathcal{X}$ on documents, such that $\{d|d\mathcal{X}d\} = \mathcal{K}$. In particular, a relation $\mathcal{X}$ is an equivalence relation on $\mathcal{K}$. For a given document model $\mathcal{X}$, let us say that two documents d and d' are **$\mathcal{X}$-equal**, iff $d\mathcal{X}d'$. We call a property p **$\mathcal{X}$-invariant**, iff for all $d\mathcal{X}d'$, p holds on d whenever p holds on d'.

A possible source of confusion is that documents can admit more than one document model (see [KK06a] for an exploration of possible document models for mathematics). Concretely, OMDoc documents admit the OMDoc document model that we will specify in section Section 22.2 and also the following four XML document models that can be restricted to OMDoc documents (as a relation).[2]

The binary document model interprets files as sequences of bytes. Two documents are equal, iff they are equal as byte sequence. This is the most concrete and fine-grained (and thus weakest) document model imaginable.

The lexical document model interprets binary files as sequences of Unicode characters [Inc03] using an encoding table. Two files may be considered equal by this document model even though they differ as binary files, if they have different encodings that map the byte sequences to the same sequence of Unicode characters.

The XML syntax document model interprets Unicode Files as sequences consisting of an XML declaration, a DOCTYPE declaration, tags, entity references, character references, CDATA sections, PCDATA comments, and processing instructions. At this level, for instance, whitespace characters between XML tags are irrelevant, and XML documents may be considered the same, if they are different as Unicode sequences.

The XML structure document model interprets documents as XML trees of elements, attributes, text nodes, processing instructions, and sometimes comments. In this document model the order of attribute declarations in

[1] A partial equivalence relation is a symmetric transitive relation. We will use $[d]_{\mathcal{X}}$ for the **equivalence class** of d, i.e. $[d]_{\mathcal{X}} := \{e|d\mathcal{X}e\}$

[2] Here we follow Eliotte Rusty Harold's classification of layers of XML processing in [Har03], where he distinguishes the binary, lexical, sequence, structure, and semantic layer, the latter being the document model of the XML application

XML elements is immaterial, double and single quotes can be used interchangeably for strings, and XML comments (<!--...-->) are ignored.

Each of these document models, is suitable for different applications, for instance the lexical document model is the appropriate one for Unicode-aware editors that interpret the encoding string in the XML declaration and present the appropriate glyphs to the user, while the binary document model would be appropriate for a simple ASCII editor. Since the last three document models are refinements of the XML document model, we will recap this in the next section and define the OMDoc document model in Section 22.2.

To get a feeling for the issues involved, let us compare the OMDoc elements in Listings 22.1 to 22.3 below. For instance, the serialization in Listing 22.2 is XML-equal to the one in Listing 22.1, but not to the one in Listing 22.3.

Listing 22.1. An OMDoc Definition

```
<definition xml:id=”comm−def” for=”comm”>
  <CMP xml:lang=”en”><xhtml:p>
    An operation <OMOBJ id=”op”><OMV name=”op”/></OMOBJ>
     is called commutative, iff
    <OMOBJ id=”comm1”>
      <OMA><OMS cd=”relation1” name=”eq”/>
        <OMA><OMV name=”op”/><OMV name=”X”/><OMV name=”Y”/></OMA>
        <OMA><OMV name=”op”/><OMV name=”Y”/><OMV name=”X”/></OMA>
      </OMA>
    </OMOBJ> for all <OMOBJ id=”x”><OMV name=”X”/></OMOBJ>
    and <OMOBJ id=”y”><OMV name=”Y”/></OMOBJ>.</xhtml:p>
  </CMP>
  <CMP xml:lang=”de”>
    <xhtml:p>Eine Operation <OMOBJ><OMR href=”#op”/></OMOBJ> heißt kommutativ, falls
    <OMOBJ><OMR href=”#comm1”/></OMOBJ> für alle
    <OMOBJ><OMR href=”#x”/></OMOBJ> und
    <OMOBJ><OMR href=”#y”/></OMOBJ>.</xhtml:p>
  </CMP>
</definition>
```

Listing 22.2. An XML-equal serialization for Listing 22.1

```
<definition for=”comm” xml:id=”comm−def” >
  ...
  <CMP xml:lang='de'> <!−− Note the unabbreviated empty element −−>
   <xhtml:p>Eine Operation <OMOBJ><OMR href=”#op”/></OMOBJ> heißt
   kommutativ, falls <OMOBJ><OMR href='comm1'/></OMOBJ> für alle
   <OMOBJ><OMR href=”#x”/></OMOBJ> und
   <OMOBJ><OMR href='y'/></OMOBJ>.</xhtml:p>
  </CMP>
</definition>
```

22.2 The OMDoc Document Model

The OMDoc document model extends the XML structure document model in various ways. We will specify the equality relation in the table below, and discuss a few general issues here.

The OMDoc document model is guided by the notion of content markup for mathematical documents. Thus, two document fragments will only be considered equal, if they have the same abstract structure. For instance, the order of `CMP` children of an `omtext` element is irrelevant, since they form a multilingual group which form the base for multilingual text assembly. Other facets of the OMDoc document model are motivated by presentation-independence, for instance the distribution of whitespace is irrelevant even in text nodes, to allow formatting and reflow in the source code, which is not considered to change the information content of a text.

Listing 22.3. An OMDoc-Equal Representation for Listings 22.1 and 22.2

```
<definition xml:id=”comm−def” for=”comm”>
  <CMP xml:lang=”de”><xhtml:p>Eine Operation <OMOBJ><OMR href=”#op”/></OMOBJ>
    heißt kommutativ, falls
    <OMOBJ id=”comm1”>
      <OMA><OMS cd=”relation1” name=”eq”/>
        <OMA><OMV name=”op”/><OMV name=”X”/><OMV name=”Y”/></OMA>
        <OMA><OMV name=”op”/><OMV name=”Y”/><OMV name=”X”/></OMA>
      </OMA>
    </OMOBJ> für alle <OMOBJ><OMR href=”#x”/></OMOBJ> und
    <OMOBJ><OMR href=”#y”/></OMOBJ>.</xhtml:p>
  </CMP>
  <CMP xml:lang=”en”>
    <xhtml:p>An operation <OMOBJ id=”op”><OMV name=”op”/></OMOBJ>
    is called commutative, iff <OMOBJ><OMR href=”#comm1”/></OMOBJ>
    for all <OMOBJ id=”x”><OMV name=”X”/></OMOBJ> and
    <OMOBJ id=”y”><OMV name=”Y”/></OMOBJ>.</xhtml:p>
  </CMP>
</definition>
```

Compared to other document models, this is a rather weak (but general) notion of equality. Note in particular, that the OMDoc document model does *not* use mathematical equality here, which would make the formula $X + Y = Y + X$ (the `om:OMOBJ` with `xml:id="comm1"` in Listing 22.3 instantiated with addition for `op`) mathematically equal to the trivial condition $X+Y = X+Y$, obtained by exchanging the right hand side $Y + X$ of the equality by $X + Y$, which is mathematically equal (but not OMDoc-equal).

Let us now specify (part of) the equality relation by the rules in the table in Figure 22.1. We have discussed a machine-readable form of these equality constraints in the XML schema for OMDoc in [KA03].

The last rule in Figure 22.1 is probably the most interesting, as we have seen in Chapter 11, OMDoc documents have both formal and informal aspects, they can contain *narrative* as well as *narrative-structured* information. The latter kind of document contains a formalization of a mathematical theory, as a reference for automated theorem proving systems. There, logical dependencies play a much greater role than the order of serialization in mathematical objects. We call such documents **content OMDoc** and specify the value `Dataset` in the `dc:type` element of the OMDoc metadata for such documents. On the other extreme we have human-oriented presentations of mathematical knowledge, e.g. for educational purposes, where didactic considerations determine the order of presentation. We call such documents **narrative-**

#	Rule	comment	elements
1	unordered	The order of children of this element is irrelevant (as far as permitted by the content model). For instance only the order of `obligation` elements in the `axiom-inclusion` element is arbitrary, since the others must precede them in the content model.	`adt` `axiom-inclusion` `metadata` `symbol` `code` `private` `presentation` `omstyle`
2	multi-group	The order between siblings elements does not matter, as long as the values of the key attributes differ.	`CMP` `FMP` `requation` `dc:description` `sortdef` `data` `dc:title` `solution`
3	DAG encoding	Directed acyclic graphs built up using `om:OMR` elements are equal, iff their tree expansions are equal.	`om:OMR` OMDoc reference
4	Dataset	If the content of the `dc:type` element is `Dataset`, then the order of the siblings of the parent `metadata` element is irrelevant.	`dc:type`

Fig. 22.1. The OMDoc Document Model

structured and specify this by the value `Text` (also see the discussion in Section 12.2)

22.3 OMDoc Sub-Languages

In the last chapters we have described the OMDoc modules. Together, they make up the OMDoc document format, a very rich format for marking up the content of a wide variety of mathematical documents. (see Part II for some worked examples). Of course not all documents need the full breadth of OMDoc functionality, and on the other hand, not all OMDoc applications (see Part **??** for examples) support the whole language.

One of the advantages of a modular language design is that it becomes easy to address this situation by specifying sub-languages that only include part of the functionality. We will discuss plausible OMDoc sub-languages and their applications that can be obtained by dropping optional modules from OMDoc. Figure 22.2 visualizes the sub-languages we will present in this chapter. The full language OMDoc is at the top, at the bottom is a minimal sub-language OMDoc Basic, which only contains the required modules (mathematical documents without them do not really make sense). The arrows signify language inclusion and are marked with the modules acquired in the extension.

The sub-language identifiers can be used as values of the `modules` attribute on the `omgroup` and `omdoc` elements. Used there, they abbreviate the list of modules these sub-languages contain.

22.3.1 Basic OMDoc

Basic OMDoc is sufficient for very simple mathematical documents that do not introduce new symbols or concepts, or for early (and non-specific) stages

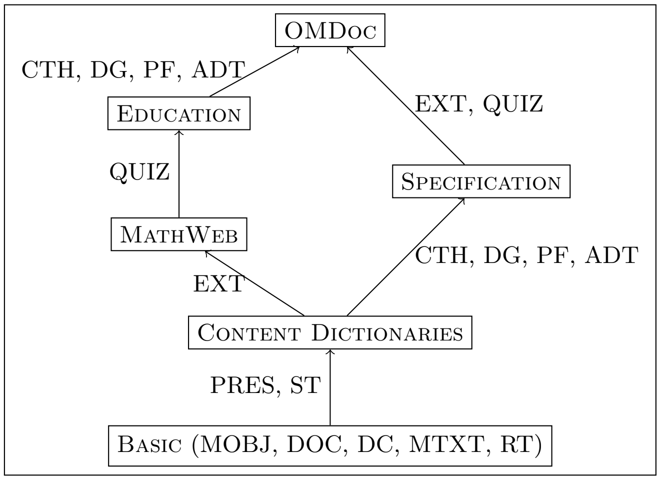


Fig. 22.2. OMDoc Sub-Languages and Modules

in the migration process from legacy representations of mathematical material (see Section 4.2). This OMDoc sub-language consists of five modules: we need module MOBJ for mathematical objects and formulae, which are present in almost all mathematical documents. Module DOC provides the document infrastructure, and in particular, the root element `omdoc`. We need DC for titles, descriptions, and administrative metadata, and module MTXT so we can state properties about the mathematical objects in `omtext` element. Finally, module RT allows to structured text below the `omtext` level. This module is not strictly needed for basic OMDoc, but we have included it for convenience.

22.3.2 OMDoc Content Dictionaries

Content Dictionaries are used to define the meaning of symbols in the OpenMath standard [Bus+04], they are the mathematical documents referred to in the `cd` attribute of the `om:OMS` element. To express content dictionaries in OMDoc, we need to add the module ST to Basic OMDoc. It provides the possibility to specify the meaning of basic mathematical objects (symbols) by axioms and definitions together with the infrastructure for inheritance, and grouping, and allows to reference the symbols defined via their home theory (see the discussion in Section 15.6).

With this extension alone, OMDoc content dictionaries add support for multilingual text, simple inheritance for theories, and document structure to the functionality of OpenMath content dictionaries. Furthermore, OMDoc content dictionaries allow the conceptual separation of mathematical properties into constitutive ones and logically redundant ones. The latter of these are not strictly essential for content dictionaries, but enhance maintainability and readability, they are included in OpenMath content dictionaries for documentation and explanation.

The sub-language for OMDoc content dictionaries also allows the specification of notations for the introduced symbols (by module PRES). So the resulting documents can be used for referencing (as in OpenMath) and as a resource for deriving presentation information for the symbols defined here. To get a feeling for this sub-language, see the example in the OMDoc variant of the OpenMath content dictionary `arith1` in Chapter 5, which shows that the OpenMath content dictionary format is (isomorphic to) a subset of the OMDoc format. In fact, the OpenMath2 standard only presents the content dictionary format used here as one of many encodings and specifies abstract conditions on content dictionaries that the OMDoc encoding below also meets. Thus OMDoc is a valid content dictionary encoding.

22.3.3 Specification OMDoc

OMDoc content dictionaries are still a relatively lightweight format for the specification of meaning of mathematical symbols and objects. Large scale formal specification efforts, e.g. for program verification need more structure to be practical. Specification languages like Casl (Common Algebraic Specification Language [Mos04]) offer the necessary infrastructure, but have a syntax that is not integrated with web standards.

The Specification OMDoc sub-language adds the modules ADT and CTH to the language of OMDoc content dictionaries. The resulting language is equivalent to the Casl standard, see [Aut+00; Hut00; MAH06] for the necessary theory.

The structured definition schemata from module ADT allow to specify abstract data types, sets of objects that are inductively defined from constructor symbols. The development graph structure built on the theory morphisms from module CTH allow to make inclusion assertions about theories that structure fragments of mathematical developments and support a management of change.

22.3.4 MathWeb OMDoc

OMDoc can be used as a content-oriented basis for web publishing of mathematics. Documents for the web often contain images, applets, code fragments, and other data, together with mathematical statements and theories.

The OMDoc sub-language MathWeb OMDoc extends the language for OMDoc content dictionaries by the module EXT, which adds infrastructure for images, applets, code fragments, and other data.

22.3.5 Educational OMDoc

OMDoc is currently used as a content-oriented basis for various systems for mathematics education (see e.g. Chapter 8 for an example and discussion). The OMDoc sub-language Educational OMDoc extends MathWeb OMDoc by the module QUIZ, which adds infrastructure for exercises and assessments.

22.3.6 Reusing OMDoc modules in other formats

Another application of the modular language design is to share modules with other XML applications. For instance, formats like DocBook [WM08] or XHTML [The02] could be extended with the OMDoc statement level. Including modules MOBJ, DC, and (parts of) MTXT, but not RT and DOC would result in content formats that mix the document-level structure of these formats. Another example is the combination of XML-RPC envelopes and OMDoc documents used for interoperability in Chapter 9.

Part IV

Appendix

In this appendix, we document the changes of the OMDoc format over the versions, provide quick reference tables, and discuss the validation helps

A

Changes to the specification

After about 18 Months of development, Version 1.0 of the OMDoc format was released on November 1^{st} 2000 to give users a stable interface to base their documents and systems on. It was adopted by various projects in automated deduction, algebraic specification, and computer-supported education. The experience from these projects uncovered a multitude of small deficiencies and extension possibilities of the format, that have been subsequently discussed in the OMDoc community.

OMDoc 1.1 was released on December 29^{th} 2001 as an attempt to roll the uncontroversial and non-disruptive part of the extensions and corrections into a consistent language format. The changes to version 1.0 were largely conservative, adding optional attributes or child elements. Nevertheless, some non-conservative changes were introduced, but only to less used parts of the format or in order to remedy design flaws and inconsistencies of version 1.0.

OMDoc 1.3 is the mature version in the OMDoc 1 series of specifications. It contains almost no large-scale changes to the document format, except that Content-MathML is now allowed as a representation for mathematical objects. But many of the representational features have been fine-tuned and brought up to date with the maturing XML technology (e.g. `ID` attributes now follow the XML ID specification [MVW05], and the Dublin Core elements follow the official syntax [DUB03a]). The main development is that the OMDoc specification, the DTD, and schema are split into a system of interdependent modules that support independent development of certain language aspects and simpler specification and deployment of sub-languages. Version 1.3 of OMDoc freezes the development so that version 2 can be started off on the modules.

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

All changes will be relative to the previous version, starting out with OMDoc 1.0.

A.1 Changes from 1.2 to 1.3

The main change from OMDoc 1.2 to OMDoc 1.3 is the use of the new notation framework described in Chapter 19. It completely replaces the presentation archicture of OMDoc 1.2.

The other large change is to use the new namespace `http://omdoc.org/ns` that will also be used in OMDoc 1.2

element	state	comments	cf.
`dd`	cha	description items now allow block content as in XHTML	Section ??
`bibliography`	new	generates the references	Section 11.2
`citation`	new	marks up a citation	Section ??
`index`	new	generates the index	Section 11.2
`li`	cha	list items now allow block content as in XHTML	Section ??
`metadata`	cha	the optional attribute `inherits` dropped, it was never sufficiently defined.	Section 11.3
`presentation`	del	replaced by the `notation` element.	Chapter 19
`style`	del	obsolete, since it was never used.	
`tableofcontents`	new	generates the tableofcontents	Section 11.2
`tgroup`	del	replaced by the `omgroup` element, it turns out that with RelaxNG we can do the necessary validation of theory content after all.	Chapter 15
`uses`	new	opens a CD catalog.	Chapter ??

A.2 Changes from 1.1 to 1.2

Most of the changes in version 1.2 are motivated by modularization. The goal was to modularize the specification so that it can be used as a DTD module, and that restricted sub-languages of OMDoc can be identified.

Perhaps the most disruptive change is in the presentation/style apparatus: In version 1.1, OMDoc used the `style` attribute for all elements that have an `id` attribute to specify generic style classes for the OMDoc elements. This was based on a misunderstanding of the XML cascading style sheet (CSS) mechanism [Bos+98], which uses the `class` attribute to specify this information and uses the `style` attribute to specify CSS directives that override the class information. This error in Version 1.1 of OMDoc so severely limits the usefulness for styling that we rename the Version 1.1 of OMDoc `style` attribute to `class`, even though it breaks 1.1-compatible implementations. Concretely, the Version 1.2 of OMDoc `class` attribute takes the role of the Version 1.1 of OMDoc `style`. and the Version 1.2 of OMDoc `style` takes CSS directives.

Furthermore, all `xml:id` on non-constitutive (see Section 15.1) elements in OMDoc were made optional.

Version 1.1 of OMDoc files can be upgraded to version 1.2 with the XSLT style sheet `https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/xsl/omdoc1.1adapt1.2.xsl`.

element	state	comments	cf.
`alternative`	aug	This element can now have `theory`, `generated-from`, and `generated-via` attributes.	162
`argument`	cha	The `sort` has been replaced by a `type` child, so that higher-order sorts can be specified.	172
`assertion`	aug	the `assertion` element now has an optional `for` attribute. Furthermore, an optional attribute `generated-via` has been added to allow generation via a theory morphism. Finally, two new attributes `status` and `just-by` have been added to mark up the deductive status of the assertion.	158
`assumption`	cha	This element can now have an attribute `inductive` for inductive assumptions. The natural langauge description in the optional `CMP` element is no longer allowed, use a `phrase` element in a `CMP` that is a sibling to the `FMP` instead.	162
`adt`	aug	the `adt` loses the `CMP` and `commonname` children, use the Dublin Core metadata elements `dc:description` and `dc:subject` instead. The `type` attribute is now on the `sortdef` element. Furthermore, an optionala attribute `generated-via` has been added to allow generation via a theory morphism. Finally, an attribute `parameters` has been added to allow for parametric ADTs.	171

`answer`	cha	the `answer` element does not allow `symbol` children any more, if these are needed, the exercise should have its own theory.	210
`attribute`	aug	the `attribute` element now has a optional `ns` attribute for the namespace URI of the generated attribute node and an attribute `select` for an XPath expression that specifies the value of the generated attribute.	??
`axiom`	aug	the `axiom` element now has an optional `for` attribute which can point to a list of symbols. Furthermore, an optional attribute `generated-via` has been added to allow generation via a theory morphism and an attribute `type` is now also allowed.	154
`axiom-inclusion`	lib	the `axiom-inclusion` element can now contain multiple `path-just` children to record multiple justifications. Furthermore, it can now have `theory`, `generated-from`, and `generated-via` attributes. New optional attributes `conservativity` and `conservativity-just` for stating and justifying conservativity.	196
`catalogue`	dep	the catalogue mechanism has been eliminated.	
`choice`	cha	the `choice` element does not allow `symbol` children any more, if these are needed, the exercise should have its own theory	210
`code`	cha	Attributes `classid` and `codebase` are deprecated. The attributes `pto` and `pto-version` have moved to the `data` element. The attribute `type` has been removed and optional attributes `theory`, `generated-from`, and `generated-via` have been added.	202
`commonname`	dep	This element is deprecated in favor of a `metadata/dc:subject` element.	
`conclusion`	cha	The natural langauge description in the optional `CMP` element is no longer allowed, use a `phrase` element in a `CMP` that is a sibling to the `FMP` instead.	148
`constructor`	cha	The `role` attribute is now fixed to `object`. The `commonname` child has been replaced by an initial `metadata` element.	172

`data`	aug	new optional attributes `original` to specify whether the external resource referenced by the `href` attribute (value `external`) or the `data` content is the original (value `local`). The `data` element has acquired attributes `pto` and `pto-version` from the `code` and `private` elements.	203
`dc:*`	aug	All Dublin Core tags have been lowercased to synchronize with the tag syntax recommended by the Dublin Core Initiative. The tags were capitalized in OMDoc1.1. Furthermore, `dc:contributor`, `dc:creator`, `dc:publisher` have received an optional `xml:id` attribute, so that they can be cross-referenced by the new `who` of the `dc:date` element.	113
`decomposition`	aug	The `for` attribute is now optional, it need not be given, if the element is a child of a `theory-inclusion` element. Furthermore, it can now have a `theory`, `generated-from`, and `generated-via` attributes.	??
`dc:description`	aug	The `dc:description` can now have the optional `xml:id`, and CSS attributes	114
`definition`	aug	The `definition` element can now have the type `pattern` for pattern-defined functions. This is a degenerate case of the type `inductive`. Furthermore, an optional attribute `generated-via` has been added to allow generation via a theory morphism.	155
`effect`	aug	allows an optional `xml:id` attribute	204
`example`	aug	The `example` element now has the optional `theory` attribute that specifies the home theory. Furthermore, it can now have attributes `theory`, `generated-from`, and `generated-via`.	163
`exercise`	cha	the `exercise` element does not allow `symbol` children any more, if these are needed, the exercise should have its own theory. Furthermore, it can now have a `theory`, `generated-from`, and `generated-via` attributes.	209
`extradata`	cha	The content of the old `extradata` element can now be directly in the `metadata/dc:subject` element.	

`element`	aug	The `element` element now allows the `map` and `separator` elements in the body. Furthermore, it carries the optional attributes `crid` for parallel markup, `cr` for cross-references, and `ns` for specifying the namespace.	**??**
`hint`	aug	the `hint` element can now appear on top-level and has a `for` attribute. It does not allow `symbol` children any more, if these are needed, the exercise should have its own theory. Furthermore, the `exercise` can now have a `theory`, `generated-from`, and `generated-via` attributes.	209
`hypothesis`	cha	the `discharged-in` attribute has been eliminated. Scoping is now specified in terms of the enclosing `proof` element. Furthermore, the `symbol` child is no longer allowed inside the element. A sibling `symbol` should be used.	179
`inclusion`	aug	allows optional attributes `xml:id`, `conservativity`, and `conservativity-just` for stating and justifying conservativity.	195
`imports`	lib	the `xml:id` is now optional. New optional attributes `conservativity` and `conservativity-just` for stating and justifying conservativity.	166
`input`	aug	allows an optional `xml:id` attribute	204
`legacy`	new	An element for encapsulating legacy mathematics, can be used wherever `m:math` and `om:OMOBJ` are allowed.	134
`loc`	dep	The catalogue mechanism has been eliminated.	
`m:math`	new	Content-MathML is now allowed wherever OpenMath objects were allowed before.	129
`map`	new	this element allows to map its style directives over a list of e.g. arguments	**??**
`mc`	aug	the `mc` element can now have a `for` attribute. It does not allow `symbol` children any more, if these are needed, the dominating `exercise` element should have its own theory. Furthermore, the `mc` element can now have a `theory`, `generated-from`, and `generated-via` attributes.	210
`measure`	aug	allows an optional `xml:id` attribute	157

metacomment	dep	This element is superseded by the omtext element.	145
morphism	aug	The morphism element now carries the optional attributes consistency, exhaustivity, hiding, and type. Furthermore the content model allows optional elements measure and ordering after the requation children to specify termination information like in definition.	100
obligation	aug	allows an optional xml:id attribute	194
omdoc	aug	This element can now have a theory, generated-from, and generated-via attributes.	98
omgroup	cha	The values dataset and labeled-dataset are deprecated in Version 1.2 of OMDoc, since we provide tables in module RT; see Section ?? for details. Furthermore, the element can now have the attributes, modules, theory, generated-from, and generated-via.	166
omlet	cha	omlet can no longer occur at top-level (it just does not make sense). The data model for this element has been totally reworked, inspired by the xhtml:object element.	205
omstyle	aug	This element can now have generated-from, and generated-via attributes. New attribute xref that allows to inherit the information from another omstyle element.	??
om:*	aug	with OpenMath2, the OpenMath elements carry an optional id attribute for structure sharing via the om:OMR element. Furthermore, in OMDoc, they carry cref attributes for parallel markup with cross-references.	122
om:OMFOREIGN	new	The om:OMFOREIGN element can be used to encapsulate arbitrary XML data in OpenMath attributions.	125
om:OMR	new	In the OpenMath2 standard, this element is the main vehicle of the structure sharing representation.	126

`omtext`	aug	the `type` attribute can now also have the values `axiom`, `definition`, `theorem`, `proposition`, `lemma`, `corollary`, `postulate`, `conjecture`, `false-conjecture`, `obligation`, `assumption`, and `formula`. Furthermore, `omtext` can now have `theory`, `generated-from`, and `generated-via` and `verbalizes` attributes.	145
`ordering`	aug	Now allows the optional `xml:id` and `terminating` attributes. The latter points to a termination assertion.	157
`output`	aug	allows an optional `xml:id` attribute	204
`pattern`	aug	this element is no longer used, the pattern of a recursive equation is determined by the position as the first child.	
`path-just`	aug	The element can now appear as a top-level element, if it does, the attribute `for` must point to the `axiom-inclusion` element it justifies. It also now allows an optional `xml:id` attribute	??
`phrase`	new	used to mark up phrases in `CMP`s and supply them with identifiers and links to context that can be used for presentation and referencing.	??
`presentation`	cha	The `theory` is not allowed any more, to refer to a symbol outside its theory use its `xml:id` attribute. The element now also allows a mutilingual `CMP` group, so that it can be used as a notation definition element in mathematical vernacular.	??
`private`	cha	The `replaces` attribute is now called `reformulates`. The attributes `pto` and `pto-version` have moved to the `data` element. The attribute `type` has been removed and optional attributes `theory`, `generated-from`, and `generated-via` have been added.	202
`proof`	lib	The `for` attribute is now optional to allow for proofs as objects of mathematical discourse. Furthermore, it can now have `generated-from` and `generated-via` attributes.	177

proofobject	lib	The for attribute is now optional to allow for proofs as objects of mathematical discourse. Furthermore, it can now have generated-from and generated-via attributes.	185
recognizer	cha	The role attribute was fixed to object. The commonname child has been replaced by an initial metadata element.	173
ref	aug	ref now has an optional xml:id attribute that identifies it.	??
selector	cha	The role attribute was fixed to object. The commonname child has been replaced by an initial metadata element.	173
solution	cha	the solution element now allows arbitrary OMDoc top-level elements as children. Furthermore, it can now have a theory, generated-from, and generated-via attributes.	209
sortdef	cha	The role attribute was fixed to sort. The type from the adt element is now on the sortdef element. The commonname child has been replaced by an initial metadata element.	172
dc:subject	aug	The dc:subject can now have the optional dc:id, and CSS attributes	114
style	aug	The style element now allows a map element in the body	??
symbol	cha	may no longer contain selector, since it only makes sense for constructors in data types. The kind attribute has been renamed to role for compatibility with OpenMath2 and can have the additional values binder, attribution, semantic-attribution, and error corresponding to the OpenMath 2 roles. Furthermore, an optional attribute generated-via has been added to allow generation via a theory morphism.	152
term	new	the term element can appear in mathematical text and contain it. It is used to link technical terms to symbols defined in content dictionaries via its cd and name attributes.	143

`theory`	cha	the `theory` element loses the `CMP` and `commonname` children, use the Dublin Core metadata elements `dc:description` and `dc:subject` instead. The `theory` element also gains the optional `cdbase` attribute to specify the disambiguating string prescribed for content dictionaries by the OpenMath2 standard. The `xml:id` is now optional, it only needs to be specified, if the theory has constitutive elements. Finally, the element has gained the optional attributes `cdurl`, `cdbase`, `cdreviewdate`, `cdversion`, `cdrevision`, and `cdstatus` attributes for encoding the management metadata of OpenMath content dictionaries.	165
`dc:title`	aug	The `dc:title` can now have the optional `dc:id`, and CSS attributes.	113
`tgroup`	new	The `tgroup` can be used to structure theories like documents.	**??**
`type`	aug	the `type` element now has the optional `just-by` and `theory` attribute. The first one points to an assertion or axiom that justifies the type judgment, the second specifies the home theory. The `system` attribute is now optional. Furthermore, the `type` element can have two math objects as children. If it does, then it is a term declaration, i.e. the first element is interpreted as a mathematical object and the second one is interpreted as its type. Finally, it can now have `generated-from` and `generated-via` attributes.	155
`theory-inclusion`	aug	the `theory-inclusion` element can now have `obligation` and `decomposition` children that justify it. Furthermore, it can now have a `theory`, `generated-from`, and `generated-via` attributes. New optional attributes `conservativity` and `conservativity-just` for stating and justifying conservativity.	**??**
`theory`	aug	the `theory` element can now be nested.	165

`use`	cha	can now contain `element`, `text`, `recurse`, `map`, and `value-of` to specify XML content. We have deprecated the `larg-group` and `rarg-group` attributes, since they were never used.	??
`value`	aug	this element is no longer used, the value of a recursive equation is determined by the position as the second child.	
`with`	ren	the role of this element is now taken by the `phrase` element.	??
`xslt`	cha	the content of this element need not be escaped any more, it is now a valid XSLT fragment.	??

A.3 Changes from 1.0 to 1.1

Version 1.1 was mainly a bug-fix release that has become necessary by the experiments of encoding legacy material in OMDoc. The changes are relatively minor, mostly added optional fields. The only non-conservative changes concern the `private`, `hypothesis`, `sortdef` and `signature` elements. OMDoc files can be upgraded to version 1.1 with the XSLT style sheet `https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/xsl/omdoc1.0adapt1.1.xsl`.

element	state	comments	cf.
`attribute`	new	presentation of attributes for XML elements	??
`alternative`	cha	new form of the `alternative-def` element, it can now also used as an alternative to `axiom`. Compared to `alternative-def` it has a new optional attribute `generated-by` to show that an assertion is generated by expanding a some other element like `adt`.	162
`alternative-def`	dep	new form is `alternative`, since there can be alternative `axiom`s too.	
`argument`	cha	attribute `sort` is now of type `IDREF`, since it must be local in the definition.	172
`assertion`	aug	more values for the `type` attribute, new optional attribute `generated-by` to show that an assertion is generated by expanding a `definition` or an `adt`. New optional attribute `just-by`.	158
`assertion-just`	dep	this is now `obligation`	

`axiom`	aug	new optional attribute `generated-by` to show that an axiom is generated by expanding a `definition`.	154
`axiom-inclusion`	cha	now allows a `CMP` group for descriptive text, includes a set of `obligation` elements instead of an `assertion-just`. The `timestamp` attribute is deprecated, use `dc:date` with appropriate `action` instead	196
`CMP`	cha	the attribute `format` is now deprecated, it makes no sense, since we are more strict and consistent about `CMP` content. `CMP` now allows an optional `id` attribute.	138
`code`	cha	Attributes `width` and `height` now in `omlet`, got attributes `classid` and `codebase` from `private`. Attribute `format` moved to `data` children. The multilingual group of `CMP` elements for description is deprecated, use `metadata/dc:description` instead. Child element `data` may appear multiple times (with different values of the `format`).	202
`constructor`	aug	new optional child `recognizer` for a recognizer predicate	172
`Coverage`	dep	this Dublin Core element specifies the place or time which the publication's contents addresses. This does not seem appropriate for the mathematical content of OMDoc.	
`data`	aug	new optional attributes `size` to specify the size of the data file that is referenced by the `href` attribute and `format` for the format the data is in.	203
`dc:date`	aug	new optional `who` attribute that can be used to specify who did the `action` on this date.	114
`Translator`	dep	this element is not part of Dublin Core, it got into OMDoc by mistake, we use `dc:contributor` with `role=trl` for this.	114
`decomposition`	aug	has a new required `id` attribute. It is no longer a child of `theory-inclusion`, but specifies which `theory-inclusion` it justifies by the new required attribute `for`.	??

`definition`	aug	new optional children `measure` and `ordering` to specify termination of recursive definitions. New optional attribute `generated-by` to show that it is generated by expanding a `definition`.	155
`element`	new	presentation of XML elements	??
`FMP`	aug	now allows multiple `conclusion` elements, to represent general Gentzen-type sequents (not only natural deduction.) `FMP` now allows an optional `id` attribute.	147
`hypothesis`	cha	new required attribute `discharged-in` to specify the `derive` element that discharges this hypothesis.	179
`measure`	new	specifies a measure function (as an OMOBJ)	157
`metadata`	aug	new optional attribute `inherits` allows to inherit metadata from other declarations	100
`method`	cha	first child that used to be an `om:OMSTR` or `ref` element is now moved into a required `xref` attribute that holds an URI that points to the element that defines the method. The `om:OMOBJ` content of the other children (they were `parameter` elements) is now directly included in the `method` element.	180
`obligation`	new	takes over the role of `assertion-just`.	
`omgroup`	aug	also allows the elements that can only appear in `theory` elements, so that `omgroup`s can also be used for grouping inside `theory` elements. The `type` attribute is now restrained to one of `narrative`, `sequence`, `alternative`, `contrast`.	166
`omlet`	aug	obtained attributes `width` and `height` from `private`. New optional attributes `action` for the action to be taken when activated, and `data` a URIref to data in a private element. New optional attribute `type` for the type of the applet.	205
`omstyle`	new	for specifying the style of OMDoc elements	??
`omtext`	cha	the `from` is deprecated, we only leave the `for` attribute, to specify the referential character of the `type`.	145
`ordering`	new	specifies a well-founded ordering (as an OMOBJ)	157

`parameter`	dep	the `om:OMOBJ` element child is now directly a child of `method`	
`pattern`	cha	the child can be an arbitraryOpenMath element.	
`premise`	cha	new optional attribute `rank` for the importance in the inference rule. The old `href` attribute is renamed to `xref` to be consistent with other cross-referencing.	
`presentation`	aug	New attribute `xref` that allows to inherit the information from another `presentation` element. New attribute `theory` to specify the theory the symbol is from; without this, referencing in OMDoc is not unique. The `parent` attribute has been renamed to `role` and now takes the values `applied`, `binding`, and `key`, since we want to be less OpenMath-centric	??
`private`	cha	new optional attribute `for` to point to an OMDoc element it provides data for. As a consequence, `private` elements are no longer allowed in other OMDoc elements, only on top-level. New attribute `replaces` as a pointer to the OMDoc elements that are replaced by the system-specific information in this element. Old attributes `width` and `height` now in `omlet`. Attribute `format` moved to `data` children. The descriptive `CMP` elements are deprecated, use `metadata/dc:description` instead. Child element `data` may appear multiple times (with different values of the `format`). The attributes `classid` and `codebase` are deprecated, since they only make sense on the `code` element.	q 202
`proof`	cha	attribute `theory` is now optional, since the element can appear inside a `theory` element.	177
`proofobject`	cha	attribute `theory` is now optional, since the element can appear inside a `theory` element.	177
`recognizer`	new	specifies the recognizer predicate of a sort.	173
`recurse`	new	recursive calls to presentation in `style`.	??
`ref`	cha	attribute `kind` renamed to `type`.	??

`selector`	cha	the old `type` attribute (had values `total` and `partial`) is deprecated, its duty is now carried by an attribute `total` (values `yes` and `no`).	173
`signature`	dep	for the moment	
`sortdef`	cha	has a mandatory `name` attribute, otherwise the defined symbol has no name.	172
`style`	new	allows to specify style information in `presentation` and `omstyle` elements using a simplified OMDoc-internalized version of XSLT.	??
`symbol`	aug	new optional attribute `generated-by` to show that it is generated by expanding a `definition`.	152
`text`	new	presentation of text in `omstyle`.	??
`theory-inclusion`	cha	now allows `CMP` group for descriptive text, no longer has a `decomposition` child, this is now attached by its `for` attribute. The `timestamp` attribute is deprecated, use `dc:date` with appropriate `action` instead.	??
`type`	aug	can now also appear on top-level. Has an optional `id` attribute for identification, and an optional `for` attribute to point to a `symbol` element it declares type information for.	155
`use`	aug	New attribute `element` allows to specify that the content should be encased in an XML element with the attribute-value pairs specified in the string specified in the attribute `attributes`.	??
`value-of`	new	presentation of values in `style`.	??
`with`	new	used to supply fragments of text in `CMP`s with `style` and `id` attributes that can be used for presentation and referencing.	??
`xslt`	new	allows to embed XSLT into `presentation` and `omstyle` elements.	??

B

Quick-Reference Table to the OMDoc Elements

Element	p.	Mod.	Required Attribs	Optional Attribs	DC	Content
adt	171	ADT		xml:id, type, style, class, theory, generated-from, generated-via	+	sortdef+
alternative	162	ST	for, entailed-by, entails, entailed-by-thm, entails-thm	xml:id, type, theory, generated-from, generated-via, uniqueness, exhaustivity, consistency, existence, style, class	+	CMP*, (FMP|requation*|(OMOBJ |m:math |legacy)*)
answer	210	QUIZ	verdict	xml:id, style, class	+	CMP*, FMP*
m:apply	130	MML		id, xlink:href	−	bvar?,⟪*CMel*⟫*
argument	172	ADT	sort		+	selector?
assertion	158	ST		xml:id, type, theory, generated-from, generated-via, style, class	+	CMP*, FMP*
assumption	148	MTXT		xml:id, inductive, style, class	+	CMP*, (OMOBJ |m:math |legacy)?
attribute	??	PRES	name		−	(value-of|text)*
axiom	154	ST	name	xml:id, type, generated-from, generated-via, style, class	+	CMP*, FMP*
axiom-inclusion	196	CTH	from, to	xml:id, style, class, theory, generated-from, generated-via	+	morphism?, (path-just|obligation*)
m:bvar	130	MML		id, xlink:href	−	ci*

m:ci	129	MML		id, xlink:href	–	PCDATA
m:cn	129	MML		id, xlink:href	–	([0-9]|,|.) (*|e([0-9]|,|.)*)?
choice	210	QUIZ		xml:id, style, class	+	CMP*, FMP*
CMP	138	MTXT		xml:lang, xml:id	–	(text| OMOBJ |m:math |legacy | with | term | omlet)*
code	202	EXT		xml:id, for, theory, generated-from, generated-via, requires, style, class	+	input?, output?, effect?, data+
conclusion	148	MTXT		xml:id, style, class	+	CMP*, (OMOBJ |m:math |legacy)?
constructor	172	ADT	name	type, scope, style, class, theory, generated-from, generated-via	+	argument*, recognizer?
dc:contributor	114	DC		xml:id, role, style, class	–	⟪*text*⟫
dc:creator	114	DC		xml:id, role, style, class	–	⟪*text*⟫
m:csymbol	129	MML	definitionURL	id, xlink:href	–	EMPTY
data	203	EXT		format, href, size, original	–	<![CDATA[...]]>
dc:date	114	DC		action, who	–	ISO 8601 norm
dd	??	RT		xml:id, style, class, index, verbalizes	+	CMPcontent
di	??	RT		xml:id, style, class, index, verbalizes	+	dt+,dd*
dl	??	RT		xml:id, style, class, index, verbalizes	+	li*
dt	??	RT		xml:id, style, class, index, verbalizes	+	CMPcontent
decomposition	??	DG	links	theory, generated-from, generated-via	–	EMPTY
definition	155	ST	xml:id, for	uniqueness, existence, consistency, exhaustivity, type, generated-from, generated-via, style, class	+	CMP*, (FMP| requation+| OMOBJ |m:math |legacy)?, measure?, ordering?
dc:description	114	DC		xml:lang	–	CMPcontent
derive	178	PF		xml:id, style, class	–	CMP*, FMP?, method?

effect	204	EXT		xml:id, style, class	–	CMP*,FMP*
element	??	PRES	name	xml:id, cr, ns	–	(attribute| element| text| recurse)*
example	163	ST	for	xml:id, type, assertion, proof, style, class, theory, generated-from, generated-via	+	CMP*| (OMOBJ |m:math |legacy)?
exercise	209	QUIZ		xml:id, type, for, from, style, class, theory, generated-from, generated-via	+	CMP*, FMP*, hint?, (solution*|mc*)
FMP	147	MTXT		logic, xml:id	–	(assumption*, conclusion*)|OMOBJ |m:math |legacy
dc:format	115	DC			–	fixed: "application/omdoc+xml"
hint	209	QUIZ		xml:id, style, class, theory, generated-from, generated-via	+	CMP*, FMP*
hypothesis	179	PF		xml:id, style, class, inductive	–	CMP*, FMP*
dc:identifier	115	DC		scheme	–	ANY
ide	142	RT	index	xml:id,sort-by,see, seealso, links, style, class		idp*
idp	142	RT		xml:id,sort-by,see, seealso, links, style, class		CMPcontent
idt	142	RT		style, class	–	CMPcontent
idx	142	RT		xml:id,sort-by,see, seealso, links, style, class		idt?,idp+
ignore	101	DOC		type, comment	–	ANY
imports	166	CTH	from	xml:id, type, style, class	+	morphism?
inclusion	195	CTH	for	xml:id	–	
input	204	EXT		xml:id, style, class	–	CMP*,FMP*
insort	172	ADT	for		–	
dc:language	115	DC			–	ISO 8601 norm
li	??	RT		xml:id, style, class, index, verbalizes	–	Math Vernacular
cc:license	118	CC		jurisdiction	–	permissions, prohibitions, requirements

link	??	RT		xml:id, style, class, index, verbalizes	–	Math Vernacular
m:math	129	MML		id, xlink:href	–	⟪*CMel*⟫+
mc	210	QUIZ		xml:id, style, class, theory, generated-from, generated-via	–	choice, hint?, answer
measure	157	ST		xml:id	–	OMOBJ |m:math |legacy
metadata	100	DC			–	(dc-element)*
method	180	PF	xref		–	(OMOBJ |m:math |legacy| premise | proof | proofobject)*
morphism	191	CTH		xml:id, base, consistency, exhaustivity, type, hiding, style, class	–	requation*, measure?, ordering?
note	141	RT		type,xml:id, style, class, index, verbalizes	–	Math Vernacular
obligation	194	CTH	induced-by, assertion	xml:id	–	EMPTY
om:OMA	122	OM		id, cdbase	–	⟪*OMel*⟫*
om:OMATTR	124	OM		id, cdbase	–	⟪*OMel*⟫
om:OMATP	124	OM		cdbase	–	(OMS, (⟪*OMel*⟫ | om:OMFOREIGN))+
om:OMB	125	OM		id, class, style, class	–	#PCDATA
om:OMBIND	123	OM		id, cdbase	–	⟪*OMel*⟫, om:OMBVAR, ⟪*OMel*⟫?
om:OMBVAR	124	OM			–	(om:OMV | om:OMATTR)+
om:OMFOREIGN	125	OM		id, cdbase	–	ANY
omdoc	98	DOC		xml:id,type, version, style, class, xmlns, theory, generated-from, generated-via	+	(top-level element)*
om:OME	125	OM		xml:id	–	(⟪*OMel*⟫)?
om:OMR	126	OM	href		–	
om:OMF	125	OM		id, dec, hex	–	#PCDATA
omgroup	166	DOC		xml:id, type, style, class, modules, theory, generated-from, generated-via	+	top-level element*
ol	??	RT		xml:id, style, class, index, verbalizes	–	li*
om:OMI	125	OM		id, class, style	–	[0-9]*

omlet	205	EXT		id, argstr, type, function, action, data, style, class	+	ANY
om:OMOBJ	122	OM		id, cdbase, class, style	−	⟪*OMel*⟫?
omstyle	??	PRES	element	for, xml:id, xref, style, class	−	(style|xslt)*
om:OMS	122	OM	cd, name	class, style	−	EMPTY
omtext	145	MTXT		xml:id, type, for, from, style, theory, generated-from, generated-via	+	CMP+, FMP?
om:OMV	122	OM	name	class, style	−	EMPTY
ordering	157	ST		xml:id	−	OMOBJ |m:math |legacy
output	204	EXT		xml:id, style, class	−	CMP*,FMP*
p	??	RT		xml:id, style, class, index, verbalizes	−	Math Vernacular
param	207	EXT	name	value, valuetype	−	EMPTY
path-just	??	DG	local, globals	for, xml:id	−	EMPTY
cc:permissions	119	CC		reproduction, distribution, derivative_works	−	EMPTY
premise	180	PF	xref		−	EMPTY
presentation	??	PRES	for	xml:id, xref, fixity, role, lbrack, rbrack, separator, bracket-style, style, class, precedence, crossref-symbol	−	(use | xslt | style)*
private	202	EXT		xml:id, for, theory, generated-from, generated-via, requires, reformulates, style, class	+	data+
cc:prohibitions	119	CC		commercial_use	−	EMPTY
proof	177	PF		xml:id, for,theory, generated-from, generated-via, style, class	+	(symbol | definition | omtext | derive | hypothesis)*
proofobject	185	PF		xml:id, for, theory, generated-from, generated-via, style, class	+	CMP*, (OMOBJ |m:math |legacy)
dc:publisher	114	DC		xml:id, style, class	−	ANY
ref	??	DOC		xref, type	−	ANY

recognizer	173	ADT	name	type, scope, role, style, class	+	
recurse	??	PRES		select	−	EMPTY
dc:relation	115	DC			−	ANY
requation	157	ST		xml:id, style, class	−	(OMOBJ |m:math |legacy),(OMOBJ |m:math |legacy)
cc:requirements	119	CC		notice, copyleft, attribution	−	EMPTY
dc:rights	115	DC			−	ANY
selector	173	ADT	name	type, scope, role, total, style, class	+	
solution	209	QUIZ		xml:id, for, style, class, theory, generated-from, generated-via	+	(CMP*, FMP*) | proof
sortdef	172	ADT	name	role, scope, style, class	+	(constructor|insort)*
dc:source	115	DC			−	ANY
style	??	PRES	format	xml:lang, requires	−	(element | text | recurse | value-of)*
dc:subject	114	DC		xml:lang	−	CMPcontent
symbol	152	ST	name	role, scope, style, class,generated-from,generated-via	+	type*
table	??	RT		xml:id, style, class, index, verbalizes	−	tr*
term	143	MTXT	cd, name	xml:id, role, style, class	−	CMP content
text	??	PRES			−	#PCDATA
td	??	RT		xml:id, style, class, index, verbalizes	−	Math Vernacular
th	??	RT		xml:id, style, class, index, verbalizes	−	Math Vernacular
theory	165	ST	xml:id	cdbase, style, class	+	(statement|theory)*
theory-inclusion	??	CTH	from, to	xml:id, style, class, theory, generated-from, generated-via	+	(morphism, decomposition?)
tr	??	RT		xml:id, style, class, index, verbalizes	−	(td|th)*
dc:title	113	DC		xml:lang	−	CMPcontent
type	155	ST	system	xml:id, for, style, class	−	CMP*, (OMOBJ |m:math |legacy)
dc:type	115	DC			−	fixed: "Dataset" or "Text" or "Collection"

ul	??	RT		xml:id, style, class, index, verbalizes	–	li*
use	??	PRES	format	xml:lang, requires, fixity, lbrack, rbrack, separator, crossref-symbol, element, attributes	–	(use | xslt | style)*
value-of	??	PRES	select		–	EMPTY
phrase	??	MTXT		xml:id, style, class, index, verbalizes, type	–	CMP content
xslt	??	PRES	format	xml:lang, requires	–	XSLT fragment

C

Quick-Reference Table to the OMDoc Attributes

Attribute	*element*	Values
`action`	`dc:date`	`unspecified`
	specifies the action taken on the document on this date.	
`action`	`omlet`	`execute, display, other`
	specifies the action to be taken when executing the `omlet`, the value is application-defined.	
`actuate`	`omlet`	`onPresent, onLoad, onRequest, other`
	specifies the timing of the action specified in the `action` attribute	
`assertion`	`example`	
	specifies the assertion that states that the objects given in the example really have the expected properties.	
`assertion`	`obligation`	
	specifies the assertion that states that the translation of the statement in the source theory specified by the `induced-by` attribute is valid in the target theory.	
`attributes`	`use`	
	the attribute string for the start tag of the XML element substituted for the brackets (this is specified in the `element` attribute).	
`attribution`	`cc:requirements`	`required, not_required`
	Specifies whether the copyright holder/author must be given credit in derivative works	
`base`	`morphism`	
	specifies another morphism that should be used as a base for expansion in the definition of this morphism	
`bracket-style`	`presentation, use`	`lisp, math`
	specifies whether a function application is of the form $f(a,b)$ or (fab)	
`cd`	`om:OMS`	
	specifies the content dictionary of an OpenMath symbol	

`cd`	`term`	
	specifies the content dictionary of a technical term	
`cdbase`	`om:*`	
	specifies the base URI of the content dictionaries used in an OpenMath object	
`cdreviewdate`	`theory`	
	specifies the date until which the content dictionary will remain unchanged	
`cdrevision`	`theory`	
	specifies the minor version number of the content dictionary	
`cdstatus`	`theory`	`official, experimental, private, obsolete`
	specifies the content dictionary status	
`cdurl`	`theory`	
	the main URL, where the newest version of the content dictionary can be found	
`cdversion`	`theory`	
	specifies the major version number of the content dictionary	
`comment`	`ignore`	
	specifies a reason why we want to ignore the contents	
`crossref-symbol`	`presentation, use`	`all, brackets, lbrack, no, rbrack, separator, yes`
	specifies whether cross-references to the symbol definition should be generated in the output format.	
`class`	`*`	
	specifies the CSS class	
`commercial_use`	`cc:permissions`	`permitted, prohibited`
	specifies, whether commercial use of the document with this license is permitted	
`consistency`	`morphism, definition`	`OMDoc reference`
	points to an `assertion` stating that the cases are consistent, i.e. that they give the same values, where they overlap	
`copyleft`	`cc:restrictions`	`required, not_required`
	specifies whether derived works must be licensed with the same license as the current document.	
`cr`	`element`	`yes/no`
	specifies whether an `xlink:href` cross-reference should be set on the result element.	
`cref`	`om:*`	`URI reference`
	extra attribute for cross-references in parallel markup	
`crid`	`element`	`XPath expression`
	the path to the sub-element that corresponds to the result element.	
`crossref-symbol`	`presentation, use`	`no, yes, brackets, separator, lbrack, rbrack, all`

	specifies which generated presentation elements should carry cross-references to the definition.	
data	omlet	
	points to a private element that contains the data for this omlet	
definitionURL	m:*	URI
	points to the definition of a mathematical concept	
derivative_works	cc:permissions	permitted, not_permitted
	specifies whether the document may be used for making derivative works.	
distribution	cc:permissions	permitted,not_permitted
	specifies whether distribution of the current document fragment is permitted.	
element	use	
	the XML element tags to be substituted for the brackets.	
element	omstyle	
	the XML element, the presentation information contained in the omstyle element should be applied to.	
encoding	m:annotation,om:OMFOREIGN	MIME type of the content
	specifies the format of the content	
entails, entailed-by	alternative	
	specifies the equivalent formulations of a definition or axiom	
entails-thm, entailed-by-thm	alternative	
	specifies the entailment statements for equivalent formulations of a definition or axiom	
exhaustivity	morphism, definition	OMDoc reference
	points to an assertion that states that the cases are exhaustive.	
existence	definition	OMDoc reference
	points to an assertion that states that the symbol described in an implicit definition exists	
fixity	presentation	assoc, infix, postfix, prefix
	specifies where the function symbol-of a function application should be displayed in the output format	
function	omlet	
	specifies the function to be called when this omlet is activated.	
format	data	
	specifies the format of the data specified by a data element. The value should e.g. be a MIME type [FB96].	
for	*	
	can be used to reference an element by its unique identifier given in its xml:id attribute.	
formalism	legacy	URI reference
	specifies the formalism in which the content is expressed	
format	legacy	URI reference
	specifies the encoding format of the content	

`format`	`use`	`cmml, default, html, mathematica, pmml, TeX,...`
	specifies the output format for which the notation is specified	
`from`	`imports, theory-inclusion, axiom-inclusion`	`URI reference`
	pointer to source `theory` of a theory morphism	
`from`	`omtext`	`URI reference`
	points to the source of a relation given by a text type	
`generated-from`	`top-level elements`	`URI reference`
	points to a higher-level syntax element, that generates this statement.	
`generated-via`	`top-level elements,...`	`URI reference`
	points to a theory-morphism, via which it is translated from the element pointed to by the `generated-from` attribute.	
`globals`	`path-just`	
	points to the `axiom-inclusion`s or `theory-inclusion`s that is the rest of the inclusion path.	
`hiding`	`morphism`	
	specifies the names of symbols that are in the domain of the morphism	
`href`	`data, link, om:OMR`	`URI reference`
	a URI to an external file containing the data.	
`xml:id`		
	associates a unique identifier to an element, which can thus be referenced by an `for` or `xref` attribute.	
`xml:base`		
	specifies a base URL for a resource fragment	
`index`	`on RT elements`	
	A path identifier to establish multilingual correspondence	
`induced-by`	`obligation`	
	points to the statement in the source theory that induces this proof obligation	
`inductive`	`assumption, hypothesis`	`yes, no`
	Marks an assumption or hypothesis inductive.	
`jurisdiction`	`cc:license`	`IANA Top level Domain designator`
	specifies the country of jurisdiction for a Creative Commons license	
`just-by`	`type`	
	points to an assertion that states the type property in question.	
`role`	`symbol, constructor, recognizer, selector, sortdef`	`object, type, sort, binder, attribution, semantic-attribution, error`
	specifies the role (possible syntactic roles) of the symbol in this declaration.	

role	dc:creator,dc:contributor	MARC relators
	specifies the role of a person who has contributed to the document	
role	presentation	applied, binding, key
	specifies which role of the symbol is annotated with notation information	
lbrack	presentation, use	
	the left bracket to use in the notation of a function symbol	
links	decomposition	
	specifies a list of theory- or axiom-inclusions that justify (by decomposition) the theory-inclusion specified in the for attribute.	
local	path-just	
	points to the axiom-inclusion that is the first element in the path.	
logic	FMP	token
	specifies the logical system used to encode the property.	
modules	omdoc, omgroup	module and sub-language shorthands, URI reference
	specifies the modules or OMDoc sub-language used in this document fragment	
name	om:OMS, om:OMV, symbol, term	
	the name of a concept referenced by a symbol, variable, or technical term.	
name	attribute, element	
	the local name of generated element.	
name	param	
	the name of a parameter for an external object.	
notice	cc:requirements	required, not_required
	specifies whether copyright and license notices must be kept intact in distributed copies of this document	
ns	element, attribute	URI
	specifies the namespace URI of the generated element or attribute node	
original	data	local, external
	specifies whether the local copy in the data element is the original or the external resource pointed to by the href attribute.	
parameters	adt	
	The list of formal parameters of a higher-order abstract data type	
precedence	presentation	
	the precedence of a function symbol (for elision of brackets)	
just-by	assertion	
	specifies a list of URIs to proofs or other justifications for the proof status given in the status attribute.	

pto, pto-version	private, code	
	specifies the system and its version this data or code is private to	
rank	premise	
	specifies the rank (importance) of a premise	
rbrack	presentation, use	
	the right bracket to use in the notation of a function symbol	
reformulates	private	
	points to a set of elements whose content is reformulated by the content of the private element for the system.	
reproduction	cc:permissions	permitted,not_permitted
	specifies whether reproduction of the current document fragment is permitted by the licensor	
requires	private, code, use, xslt, style	URI reference
	points to a code element that is needed for the execution of this data by the system.	
role	dc:creator, dc:collaborator	aft, ant, aqt, aui, aut, clb, edt, ths, trc, trl
	the MARC relator code for the contribution of the individual.	
role	phrase, term	
	the role of the phrase annotation	
role	presentation	applied, binding, key
	specifies for which role (as the head of a function application, as a binding symbol, or as a key in a attribution, or as a stand-alone symbol (the default)) of the symbol presentation is intended	
scheme	dc:identifier	scheme name
	specifies the identification scheme (e.g. ISBN) of a resource	
scope	symbol	global, local
	specifies the visibility of the symbol declared. This is a very crude specification, it is better to use theories and importing to specify symbol accessibility.	
select	map, recurse, value-of	XPATH expression
	specifies the path to the sub-expression to act on	
separator	presentation, use	
	the separator for the arguments to use in the notation of a function symbol	
show	omlet	new, replace, embed, other
	specifies the desired presentation of the external object.	
size	data	
	specifies the size the data specified by a data element. The value should be number of kilobytes	
sort	argument	
	specifies the argument sort of the constructor	
style	*	

	specifies a token for a presentation style to be picked up in a `presentation` element.	
`system`	`type`	
	A token that specifies the logical type system that governs the type specified in the type element.	
`theory`	`*`	
	specifies the home theory of an OMDoc statement.	
`to`	`theory-inclusion`, `axiom-inclusion`	
	specifies the target theory	
`total`	`selector`	`no, yes`
	specifies whether the symbol declared here is a total or partial function.	
`type`	`adt`	`free, generated, loose`
	defines the semantics of an abstract data type `free` = no junk, no confusion, `generated` = no junk, `loose` is the general case.	
`type`	`assertion`	`theorem, lemma, corollary, conjecture, false-conjecture, obligation, postulate, formula, assumption, proposition`
	tells you more about the intention of the assertion	
`type`	`definition`	`implicit, inductive, obj, recursive, simple`
	specifies the definition principle	
`type`	`derive`	`conclusion, gap`
	singles out special proof steps: conclusions and gaps (unjustified proof steps)	
`type`	`example`	`against, for`
	specifies whether the objects in this example support or falsify some conjecture	
`type`	`ignore`	
	specifies the type of error, if ignore is used for in-place error markup	
`type`	`imports`	`global, local`
	`local` imports only concern the assumptions directly stated in the theory. `global` imports also concern the ones the source theory inherits.	
`type`	`morphism`	
	specifies whether the morphism is recursive or merely pattern-defined	
`type`	`omgroup, omdoc`	`enumeration, sequence, itemize`
	the first three give the text category, the second three are used for generalized tables	

`type`	`omtext`	`abstract, antithesis, comment, conclusion, elaboration, evidence, introduction, motivation, thesis`
	a specification of the intention of the text fragment, in reference to context.	
`type`	`phrase`	
	the linguistic or mathematical type of the phrase	
`uniqueness`	`definition`	`URI reference`
	points to an `assertion` that states the uniqueness of the concept described in an implicit definition	
`value`	`param`	
	specifies the value of the parameter	
`valuetype`	`param`	
	specifies the type of the value of the parameter	
`verbalizes`	`on RT elements`	`URI references`
	contains a whitespace-separated list of pointers to OMDoc elements that are verbalized	
`verdict`	`answer`	
	specifies the truth or falsity of the answer. This can be used e.g. by a grading application.	
`version`	`omdoc`	`1.2`
	specifies the version of the document, so that the right DTD is used	
`version`	`cc:license`	
	specifies the version of the Creative Commons license that applies, if not present, the newest one is assumed	
`via`	`inclusion`	
	points to a theory-inclusion that is required for an actualization	
`who`	`dc:date`	
	specifies who acted on the document fragment	
`xml:lang`	`CMP, dc:*`	`ISO 639 code`
	the language the text in the element is expressed in.	
`xml:lang`	`use, xslt, style`	`whitespace-separated list of ISO 639 codes`
	specifies for which language the notation is meant	
`xlink:*`	`om:OMR, m:*`	`URI reference`
	specify the link behavior on the elements	
`xref`	`ref, method, premise`	`URI reference`
	Identifies the resource in question	
`xref`	`presentation, omstyle`	`URI reference`
	The element, this URI points to should be in the place of the object containing this attribute.	

D

The RelaxNG Schema for OMDoc

We reprint the modularized RelaxNG schema for OMDoc here. It is available at `http://omdoc.org/rnc` and consists of separate files for the OMDoc modules, which are loaded by the schema driver `omdoc.rnc` in this directory. We will use the abbreviated syntax for RelaxNG here, since the XML syntax, document type definitions and even XML schemata can be generated from it by standard tools.

The RelaxNG schema consists of the grammar fragments for the modules (see Appendices D.2 to D.14), a definition of the most common attributes that occur in several of the modules (see Appendix D.1), and the sub-language driver files which we will introduce next.

D.1 Common Parts of the Schema

The RelaxNG grammar for OMDoc separates out declarations for commonly used objects.

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.3) Common attributes
# original in  http://github.com/KWARC/OMDoc-1.3/schema/rnc
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2015 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://omdoc.org/ns"
namespace local = ""

# all the explicitly namespaced attributes, except xml:lang, which
# is handled explicitly
nonlocal.attribs = attribute * - (local:* | xml:*) {xsd:string}*

# the attributes for CSS and PRES styling
css.attribs = attribute style {xsd:string}? & attribute class {xsd:string}?

omdocref = xsd:anyURI          # an URI reference pointing to an OMDoc fragment
omdocrefs = list {xsd:anyURI*} # a whitespace-separated list of omdocref

xref.attrib = attribute xref {omdocref}
tref = attribute tref {omdocref}
```

```
# for the moment, we may get regexp at some point.
curie = xsd:string
curies = xsd:string
safecurie = xsd:string

about.attrib = attribute about {xsd:anyURI|safecurie}
xmlbase.attrib = attribute xml:base {xsd:anyURI}
xmlid.attrib = attribute xml:id {xsd:ID}

idrest . attribs = css. attribs & nonlocal.attribs & about.attrib? & xmlbase.attrib?

id . attribs = xmlid.attrib? & idrest. attribs

toplevel . attribs = id. attribs , attribute generated−from {omdocref}?

iso639 = ”aa” | ”ab” | ”af” | ”am” | ”ar” | ”as” |
”ay” | ”az” | ”ba” | ”be” | ”bg” | ”bh” | ”bi” | ”bn” | ”bo” | ”br” | ”ca” | ”co”
| ”cs” | ”cy” | ”da” | ”de” | ”dz” | ”el” | ”en” | ”eo” | ”es” | ”et” | ”eu” |
”fa” | ”fi” | ”fj” | ”fo” | ”fr” | ”fy” | ”ga” | ”gd” | ”gl” | ”gn” | ”gu” | ”ha”
| ”he” | ”hi” | ”hr” | ”hu” | ”hy” | ”ia” | ”ie” | ”ik” | ”id” | ”is” | ”it” |
”iu” | ”ja” | ”jv” | ”ka” | ”kk” | ”kl” | ”km” | ”kn” | ”ko” | ”ks” | ”ku” | ”ky”
| ”la” | ”ln” | ”lo” | ”lt” | ”lv” | ”mg” | ”mi” | ”mk” | ”ml” | ”mn” | ”mo” |
”mr” | ”ms” | ”mt” | ”my” | ”na” | ”ne” | ”nl” | ”no” | ”oc” | ”om” | ”or” | ”pa”
| ”pl” | ”ps” | ”pt” | ”qu” | ”rm” | ”rn” | ”ro” | ”ru” | ”rw” | ”sa” | ”sd” |
”sg” | ”sh” | ”si” | ”sk” | ”sl” | ”sm” | ”sn” | ”so” | ”sq” | ”sr” | ”ss” | ”st”
| ”su” | ”sv” | ”sw” | ”ta” | ”te” | ”tg” | ”th” | ”ti” | ”tk” | ”tl” | ”tn” |
”to” | ”tr” | ”ts” | ”tt” | ”tw” | ”ug” | ”uk” | ”ur” | ”uz” | ”vi” | ”vo” | ”wo”
| ”xh” | ”yi” | ”yo” | ”za” | ”zh” | ”zu”

xml.lang. attrib = attribute xml:lang {iso639}?

Anything = (AnyElement|text)∗
AnyElement = element ∗ {AnyAttribute,(text | AnyElement)∗}
AnyAttribute = attribute ∗ { text }∗

## useful classes to be extended in the modules
inline . class = empty
omdoc.class = empty
plike . class = empty

## mixed models
inline .model = text & inline.class

metadata.model &= dublincore
```

D.2 Module MOBJ: Mathematical Objects and Text

The RNC module MOBJ includes the representations for mathematical objects and defines the `legacy` element (see Chapter 13 for a discussion). It includes the standard RelaxNG schema for OpenMath (we have reprinted it in Appendix E.1) adding the OMDoc identifier and CSS attributes to all elements. If also includes a schema for MathML (see Appendix E.2).

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.3) Module MOBJ
# original in http://github.com/KWARC/OMDoc−1.3/schema/rnc
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2015 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = ”http://omdoc.org/ns”
```

```
namespace om = ”http://www.openmath.org/OpenMath”
namespace local = ””

name.attrib = attribute name {xsd:NCName}?
triple.att = attribute cdbase {xsd:anyURI}? & name.attrib & attribute cd {xsd:NCName}?

# the legacy element, it can encapsulate the non−migrated formats
legacy.attribs = id.attribs &
                 attribute formalism {xsd:anyURI}? &
                 attribute format {xsd:anyURI}
legacy.model = Anything
legacy = element legacy {tref|(legacy.attribs & legacy.model)}

nonom.attribs = attribute ∗ − (local:∗ | om:∗) {text}∗
extom.attribs = idrest.attribs & nonom.attribs
omobj = grammar {include ”openmath2ext.rnc”
                           common.attributes &= parent extom.attribs}

cmml = grammar {include ”mathml3−common.rnc”
                          include ”mathml3−strict−content.rnc”}

mobj = legacy | omobj | cmml
```

D.3 Module MTXT: Mathematical Text

The RNC module MTXT provides infrastructure for mathematical vernacular (see Chapter 14 for a discussion).

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.3) Module MTXT
# original in  http://github.com/KWARC/OMDoc−1.3/schema/rnc
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2015 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = ”http://omdoc.org/ns”

omdoc.class &= omtext∗

#attribute for is a whitespace−separated list of  URIrefs
for.attrib = attribute for {omdocrefs}
from.attrib = attribute from {omdocref}
mc.class = metadata.class & uses∗ & CMP∗
mcf.class = mc.class & FMP∗

uses.attribs = id.attribs & from.attrib
uses.model = metadata.class
uses = element uses {tref|(uses.attribs & uses.model)}


rsttype = ”abstract” | ”introduction” | ”annote” |
          ”conclusion” | ”thesis” | ”comment” | ”antithesis” |
          ”elaboration” | ”motivation” | ”evidence” | ”note” |
          ”warning” | ”question” | ”answer” | ”transition”

statementtype = ”axiom” | ”definition” | ”example” | ”proof” |
                ”derive” | ”hypothesis” | ”notation”

assertiontype = ”assertion” | ”theorem” | ”lemma” | ”corollary” | ”proposition” |
                ”conjecture” | ”false−conjecture” | ”obligation” |
                ”postulate” | ”formula” | ”assumption” | ”rule” |
                ”observation” | ”remark”

verbalizes.attrib = attribute verbalizes {omdocrefs}
```

```
omtext.type.attrib = attribute type {rsttype | statementtype | assertiontype | xsd:anyURI}
index.attrib = attribute index {xsd:NMTOKEN}
parallel.attribs = verbalizes.attrib? & index.attrib? & omtext.type.attrib?

omtext.attribs = toplevel.attribs &
                 omtext.type.attrib? &
                 for.attrib? &
                 attribute from {omdocref}? &
                 verbalizes.attrib?
omtext.model = mcf.class
omtext = element omtext {tref|(omtext.attribs & omtext.model)}

CMP.attribs = xml.lang.attrib & id.attribs
CMP.model = plike.class
CMP = element CMP {tref|(CMP.attribs & CMP.model)}

role.attrib = attribute role {text}?

term.attribs = id.attribs & role.attrib & triple.att
term.model = inline.model
term = element term {tref|(term.attribs & term.model)}

definiendum.attribs = id.attribs & role.attrib & triple.att
definiendum.model = inline.model & metadata.class
definiendum = element definiendum {tref|(definiendum.attribs & definiendum.model)}

forids.attrib = attribute for {xsd:IDREFS}
declaration.attribs = id.attribs & role.attrib? & forids.attrib?
declaration.model = text & inline.class & condition* & restriction*
declaration= element declaration {tref|(declaration.attribs & declaration.model)}

condition.attribs = id.attribs & forids.attrib?
condition.model = inline.model
condition= element condition {tref|(condition.attribs & condition.model)}

restriction.attribs = id.attribs & role.attrib & forids.attrib?
restriction.model = inline.model
restriction = element restriction {tref|( restriction.attribs & restriction.model)}

FMP.attribs = id.attribs & attribute logic {xsd:NMTOKEN}?
FMP.model = (assumption*,conclusion*)|mobj
FMP = element FMP {tref|(FMP.attribs & FMP.model)}

assumption.attribs = id.attribs &
                     attribute inductive {"yes" | "no"}?
assumption.model = mobj
assumption = element assumption {tref|(assumption.attribs & assumption.model)}

conclusion.attribs = id.attribs
conclusion.model = mobj
conclusion = element conclusion {tref|(conclusion.attribs & conclusion.model)}

note.attribs = id.attribs & for.attrib? & parallel.attribs & attribute type {xsd:NMTOKEN}?
note.model = plike.class
note = element note {tref|(note.attribs & note.model)}

# index
index.att = attribute sort-by {text}? &
            attribute see {omdocrefs}? &
            attribute seealso {omdocrefs}? &
            attribute links { list {xsd:anyURI*}}?

idx.attribs = id.attribs|xref.attrib
idx.model = idt? & ide+
idx = element idx {tref|(idx.attribs & idx.model)}

ide.attribs = (id.attribs & index.att & index.attrib?) | xref.attrib
```

```
ide.model = idp*
ide = element ide {tref|(ide.attribs & ide.model)}

idt.attribs = id.attribs|xref.attrib
idt.model = inline.model
idt = element idt {tref|(idt.attribs & idt.model)}

idp.attribs = index.att
idp.model = inline.model
idp = element idp {tref|(idp.attribs & idp.model)}

# citations
citation.attribs = id.attribs & attribute bibrefs {text}
citation.model = empty
citation = element citation {tref|(citation.attribs & citation.model)}

# references to OMDoc objects
oref.attribs = id.attribs & attribute href {xsd:anyURI}
oref.model = pre? & post? & fallback?
oref = element oref {tref|(oref.attribs & oref.model)}
pre = element pre {id.attribs & inline.model}
post = element post {id.attribs & inline.model}
fallback = element fallback {id.attribs & inline.model}

# what can go into a mathematical text
op.class = \term* & mobj* & note* & idx* & citation* & oref* &
                 declaration* & definiendum*
inline.class &=op.class
omdoc.class &= oref*
```

D.4 Module DOC: Document Infrastructure

The RNC module DOC specifies the document infrastructure of OMDoc documents (see Chapter 11 for a discussion).

```
# A RelaxNG for Open Mathematical documents (OMDoc 1.3) Module DOC
# original in http://github.com/KWARC/OMDoc-1.3/schema/rnc
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2015 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://omdoc.org/ns"
# extend the stuff that can go into a mathematical text

omdoc.class &= ignore* & tableofcontents*

ignore.attribs = id.attribs &
                     attribute type {xsd:string}? &
                     attribute comment {xsd:string}?
ignore.model = Anything
ignore = element ignore {tref|(ignore.attribs & ignore.model)}

tableofcontents.attribs = attribute level {xsd:nonNegativeInteger}?
tableofcontents.model = empty
tableofcontents = element tableofcontents {tref|(tableofcontents.attribs & tableofcontents.model)}

index.attribs = id.attribs
index.model = empty
index = element index {tref|(index.attribs & index.model)}

bibliography.attribs = id.attribs, attribute files {text}
bibliography.model = empty
bibliography = element bibliography {tref|(bibliography.attribs & bibliography.model)}
```

```
group.attribs = id.attribs,
                         attribute type {xsd:anyURI}?,
                         attribute modules {xsd:anyURI}?,
                         attribute layout {text}?

## The <omdoc> and <omgroup> elements allow frontmatter and backmatter,
## which we will now define
frontmatter = metadata.class & tableofcontents?
backmatter = index? & bibliography?
docstruct.class = omgroup*
omdoc.class &= docstruct.class
mainmatter = omdoc.class

omgroup.attribs = toplevel.attribs & group.attribs
omgroup.model = frontmatter,mainmatter,backmatter
omgroup = element omgroup {tref|(omgroup.attribs & omgroup.model)}

## the model of the document root only differs from <omgroup> in the version attribute
omdoc.attribs = toplevel.attribs & group.attribs &
                          attribute version {xsd:string {pattern = "1.3"}}?
omdoc.model =frontmatter,mainmatter,backmatter
omdoc = element omdoc {tref|(omdoc.attribs & omdoc.model)}


############################## deprecated ################################
# the following is for legacy only, and will be removed soon.
ref.attribs = id.attribs & xref.attrib & attribute type {"include" | "cite"}
ref.model = empty
ref = element ref{ref.attribs & ref.model}

omdoc.class &= ref*
inline.class &= ref*
```

```
# A RelaxNG for Open Mathematical documents (OMDoc 1.3) Module META
# original in  http://github.com/KWARC/OMDoc-1.3/schema/rnc
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2015 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://omdoc.org/ns"

rel.attrib = attribute rel {curies}
rev.attrib = attribute rev {curies}
content.attrib = attribute content {xsd:string}
resource.attrib = attribute resource {xsd:anyURI|safecurie}
property.attrib = attribute property {curies}
datatype.attrib = attribute datatype {curie}
typeof.attrib = attribute typeof {curies}

meta.attribs = id.attribs & property.attrib?& datatype.attrib? & xml.lang.attrib
meta.model = content.attrib | Anything | (content.attrib & Anything)
meta = element meta {tref|(meta.attribs & meta.model)}

mlink.attribs = id.attribs & rel.attrib? & rev.attrib? & resource.attrib?
mlink.class = resource* & mlink* & meta*
mlink.model = attribute href {curie}|mlink.class
mlink = element link {tref|(mlink.attribs,mlink.model)}

resource.attribs = id.attribs & typeof.attrib? & about.attrib?
resource.class = meta* & mlink*
resource = element resource {tref|(resource.attribs & resource.class)}

metadata.class = metadata? & meta* & mlink*
metadata.model = metadata.class
metadata.attribs = id.attribs
```

```
metadata = element metadata {tref|(metadata.attribs & metadata.model)}

rdfa. attribs = rel. attrib? & rev.attrib? & content.attrib? & about.attrib?
                    & resource.attrib? & property.attrib? & datatype.attrib?
                    & typeof.attrib?

id. attribs &= rdfa.attribs
```

D.5 Module DC: Dublin Core Metadata

The RNC module DC includes an extension of the Dublin Core vocabulary for bibliographic metadata, see Sections 12.2 and 12.3 for a discussion.

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.3) Module DC
# original in http://github.com/KWARC/OMDoc−1.3/schema/rnc
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2015 Michael Kohlhase, released under the GNU Public License (GPL)

## common attributes
dc.common = id.attribs & nonlocal.attribs & schemetype.attribs
dc.comlang =dc.common & xml.lang.attrib

## scheme and type e.g. according to http://dimes.lins.fju.edu.tw/dimes/meta−ref/DC−SubElements.html
schemetype.attribs =
      attribute scheme {text}? &
      attribute type {text}?

dublincore = grammar {include "MARCRelators.rnc"
      include "dublincore.rnc"
            {dc.date = parent dc.common &
                              attribute action {xsd:NMTOKEN}? &
                              attribute who {xsd:anyURI}? &
                              (xsd:date|xsd:dateTime)
             dc. identifier = parent tref|(parent dc.common &
                                                      attribute scheme {xsd:NMTOKEN} &
                                                      text)
             dc.type = parent tref|(parent dc.common & ("Dataset" | "Text" | "Collection"))
             dc. inline = parent tref|(parent dc.comlang & parent inline.model)
             dc.text = parent tref|(parent dc.comlang & parent plike.class)
             dc.person = parent tref|(parent dc.common &
                                                   attribute role {MARCRelators}? &
                                                   parent inline .model)
             dc. rights = parent tref|(parent dc.comlang & parent plike.class)
             dc.source = parent plike. class }}

metadata.model &= dublincore
```

D.6 Module ST: Mathematical Statements

The RNC module ST deals with mathematical statements like assertions and examples in OMDoc and provides an infrastructure for mathematical theories as contexts, for the OMDoc elements that fix the meaning for symbols, see Chapter 15 for a discussion.

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.3) Module ST
# original in http://github.com/KWARC/OMDoc−1.3/schema/rnc
```

```
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2015 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://omdoc.org/ns"

omdoc.class &= symbol* & axiom* & definition* & assertion* & type* & alternative* & example*
& theory*

constitutive.attribs = id.attribs & attribute generated−from {omdocref}?
sym.role.attrib = attribute role {"type" | "sort" | "object" |
                                  "binder" | "attribution" | "application" | "constant" |
                                  "semantic−attribution" | "error"}
scope.attrib = attribute scope {"global" | "local"}?
symbol.attribs = scope.attrib &
                 name.attrib &
                 constitutive.attribs &
                 sym.role.attrib?
symbol.model = metadata.class & type*
symbol = element symbol {tref|(symbol.attribs & symbol.model)}

forname.attrib = attribute for { list {xsd:anyURI+}}
axiom.attribs = constitutive.attribs & forname.attrib & attribute type {xsd:string}?
axiom.model = metadata.class & mcf.class
axiom = element axiom {tref|(axiom.attribs & axiom.model)}

#informal definitions
def.informal = attribute type {"informal"}?

#simple definitions
def.simple.attribs = attribute type {"simple"}
def.simple = def.simple.attribs & mobj

#implicit definitions
exists.attrib = attribute existence {omdocref}
unique.attrib = attribute uniqueness {omdocref}
def.implicit.attribs = attribute type {"implicit"} & exists.attrib? & unique.attrib?
def.implicit = def.implicit.attribs & FMP*

exhaust.attrib = attribute exhaustivity {omdocref}
consist.attrib = attribute consistency {omdocref}

def.pattern.attribs = attribute type {"pattern"}? & exhaust.attrib? & consist.attrib?
def.pattern.model = requation*
def.pattern = def.pattern.attribs & def.pattern.model

def.inductive.attribs = attribute type {"inductive"}? & exhaust.attrib? & consist.attrib?
def.inductive.model = requation* & measure? & ordering?
def.inductive = def.inductive.attribs & def.inductive.model

def.eq = def.pattern | def.simple

#all definition forms, add more by extending this.
defs.all = def.informal | def.eq | def.inductive | def.implicit

# 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.
definition.attribs = constitutive.attribs & forname.attrib
definition = element definition {tref|(definition.attribs & mc.class & defs.all)}

requation.attribs = id.attribs
requation.model = CMP? & (mobj,mobj)
requation = element requation {tref|(requation.attribs & requation.model)}

measure.attribs = id.attribs
measure.model = mobj
```

```
measure = element measure {tref|(measure.attribs & measure.model)}

ordering.attribs = id.attribs & attribute terminating {omdocref}?
ordering.model = mobj
ordering = element ordering {tref|(ordering.attribs & ordering.model)}

# the non-constitutive statements, they need a theory attribute
toplevel.attribs &= attribute theory {omdocref}?

ded.status.class = "satisfiable" | "counter-satisfiable" | "no-consequence" |
                   "theorem" | "conter-theorem" | "contradictory-axioms" |
                   "tautologous-conclusion" | "tautology" | "equivalent" |
                   "conunter-equivalent" | "unsatisfiable-conclusion" | "unsatisfiable"

just-by.attrib = attribute just-by {omdocref}
assertion.attribs = toplevel.attribs &
                                      attribute type {assertiontype}? &
                                      attribute status {ded.status.class}? &
                                      just-by.attrib?
assertion.model = mcf.class
assertion = element assertion {tref|(assertion.attribs & assertion.model)}
# the assertiontype has no formal meaning yet, it is solely for human consumption.
# 'just-by' is a list of URIRefs that point to proof objects, etc that justifies the status.

type.attribs = toplevel.attribs & just-by.attrib? &
                     attribute system {omdocref}? &
                     attribute for {omdocref}?
type.model = mc.class, mobj, mobj?
type = element type {tref|(type.attribs & type.model)}

##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)
alternative.attribs = toplevel.attribs & for.attrib &
                     ((attribute equivalence {omdocref},
                                      attribute equivalence-thm {omdocref}) |
                                    (attribute entailed-by {omdocref} &
                                    attribute entails {omdocref} &
                                    attribute entailed-by-thm {omdocref} &
                                    attribute entails-thm {omdocref}))
alternative.model = mc.class & defs.all
alternative = element alternative {tref|(alternative.attribs & alternative.model)}

example.attribs = toplevel.attribs & for.attrib &
                         attribute type {"for" | "against" }? &
                         attribute assertion {omdocref}?
example.model = mc.class,mobj*
example = element example {tref|(example.attribs & example.model)}

theory.attribs = id.attribs &
                        attribute cdurl {xsd:anyURI}? &
                        attribute cdbase {xsd:anyURI}? &
                        attribute cdreviewdate {xsd:date}? &
                        attribute cdversion {xsd:nonNegativeInteger}? &
                        attribute cdrevision {xsd:nonNegativeInteger}? &
                        attribute cdstatus {"official" | "experimental" |"private" | "obsolete"}?
theory.model = metadata.class & omdoc.class & intheory.class
theory = element theory {tref|(theory.attribs & theory.model)}

imports.attribs = id.attribs & from.attrib & attribute conservative {"true" | "false"}?
imports.model = metadata.class
imports = element imports {tref|(imports.attribs & imports.model)}

intheory.class = imports*
```

D.7 Module ADT: Abstract Data Types

The RNC module ADT specifies the grammar for abstract data types in OMDoc, see Chapter 16 for a discussion.

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.3) Module ADT
# original in  http://github.com/KWARC/OMDoc−1.3/schema/rnc
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2015 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://omdoc.org/ns"
omdoc.class &= adt∗

adt.sym.attrib = id.attribs,scope.attrib,name.attrib

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

adt.attribs = toplevel.attribs &
                    attribute parameters {list {xsd:NCName∗}}?
adt.class = sortdef+
adt.model = metadata.class & adt.class
adt = element adt {tref|(adt.attribs & adt.model)}

adttype = "loose" | "generated" | "free"
sortdef.attribs = adt.sym.attrib &
                        attribute role {"sort"}? &
                        attribute type {adttype}?
sortdef.model = metadata.class & constructor∗ & insort∗ & recognizer?
sortdef = element sortdef {tref|(sortdef.attribs &sortdef.model)}

insort.attribs = attribute for {omdocref}
insort.model = empty
insort = element insort {tref|(insort.attribs & insort.model)}

constructor.attribs = adt.sym.attrib & sym.role.attrib?
constructor.model = metadata.class & argument∗
constructor = element constructor {tref|(constructor.attribs & constructor.model)}

recognizer.attribs = adt.sym.attrib & sym.role.attrib?
recognizer.model = metadata.class
recognizer = element recognizer {tref|(recognizer.attribs & recognizer.model)}

argument.attribs = empty
argument.model = type & selector?
argument = element argument {tref|(argument.attribs & argument.model)}

selector.attribs = adt.sym.attrib &
                         sym.role.attrib? &
                         attribute total {"yes" | "no"}?
selector.model = metadata.class
selector = element selector {tref|(selector.attribs & selector.model)}
```

D.8 Module PF: Proofs and Proof objects

The RNC module PF deals with mathematical argumentations and proofs in OMDoc, see Chapter 17 for a discussion.

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.3) Module PF
# original in  http://github.com/KWARC/OMDoc−1.3/schema/rnc
```

```
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2015 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://omdoc.org/ns"

omdocpf.opt.content &= proof* & proofobject*
omdoc.class         &= proof* & proofobject*

proof.attribs = toplevel.attribs & for.attrib?
proof.model = metadata.class & omtext* & symbol* & definition* & derive* & hypothesis*
proof = element proof {tref|(proof.attribs & proof.model)}

proofobject.attribs = proof.attribs
proofobject.model = metadata.class & mobj
proofobject = element proofobject {tref|(proofobject.attribs & proofobject.model)}

derive.attribs = id.attribs & attribute type {"conclusion" | "gap"}?
derive.model = mcf.class & method?
derive       = element derive {tref|(derive.attribs & derive.model)}

hypothesis.attribs = id.attribs & attribute inductive {"yes" | "no"}?
hypothesis.model = mcf.class
hypothesis = element hypothesis {tref|(hypothesis.attribs & hypothesis.model)}

method.attribs = id.attribs & xref.attrib?
method.model = mobj* & premise* & proof* & proofobject*
method = element method {tref|(method.attribs & method.model)}

premise.attribs = xref.attrib & attribute rank {xsd:nonNegativeInteger}?
premise.model = empty
premise = element premise {tref|(premise.attribs & premise.model)}

# The rank of a premise specifies its importance in the inference rule.
# Rank 0 (the default) is a real premise, whereas positive rank signifies
# sideconditions of varying degree.
```

D.9 Module CTH: Complex Theories

The RNC presented in this section deals with the module CTH of complex theories (see Chapter 18 for a discussion).

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.3) Module CTH
# original in http://github.com/KWARC/OMDoc-1.3/schema/rnc
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2015 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://omdoc.org/ns"

constitutive.class &= inclusion*
imports.model &= morphism? &
                 attribute type { "local" | "global"}? &
                 attribute conservativity {"conservative" | "monomorphism" | "definitional"}? &
                 attribute conservativity-just {omdocref}?

toplevel.attribs      &= attribute generated-via {omdocref}?
constitutive.attribs  &= attribute generated-via {omdocref}?

omdoc.class &= theory-inclusion* & axiom-inclusion*
theory-inclusion.justification = obligation* & assertion* & proof*
axiom-inclusion.justification = obligation* & assertion* & proof*

fromto.attrib = from.attrib & attribute to {omdocref}
```

```
# attributes 'to' and 'from' are URIref

morphism.attribs = id.attribs &
                                    attribute hiding {omdocrefs}? &
                                    attribute base {omdocrefs}?
morphism.model = def.eq?
morphism = element morphism {tref|(morphism.attribs & morphism.model)}
# base points to some other morphism it extends

inclusion . attribs = id. attribs & attribute via {omdocref}
inclusion . model = empty
inclusion = element inclusion {tref|( inclusion . attribs & inclusion . model)}
# via points to a theory−inclusion

theory−inclusion.attribs = toplevel. attribs & fromto.attrib
theory−inclusion.model = metadata? & morphism? & theory−inclusion.justification & CMP∗
theory−inclusion = element theory−inclusion {tref|(theory−inclusion.attribs & theory−inclusion.model)}

axiom−inclusion.attribs = toplevel. attribs & fromto.attrib
axiom−inclusion.model = metadata? & morphism? & axiom−inclusion.justification
axiom−inclusion = element theory−inclusion {tref|(axiom−inclusion.attribs & axiom−inclusion.model)}

obligation . attribs = id. attribs &
                                   attribute induced−by {omdocref} &
                                   attribute assertion {omdocref}
obligation . model = empty
obligation = element obligation {tref|( obligation . attribs & obligation . model)}
# attribute 'assertion' is a URIref, points to an assertion
# that is the proof obligation induced by the axiom or definition
# specified by 'induced−by'.

## we allow morphisms now that we have them instead of a sequence of objects in examples.
example.model |= mc.class,morphism
```

D.10 Module DG: Development Graphs

The RNC presented in this section deals with the module CTH of development graphs (see Section ?? for a discussion).

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.3) Module CTH
# original in http://github.com/KWARC/OMDoc−1.3/schema/rnc
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2015 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://omdoc.org/ns"

constitutive . class &= inclusion∗
imports.model &= morphism? &
                             attribute type { "local" | "global"}? &
                             attribute conservativity {"conservative" | "monomorphism" | "definitional"}? &
                             attribute conservativity −just {omdocref}?

toplevel . attribs        &= attribute generated−via {omdocref}?
constitutive . attribs &= attribute generated−via {omdocref}?

omdoc.class &= theory−inclusion∗ & axiom−inclusion∗
theory−inclusion. justification = obligation∗ & assertion∗ & proof∗
axiom−inclusion. justification = obligation∗ & assertion∗ & proof∗

fromto.attrib = from.attrib & attribute to {omdocref}
# attributes 'to' and 'from' are URIref
```

```
morphism.attribs = id.attribs &
                                   attribute hiding {omdocrefs}? &
                                   attribute base {omdocrefs}?
morphism.model = def.eq?
morphism = element morphism {tref|(morphism.attribs & morphism.model)}
# base points to some other morphism it extends

inclusion . attribs = id. attribs & attribute via {omdocref}
inclusion . model = empty
inclusion = element inclusion { tref |( inclusion . attribs & inclusion. model)}
# via points to a theory−inclusion

theory−inclusion.attribs = toplevel. attribs & fromto.attrib
theory−inclusion.model = metadata? & morphism? & theory−inclusion.justification & CMP∗
theory−inclusion = element theory−inclusion {tref|(theory−inclusion.attribs & theory−inclusion.model)}

axiom−inclusion.attribs = toplevel. attribs & fromto.attrib
axiom−inclusion.model = metadata? & morphism? & axiom−inclusion.justification
axiom−inclusion = element theory−inclusion {tref|(axiom−inclusion.attribs & axiom−inclusion.model)}

obligation . attribs = id. attribs &
                                  attribute induced−by {omdocref} &
                                  attribute assertion {omdocref}
obligation . model = empty
obligation = element obligation { tref |( obligation . attribs & obligation. model)}
# attribute ' assertion ' is a URIref, points to an assertion
# that is the proof obligation induced by the axiom or definition
# specified by 'induced−by'.

## we allow morphisms now that we have them instead of a sequence of objects in examples.
example.model |= mc.class,morphism
```

D.11 Module RT: Rich Text Structure

The RNC module RT provides text structuring elements for mathematical text below the level of mathematical statements (see Section **??** for a discussion).

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.3) Module RT
# original in http://github.com/KWARC/OMDoc−1.3/schema/rnc
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2015 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://omdoc.org/ns"

## We extend the three main content models by xhtml elements
inline . class &= grammar {include "pxhtml.rnc"
       {Inline . model = text & parent metadata.class & Inline.class}
                                            Inline . class &= parent op.class
                                            span. attlist &= parent parallel. attribs
                                            start = Inline. class }

plike . class &= grammar {include "pxhtml.rnc"
           {Inline . model = text & parent metadata.class & Inline.class}
                        Common.attrib &= parent idrest.attribs & parent parallel.attribs
                        Inline . class &= parent op.class
                        span. attlist &= parent parallel. attribs
                        start = Block.class}

omdoc.class &= grammar {include "pxhtml.rnc"
           {Inline . model = text & parent metadata.class & Inline.class}
                        Common.attrib &= parent idrest.attribs & parent parallel.attribs
                        Inline . class &= parent op.class
```

```
                    Block.class &= parent plike.class
                    Flow.model &= parent omdoc.class
                    span. attlist &= parent parallel.attribs
                    start = List.class}
```

D.12 Module EXT: Applets and non-XML data

The RNC module EXT provides an infrastructure for applets, program code, and non-XML data like images or measurements (see Chapter 20 for a discussion).

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.3) Module EXT
# original in http://github.com/KWARC/OMDoc−1.3/schema/rnc
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2015 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://omdoc.org/ns"

plike.class &= omlet*
omdoc.class &= private* & code*

private.attribs = toplevel.attribs &
                  for.attrib? &
                  attribute requires {omdocref}? &
                  attribute reformulates {omdocref}?
private.model = metadata.class & data+
private = element private {tref|(private.attribs & private.model)}
# reformulates is a URIref to the omdoc elements that are reformulated by the
# system−specific information in this element

code.attribs = private.attribs
code.model = metadata.class & data* & input* & output* & effect*
code = element code {tref|(code.attribs & code.model)}

input.attribs = id.attribs
input.model = mcf.class
input = element input {tref|(input.attribs & input.model)}

output.attribs = id.attribs
output.model = mcf.class
output = element output {tref|(output.attribs & output.model)}

effect.attribs = id.attribs
effect.model = mcf.class
effect = element effect {tref|(effect.attribs & effect.model)}

data.attribs = id.attribs &
                 attribute href {xsd:anyURI}? &
                  attribute size {xsd:string}? &
                  attribute pto {xsd:string}? &
                  attribute pto−version {xsd:string}? &
                  attribute original {"external" | "local"}?

data.textformat = "TeX"
data.text = data.attribs & attribute format {data.textformat}? & text
data.any = data.attribs & attribute format {xsd:anyURI}? & Anything
data.model = data.text | data.any
data = element data {tref|data.model}

omlet.attribs = id.attribs &
                   attribute action    {"display" | "execute" | "other"}? &
```

```
                              attribute show      {"new" | "replace" | "embed" | "other"}? &
                              attribute actuate  {"onPresent" | "onLoad" | "onRequest" | "other"}?
omlet.param = text & inline.class & param*
omlet.data = attribute data {xsd:anyURI}|(private|code)
omlet.model = metadata.class & omlet.param & omlet.data
omlet = element omlet {tref|(omlet.attribs & omlet.model)}

param.attribs = id.attribs & name.attrib &
    attribute value      {xsd:string}? &
    attribute valuetype {"data" | "ref" | "object"}?
param.model =  mobj?
param = element param {tref|(param.attribs & param.model)}
```

D.13 Module PRES: Adding Presentation Information

The RNC module PRES provides a sub-language for defining notations for mathematical symbols and for styling OMDoc elements (see Chapter 19 for a discussion).

```
# A RelaxNG for Open Mathematical documents (OMDoc 1.3) Module PRES
# original in  http://github.com/KWARC/OMDoc−1.3/schema/rnc
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2015 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://omdoc.org/ns"
omdoc.class &= notation*

## we first add the ic and ec attributes for notation contexts everywhere
ic.attrib = attribute ic {text}
ec.attrib = attribute ec {text}
idrest.attribs &= ic.attrib? & ec.attrib?

prototype.attribs = empty
prototype.model = protoexp
prototype = element prototype {tref|(prototype.attribs & prototype.model)}

protoexp = grammar {include "openmath2.rnc"
                              {start = omel
                               common.attributes = parent id.attribs}
                          omel |= parent proto.class
                          omvar |= parent proto.class
                          common.attributes &= parent ntn.attrib}
           | grammar {include "mathml3.rnc" {start = ContExp}
                          ContExp |= parent proto.class
                          ci |= parent proto.class
                          CommonAtt &= parent ntn.attrib}

precedence.att = attribute precedence {xsd:integer} | attribute argprec {xsd:integer}
context.att = attribute xml:lang {text}? &
                           attribute context {text}? &
                           attribute variant {text}?

format.att = attribute format {text}?

rendering.attribs = precedence.att? & context.att & format.att &
                                    attribute ic {text}? & attribute ec {text}?
rendering.model = renderexp*

rendering = element rendering {tref|(rendering.attribs & rendering.model)}

renderexp = grammar {include "mathml3−common.rnc" {start = PresentationExpression}
```

```
                                        include "mathml3−presentation.rnc"
                                        PresentationExpression |= parent render.class | mtr | mtd
                                        CommonAtt &= parent ntn.attrib
                                        TableRowExpression |= parent render.class
                                        TableCellExpression |= parent render.class}
                        | (pdata|render.class)∗

pdata.attribs = empty
pdata.model = text
pdata = element pdata {pdata.attribs & pdata.model}

iterexp = grammar {include "mathml3.rnc"
          {start = PresentationExpression|mtr|mlabeledtr|mtd}
                   PresentationExpression |= parent render.class
                   MathML.Common.attrib &= parent ntn.attrib
                   TableRowExpression |= parent render.class
                   TableCellExpression |= parent render.class}

notation.attribs = id.attribs & triple.att
notation.model = metadata.class & CMP∗ & prototype+ & rendering∗
notation = element notation {tref|(notation.attribs & notation.model)}

# we extend the content and presentation models by metavariables
proto.class = exprlist | expr
render.class = render | iterate
ntn.attrib = attribute cr {text}? & attribute egroup {text}?

exprlist.attribs = name.attrib
exprlist.model = protoexp∗
exprlist = element exprlist {exprlist.attribs & exprlist.model}

expr.attribs = name.attrib
expr.model = empty
expr = element expr {tref|(expr.attribs & expr.model)}

iterate.attribs = name.attrib & precedence.att?
iterate.model = separator & iterexp∗
iterate = element iterate {tref|(iterate.attribs & iterate.model)}

render.attribs = name.attrib & precedence.att?
render.model = empty
render = element render {tref|(render.attribs & render.model)}

separator.attribs = empty
separator.model = renderexp∗
separator = element separator {tref|(separator.attribs & separator.model)}
```

D.14 Module QUIZ: Infrastructure for Assessments

The RNC module QUIZ provides a basic infrastructure for various kinds of exercises (see Chapter 21 for a discussion).

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.3) Module QUIZ
# original in http://github.com/KWARC/OMDoc−1.3/schema/rnc
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2015 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://omdoc.org/ns"

# this is really bad style, mc and hint should not be in omdoc.class
omdoc.class &= exercise∗ & hint∗ & mc∗ & solution∗
# and hint should only be plike inside an exercise
```

```
plike.class &= hint*

exercise.attribs = toplevel.attribs & for.attrib?
exercise.model = mcf.class,solution*
exercise = element exercise {tref|(exercise.attribs & exercise.model)}

omdocpf.opt.content = notAllowed

hint.attribs = toplevel.attribs & for.attrib?
hint.model = mcf.class
hint = element hint {tref|(hint.attribs & hint.model)}

solution.attribs = toplevel.attribs & for.attrib?
solution.model = mcf.class
solution = element solution {tref|(solution.attribs & solution.model)}

mc.attribs = toplevel.attribs & for.attrib?
mc.model = choice,hint?,answer
mc = element mc {tref|(mc.attribs & mc.model)}

choice.attribs = id.attribs
choice.model = mcf.class
choice = element choice {tref|(choice.attribs & choice.model)}

answer.attribs = id.attribs & attribute verdict {"true" | "false"}?
answer.model = mcf.class
answer = element answer {tref|(answer.attribs & answer.model)}
```

E

The RelaxNG Schemata for Mathematical Objects

For completeness we reprint the RelaxNG schemata for the external formats OMDoc makes use of.

E.1 The RelaxNG Schema for OpenMath

For completeness we reprint the RelaxNG schema for OpenMath, the original can be found in the OpenMath2 standard [Bus+04].

```
# RELAX NG Schema for OpenMath 2
# original in  http://github.com/KWARC/OMDoc-1.3/schema/rnc
# See the documentation and examples at http://www.openmath.org

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

start = OMOBJ

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

# Elements which can appear inside an OpenMath object
omel =
  OMS | OMV | OMI | OMB | OMSTR | OMF | OMA | OMBIND | OME | OMATTR |OMR

# things which can be variables
omvar = OMV | attvar

attvar = element OMATTR { common.attributes,(OMATP , (OMV | attvar))}


cdbase = attribute cdbase { xsd:anyURI}?

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

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

# symbol
OMS = element OMS { common.attributes,
```

```
                    attribute name {xsd:NCName},
                    attribute cd {xsd:NCName},
                    cdbase }

# variable
OMV = element OMV { common.attributes,
                    attribute name { xsd:NCName} }

# integer
OMI = element OMI { common.attributes,
                    xsd:string {pattern = "\s*(-\s?)?[0-9]+(\s[0-9]+)*\s*"}}
# byte array
OMB = element OMB { common.attributes, xsd:base64Binary }

# string
OMSTR = element OMSTR { common.attributes, text }

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

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

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

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

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

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

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

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

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

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

E.2 The RelaxNG Schema for MathML

For completeness, we reprint the RelaxNG schema for MathML. It comes in three parts, the schema driver, and the parts for content- and presentation MathML which we will present in the next two subsections.

```
#     This is the Mathematical Markup Language (MathML) 3.0, an XML
#     application for describing mathematical notation and capturing
#     both its structure and content.
#
#     Copyright 1998−2009 W3C (MIT, ERCIM, Keio)
#
#     Use and distribution of this code are permitted under the terms
#     W3C Software Notice and License
#     http://www.w3.org/Consortium/Legal/2002/copyright−software−20021231


default namespace m = ”http://www.w3.org/1998/Math/MathML”

## Content MathML
include ”mathml3−content.rnc”

## Presentation MathML
include ”mathml3−presentation.rnc”

## math and semantics common to both Content and Presentation
include ”mathml3−common.rnc”
```

E.2.1 Presentation MathML

```
#     This is the Mathematical Markup Language (MathML) 3.0, an XML
#     application for describing mathematical notation and capturing
#     both its structure and content.
#
#     Copyright 1998−2009 W3C (MIT, ERCIM, Keio)
#
#     Use and distribution of this code are permitted under the terms
#     W3C Software Notice and License
#     http://www.w3.org/Consortium/Legal/2002/copyright−software−20021231


default namespace m = ”http://www.w3.org/1998/Math/MathML”
namespace local = ””

start = math

math = element math {math.attributes,MathExpression∗}
MathExpression = semantics

NonMathMLAtt = attribute (∗ − (local:∗|m:∗)) {xsd:string}

CommonDeprecatedAtt = attribute other {text}?

CommonAtt = attribute id {xsd:ID}?,
            attribute xref {text}?,
            attribute class {xsd:NMTOKENS}?,
            attribute style {xsd:string}?,
            attribute href {xsd:anyURI}?,
            CommonDeprecatedAtt,
            NonMathMLAtt∗


math.attributes = CommonAtt,
               attribute display {”block” | ”inline”}?,
               attribute maxwidth {length}?,
               attribute overflow {”linebreak” | ”scroll” | ”elide” | ”truncate” | ”scale”}?,
               attribute altimg {xsd:anyURI}?,
               attribute altimg−width {length}?,
               attribute altimg−height {length}?,
```

```
               attribute altimg−valign {length | ”top” | ”middle” | ”bottom”}?,
               attribute alttext {text}?,
               attribute cdgroup {xsd:anyURI}?,
               math.deprecatedattributes

# the mathml3−presentation schema adds additional attributes
# to the math element, all those valid on mstyle

math.deprecatedattributes = attribute mode {xsd:string}?,
                            attribute macros {xsd:string}?


name = attribute name {xsd:NCName}
cd = attribute cd {xsd:NCName}

src = attribute src {xsd:anyURI}?

annotation = element annotation {annotation.attributes,text}

annotation−xml.model = (MathExpression|anyElement)∗

anyElement = element (∗ − m:∗) {(attribute ∗ {text}|text| anyElement)∗}

annotation−xml = element annotation−xml {annotation.attributes,
                                         annotation−xml.model}
annotation.attributes = CommonAtt,
                        cd?,
                        name?,
                        DefEncAtt,
                        src?

DefEncAtt = attribute encoding {xsd:string}?,
            attribute definitionURL {xsd:anyURI}?

semantics = element semantics {semantics.attributes,
                               MathExpression,
                              (annotation|annotation−xml)∗}
semantics.attributes = CommonAtt,DefEncAtt,cd?,name?


length = xsd:string {
  pattern = '\s∗((−?[0−9]∗(\.[0−9]∗)?(e[mx]|in|cm|mm|p[xtc]|%)?)|(negative)?((very){0,2}thi(n|ck)|medium)mathspace)\s∗'
}
```

E.2.2 Presentation MathML

```
#     This is the Mathematical Markup Language (MathML) 3.0, an XML
#     application for describing mathematical notation and capturing
#     both its structure and content.
#
#     Copyright 1998−2010 W3C (MIT, ERCIM, Keio)
#
#     Use and distribution of this code are permitted under the terms
#     W3C Software Notice and License
#     http://www.w3.org/Consortium/Legal/2002/copyright−software−20021231

default namespace m = ”http://www.w3.org/1998/Math/MathML”

MathExpression |= PresentationExpression

ImpliedMrow = MathExpression∗

```

```
TableRowExpression = mtr|mlabeledtr

TableCellExpression = mtd

MstackExpression = MathExpression|mscarries|msline|msrow|msgroup

MsrowExpression = MathExpression|none

MultiScriptExpression = (MathExpression|none),(MathExpression|none)

mpadded-length = xsd:string {
  pattern = '\s*([\+\-]?[0-9]*(\.[0-9]*)?\s*((%?\s*(height|depth|width)?)|e[mx]|in|cm|mm|p[xtc]|((negative)?((very){0,2}thi(n|ck)|me

 linestyle = "none" | "solid" | "dashed"

 verticalalign =
      "top" |
      "bottom" |
      "center" |
      "baseline" |
      "axis"

columnalignstyle = "left" | "center" | "right"

notationstyle =
     "longdiv" |
     "actuarial" |
     "radical" |
     "box" |
     "roundedbox" |
     "circle" |
     "left" |
     "right" |
     "top" |
     "bottom" |
     "updiagonalstrike" |
     "downdiagonalstrike" |
     "verticalstrike" |
     "horizontalstrike" |
     "madruwb"

idref = text
unsigned-integer = xsd:unsignedLong
integer = xsd:integer
number = xsd:decimal

character = xsd:string {
  pattern = '\s*\S\s*'}

color = xsd:string {
  pattern = '\s*((#[0-9a-fA-F]{3}([0-9a-fA-F]{3})?)|[aA][qQ][uU][aA]|[bB][lL][aA][cC][kK]|[bB][lL][uU][eE]|[fF][uU][cC][hH][sS][iI][a


group-alignment = "left" | "center" | "right" | "decimalpoint"
group-alignment-list = list {group-alignment+}
group-alignment-list-list = xsd:string {
  pattern = '(\s*\{\s*(left|center|right|decimalpoint)(\s+(left|center|right|decimalpoint))*\})*\s*' }
positive-integer = xsd:positiveInteger


TokenExpression = mi|mn|mo|mtext|mspace|ms

token.content = mglyph|malignmark|text

mi = element mi {mi.attributes, token.content*}
mi.attributes =
  CommonAtt,
```

```
  CommonPresAtt,
  TokenAtt


mn = element mn {mn.attributes, token.content*}
mn.attributes =
  CommonAtt,
  CommonPresAtt,
  TokenAtt


mo = element mo {mo.attributes, token.content*}
mo.attributes =
  CommonAtt,
  CommonPresAtt,
  TokenAtt,
  attribute form {”prefix” | ”infix” | ”postfix”}?,
  attribute fence {”true” | ”false”}?,
  attribute separator {”true” | ”false”}?,
  attribute lspace {length}?,
  attribute rspace {length}?,
  attribute stretchy {”true” | ”false”}?,
  attribute symmetric {”true” | ”false”}?,
  attribute maxsize {length | ”infinity”}?,
  attribute minsize {length}?,
  attribute largeop {”true” | ”false”}?,
  attribute movablelimits {”true” | ”false”}?,
  attribute accent {”true” | ”false”}?,
  attribute linebreak {”auto” | ”newline” | ”nobreak” | ”goodbreak” | ”badbreak”}?,
  attribute lineleading {length}?,
  attribute linebreakstyle {”before” | ”after” | ”duplicate” | ”infixlinebreakstyle”}?,
  attribute linebreakmultchar {text}?,
  attribute indentalign {”left” | ”center” | ”right” | ”auto” | ”id”}?,
  attribute indentshift {length}?,
  attribute indenttarget {idref}?,
  attribute indentalignfirst {”left” | ”center” | ”right” | ”auto” | ”id” | ”indentalign”}?,
  attribute indentshiftfirst {length | ”indentshift”}?,
  attribute indentalignlast {”left” | ”center” | ”right” | ”auto” | ”id” | ”indentalign”}?,
  attribute indentshiftlast {length | ”indentshift”}?


mtext = element mtext {mtext.attributes, token.content*}
mtext.attributes =
  CommonAtt,
  CommonPresAtt,
  TokenAtt


mspace = element mspace {mspace.attributes, empty}
mspace.attributes =
  CommonAtt,
  CommonPresAtt,
  TokenAtt,
  attribute width {length}?,
  attribute height {length}?,
  attribute depth {length}?,
  attribute linebreak {”auto” | ”newline” | ”nobreak” | ”goodbreak” | ”badbreak” | ”indentingnewline”}?


ms = element ms {ms.attributes, token.content*}
ms.attributes =
  CommonAtt,
  CommonPresAtt,
  TokenAtt,
  attribute lquote {text}?,
  attribute rquote {text}?
```

```
mglyph = element mglyph {mglyph.attributes,mglyph.deprecatedattributes,empty}
mglyph.attributes =
  CommonAtt, CommonPresAtt,
  attribute src {xsd:anyURI}?,
  attribute width {length}?,
  attribute height {length}?,
  attribute valign {length}?,
  attribute alt {text}?
mglyph.deprecatedattributes =
  attribute index {integer}?,
  attribute mathvariant {"normal" | "bold" | "italic" | "bold-italic" | "double-struck" | "bold-fraktur" | "script" | "bold-script" |
  attribute mathsize {"small" | "normal" | "big" | length}?,
  DeprecatedTokenAtt

msline = element msline {msline.attributes,empty}
msline.attributes =
  CommonAtt, CommonPresAtt,
  attribute position {integer}?,
  attribute length {unsigned-integer}?,
  attribute leftoverhang {length}?,
  attribute rightoverhang {length}?,
  attribute mslinethickness {length | "thin" | "medium" | "thick"}?

none = element none {none.attributes,empty}
none.attributes =
  CommonAtt,
  CommonPresAtt

mprescripts = element mprescripts {mprescripts.attributes,empty}
mprescripts.attributes =
  CommonAtt,
  CommonPresAtt


CommonPresAtt =
  attribute mathcolor {color}?,
  attribute mathbackground {color | "transparent"}?

TokenAtt =
  attribute mathvariant {"normal" | "bold" | "italic" | "bold-italic" | "double-struck" | "bold-fraktur" | "script" | "bold-script" |
  attribute mathsize {"small" | "normal" | "big" | length}?,
  attribute dir {"ltr" | "rtl"}?,
  DeprecatedTokenAtt

DeprecatedTokenAtt =
  attribute fontfamily {text}?,
  attribute fontweight {"normal" | "bold"}?,
  attribute fontstyle {"normal" | "italic"}?,
  attribute fontsize {length}?,
  attribute color {color}?,
  attribute background {color | "transparent"}?

MalignExpression = maligngroup|malignmark

malignmark = element malignmark {malignmark.attributes, empty}
malignmark.attributes =
  CommonAtt, CommonPresAtt,
  attribute edge {"left" | "right"}?


maligngroup = element maligngroup {maligngroup.attributes, empty}
maligngroup.attributes =
  CommonAtt, CommonPresAtt,
  attribute groupalign {"left" | "center" | "right" | "decimalpoint"}?
```

```
PresentationExpression = TokenExpression|MalignExpression|
                         mrow|mfrac|msqrt|mroot|mstyle|merror|mpadded|mphantom|
                         mfenced|menclose|msub|msup|msubsup|munder|mover|munderover|
                         mmultiscripts|mtable|mstack|mlongdiv|maction



mrow = element mrow {mrow.attributes, MathExpression*}
mrow.attributes =
  CommonAtt, CommonPresAtt,
  attribute dir {"ltr" | "rtl"}?


mfrac = element mfrac {mfrac.attributes, MathExpression, MathExpression}
mfrac.attributes =
  CommonAtt, CommonPresAtt,
  attribute linethickness {length | "thin" | "medium" | "thick"}?,
  attribute numalign {"left" | "center" | "right"}?,
  attribute denomalign {"left" | "center" | "right"}?,
  attribute bevelled {"true" | "false"}?


msqrt = element msqrt {msqrt.attributes, ImpliedMrow}
msqrt.attributes =
  CommonAtt, CommonPresAtt


mroot = element mroot {mroot.attributes, MathExpression, MathExpression}
mroot.attributes =
  CommonAtt, CommonPresAtt


mstyle = element mstyle {mstyle.attributes, ImpliedMrow}
mstyle.attributes =
  CommonAtt, CommonPresAtt,
  mstyle.specificattributes,
  mstyle.generalattributes,
  mstyle.deprecatedattributes

mstyle.specificattributes =
  attribute scriptlevel {integer}?,
  attribute displaystyle {"true" | "false"}?,
  attribute scriptsizemultiplier {number}?,
  attribute scriptminsize {length}?,
  attribute infixlinebreakstyle {"before" | "after" | "duplicate"}?,
  attribute decimalpoint {character}?

mstyle.generalattributes =
  attribute accent {"true" | "false"}?,
  attribute accentunder {"true" | "false"}?,
  attribute align {"left" | "right" | "center"}?,
  attribute alignmentscope {list {("true" | "false") +}}?,
  attribute bevelled {"true" | "false"}?,
  attribute charalign {"left" | "center" | "right"}?,
  attribute charspacing {length | "loose" | "medium" | "tight"}?,
  attribute close {text}?,
  attribute columnalign {list {columnalignstyle+} }?,
  attribute columnlines {list {linestyle +}}?,
  attribute columnspacing {list {(length) +}}?,
  attribute columnspan {positive-integer}?,
  attribute columnwidth {list {("auto" | length | "fit") +}}?,
  attribute crossout {list {("none" | "updiagonalstrike" | "downdiagonalstrike" | "verticalstrike" | "horizontalstrike")*}}?,
  attribute denomalign {"left" | "center" | "right"}?,
  attribute depth {length}?,
  attribute dir {"ltr" | "rtl"}?,
  attribute edge {"left" | "right"}?,
  attribute equalcolumns {"true" | "false"}?,
```

```
  attribute equalrows {"true" | "false"}?,
  attribute fence {"true" | "false"}?,
  attribute form {"prefix" | "infix" | "postfix"}?,
  attribute frame {linestyle}?,
  attribute framespacing {list {length, length}}?,
  attribute groupalign {group-alignment-list-list}?,
  attribute height {length}?,
  attribute indentalign {"left" | "center" | "right" | "auto" | "id"}?,
  attribute indentalignfirst {"left" | "center" | "right" | "auto" | "id" | "indentalign"}?,
  attribute indentalignlast {"left" | "center" | "right" | "auto" | "id" | "indentalign"}?,
  attribute indentshift {length}?,
  attribute indentshiftfirst {length | "indentshift"}?,
  attribute indentshiftlast {length | "indentshift"}?,
  attribute indenttarget {idref}?,
  attribute largeop {"true" | "false"}?,
  attribute leftoverhang {length}?,
  attribute length {unsigned-integer}?,
  attribute linebreak {"auto" | "newline" | "nobreak" | "goodbreak" | "badbreak"}?,
  attribute linebreakmultchar {text}?,
  attribute linebreakstyle {"before" | "after" | "duplicate" | "infixlinebreakstyle"}?,
  attribute lineleading {length}?,
  attribute linethickness {length | "thin" | "medium" | "thick"}?,
  attribute location {"w" | "nw" | "n" | "ne" | "e" | "se" | "s" | "sw"}?,
  attribute longdivstyle {"lefttop" | "stackedrightright" | "mediumstackedrightright" | "shortstackedrightright" | "righttop" | "left/
  attribute lquote {text}?,
  attribute lspace {length}?,
  attribute mathsize {"small" | "normal" | "big" | length}?,
  attribute mathvariant {"normal" | "bold" | "italic" | "bold-italic" | "double-struck" | "bold-fraktur" | "script" | "bold-script" |
  attribute maxsize {length | "infinity"}?,
  attribute minlabelspacing {length}?,
  attribute minsize {length}?,
  attribute movablelimits {"true" | "false"}?,
  attribute mslinethickness {length | "thin" | "medium" | "thick"}?,
  attribute notation {text}?,
  attribute numalign {"left" | "center" | "right"}?,
  attribute open {text}?,
  attribute position {integer}?,
  attribute rightoverhang {length}?,
  attribute rowalign {list {verticalalign+} }?,
  attribute rowlines {list {linestyle +}}?,
  attribute rowspacing {list {(length) +}}?,
  attribute rowspan {positive-integer}?,
  attribute rquote {text}?,
  attribute rspace {length}?,
  attribute selection {positive-integer}?,
  attribute separator {"true" | "false"}?,
  attribute separators {text}?,
  attribute shift {integer}?,
  attribute side {"left" | "right" | "leftoverlap" | "rightoverlap"}?,
  attribute stackalign {"left" | "center" | "right" | "decimalpoint"}?,
  attribute stretchy {"true" | "false"}?,
  attribute subscriptshift {length}?,
  attribute superscriptshift {length}?,
  attribute symmetric {"true" | "false"}?,
  attribute valign {length}?,
  attribute width {length}?

mstyle.deprecatedattributes =
  DeprecatedTokenAtt,
  attribute veryverythinmathspace {length}?,
  attribute verythinmathspace {length}?,
  attribute thinmathspace {length}?,
  attribute mediummathspace {length}?,
  attribute thickmathspace {length}?,
  attribute verythickmathspace {length}?,
  attribute veryverythickmathspace {length}?

```

```
math.attributes &= CommonPresAtt
math.attributes &= mstyle.specificattributes
math.attributes &= mstyle.generalattributes



merror = element merror {merror.attributes, ImpliedMrow}
merror.attributes =
  CommonAtt, CommonPresAtt


mpadded = element mpadded {mpadded.attributes, ImpliedMrow}
mpadded.attributes =
  CommonAtt, CommonPresAtt,
  attribute height {mpadded-length}?,
  attribute depth {mpadded-length}?,
  attribute width {mpadded-length}?,
  attribute lspace {mpadded-length}?,
  attribute voffset {mpadded-length}?


mphantom = element mphantom {mphantom.attributes, ImpliedMrow}
mphantom.attributes =
  CommonAtt, CommonPresAtt


mfenced = element mfenced {mfenced.attributes, MathExpression*}
mfenced.attributes =
  CommonAtt, CommonPresAtt,
  attribute open {text}?,
  attribute close {text}?,
  attribute separators {text}?


menclose = element menclose {menclose.attributes, ImpliedMrow}
menclose.attributes =
  CommonAtt, CommonPresAtt,
  attribute notation {text}?


msub = element msub {msub.attributes, MathExpression, MathExpression}
msub.attributes =
  CommonAtt, CommonPresAtt,
  attribute subscriptshift {length}?


msup = element msup {msup.attributes, MathExpression, MathExpression}
msup.attributes =
  CommonAtt, CommonPresAtt,
  attribute superscriptshift {length}?


msubsup = element msubsup {msubsup.attributes, MathExpression, MathExpression, MathExpression}
msubsup.attributes =
  CommonAtt, CommonPresAtt,
  attribute subscriptshift {length}?,
  attribute superscriptshift {length}?


munder = element munder {munder.attributes, MathExpression, MathExpression}
munder.attributes =
  CommonAtt, CommonPresAtt,
  attribute accentunder {"true" | "false"}?,
  attribute align {"left" | "right" | "center"}?
```

```
mover = element mover {mover.attributes, MathExpression, MathExpression}
mover.attributes =
  CommonAtt, CommonPresAtt,
  attribute accent {"true" | "false"}?,
  attribute align {"left" | "right" | "center"}?


munderover = element munderover {munderover.attributes, MathExpression, MathExpression, MathExpression}
munderover.attributes =
  CommonAtt, CommonPresAtt,
  attribute accent {"true" | "false"}?,
  attribute accentunder {"true" | "false"}?,
  attribute align {"left" | "right" | "center"}?


mmultiscripts = element mmultiscripts {mmultiscripts.attributes, MathExpression,MultiScriptExpression*,(mprescripts,MultiScriptExpr
mmultiscripts.attributes =
  msubsup.attributes


mtable = element mtable {mtable.attributes, TableRowExpression*}
mtable.attributes =
  CommonAtt, CommonPresAtt,
  attribute align {xsd:string {
    pattern ='\s*(top|bottom|center|baseline|axis)\s*[0-9]*'}}?,
  attribute rowalign {list {verticalalign+} }?,
  attribute columnalign {list {columnalignstyle+} }?,
  attribute groupalign {group-alignment-list-list}?,
  attribute alignmentscope {list {("true" | "false") +}}?,
  attribute columnwidth {list {("auto" | length | "fit") +}}?,
  attribute width {"auto" | length}?,
  attribute rowspacing {list {(length) +}}?,
  attribute columnspacing {list {(length) +}}?,
  attribute rowlines {list {linestyle +}}?,
  attribute columnlines {list {linestyle +}}?,
  attribute frame {linestyle}?,
  attribute framespacing {list {length, length}}?,
  attribute equalrows {"true" | "false"}?,
  attribute equalcolumns {"true" | "false"}?,
  attribute displaystyle {"true" | "false"}?,
  attribute side {"left" | "right" | "leftoverlap" | "rightoverlap"}?,
  attribute minlabelspacing {length}?


mlabeledtr = element mlabeledtr {mlabeledtr.attributes, TableCellExpression+}
mlabeledtr.attributes =
  mtr.attributes


mtr = element mtr {mtr.attributes, TableCellExpression*}
mtr.attributes =
  CommonAtt, CommonPresAtt,
  attribute rowalign {"top" | "bottom" | "center" | "baseline" | "axis"}?,
  attribute columnalign {list {columnalignstyle+} }?,
  attribute groupalign {group-alignment-list-list}?


mtd = element mtd {mtd.attributes, ImpliedMrow}
mtd.attributes =
  CommonAtt, CommonPresAtt,
  attribute rowspan {positive-integer}?,
  attribute columnspan {positive-integer}?,
  attribute rowalign {"top" | "bottom" | "center" | "baseline" | "axis"}?,
  attribute columnalign {columnalignstyle}?,
  attribute groupalign {group-alignment-list}?
```

```
mstack = element mstack {mstack.attributes, MstackExpression*}
mstack.attributes =
  CommonAtt, CommonPresAtt,
  attribute align {xsd:string {
    pattern ='\s*(top|bottom|center|baseline|axis)\s*[0-9]*'}}?,
  attribute stackalign {"left" | "center" | "right" | "decimalpoint"}?,
  attribute charalign {"left" | "center" | "right"}?,
  attribute charspacing {length | "loose" | "medium" | "tight"}?


mlongdiv = element mlongdiv {mlongdiv.attributes, MstackExpression,MstackExpression,MstackExpression+}
mlongdiv.attributes =
  msgroup.attributes,
  attribute longdivstyle {"lefttop" | "stackedrightright" | "mediumstackedrightright" | "shortstackedrightright" | "righttop" | " left /

msgroup = element msgroup {msgroup.attributes, MstackExpression*}
msgroup.attributes =
  CommonAtt, CommonPresAtt,
  attribute position {integer}?,
  attribute shift {integer}?


msrow = element msrow {msrow.attributes, MsrowExpression*}
msrow.attributes =
  CommonAtt, CommonPresAtt,
  attribute position {integer}?


mscarries = element mscarries {mscarries.attributes, (MsrowExpression|mscarry)*}
mscarries. attributes =
  CommonAtt, CommonPresAtt,
  attribute position {integer}?,
  attribute location {"w" | "nw" | "n" | "ne" | "e" | "se" | "s" | "sw"}?,
  attribute crossout { list {("none" | "updiagonalstrike" | "downdiagonalstrike" | " verticalstrike " | " horizontalstrike ")*}}?,
  attribute scriptsizemultiplier {number}?


mscarry = element mscarry {mscarry.attributes, MsrowExpression*}
mscarry.attributes =
  CommonAtt, CommonPresAtt,
  attribute location {"w" | "nw" | "n" | "ne" | "e" | "se" | "s" | "sw"}?,
  attribute crossout { list {("none" | "updiagonalstrike" | "downdiagonalstrike" | " verticalstrike " | " horizontalstrike ")*}}?


maction = element maction {maction.attributes, MathExpression+}
maction.attributes =
  CommonAtt, CommonPresAtt,
  attribute actiontype {text}?,
  attribute selection {positive-integer}?
```

E.2.3 Strict Content MathML

```
#     This is the Mathematical Markup Language (MathML) 3.0, an XML
#     application for describing mathematical notation and capturing
#     both its structure and content.
#
#     Copyright 1998-2009 W3C (MIT, ERCIM, Keio)
#
#     Use and distribution of this code are permitted under the terms
#     W3C Software Notice and License
#     http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231

```

```
default namespace m = "http://www.w3.org/1998/Math/MathML"

ContExp = semantics-contexp | cn | ci | csymbol | apply | bind | share | cerror | cbytes | cs

cn = element cn {cn.attributes,cn.content}
cn.content = text
cn.attributes = attribute type {"integer" | "real" | "double" | "hexdouble"}

semantics-ci = element semantics {semantics.attributes,(ci|semantics-ci),
  (annotation|annotation-xml)*}

semantics-contexp = element semantics {semantics.attributes,ContExp,
  (annotation|annotation-xml)*}

ci = element ci {ci.attributes, ci.content}
ci.attributes = CommonAtt, ci.type?
ci.type = attribute type {"integer" | "rational" | "real" | "complex" | "complex-polar" | "complex-cartesian" | "constant" | "functi
ci.content = text


csymbol = element csymbol {csymbol.attributes,csymbol.content}

SymbolName = xsd:NCName
csymbol.attributes = CommonAtt, cd
csymbol.content = SymbolName

BvarQ = bvar*
bvar = element bvar { ci | semantics-ci}

apply = element apply {CommonAtt,apply.content}
apply.content = ContExp+


bind = element bind {CommonAtt,bind.content}
bind.content = ContExp,bvar*,ContExp

share = element share {CommonAtt, src, empty}

cerror = element cerror {cerror.attributes, csymbol, ContExp*}
cerror.attributes = CommonAtt

cbytes = element cbytes {cbytes.attributes, base64}
cbytes.attributes = CommonAtt
base64 = xsd:base64Binary

cs = element cs {cs.attributes, text}
cs.attributes = CommonAtt

MathExpression |= ContExp
```

E.2.4 Content MathML

```
#       This is the Mathematical Markup Language (MathML) 3.0, an XML
#       application for describing mathematical notation and capturing
#       both its structure and content.
#
#       Copyright 1998-2010 W3C (MIT, ERCIM, Keio)
#
#       Use and distribution of this code are permitted under the terms
#       W3C Software Notice and License
#       http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231

```

```
include "mathml3-strict-content.rnc"{
  cn.content = (text | mglyph | sep | PresentationExpression)*
  cn.attributes = CommonAtt, DefEncAtt, attribute type {text}?, base?

  ci.attributes = CommonAtt, DefEncAtt, ci.type?
  ci.type = attribute type {text}
  ci.content = (text | mglyph | PresentationExpression)*

  csymbol.attributes = CommonAtt, DefEncAtt, attribute type {text}?,cd?
  csymbol.content = (text | mglyph | PresentationExpression)*

  bvar = element bvar { (ci | semantics-ci) & degree?}

  cbytes.attributes = CommonAtt, DefEncAtt

  cs.attributes = CommonAtt, DefEncAtt

  apply.content = ContExp+ | (ContExp, BvarQ, Qualifier*, ContExp*)

  bind.content = apply.content
}

base = attribute base {text}


sep = element sep {empty}
PresentationExpression |= notAllowed


DomainQ = (domainofapplication|condition|interval|(lowlimit,uplimit?))*
domainofapplication = element domainofapplication {ContExp}
condition = element condition {ContExp}
uplimit = element uplimit {ContExp}
lowlimit = element lowlimit {ContExp}

Qualifier = DomainQ|degree|momentabout|logbase
degree = element degree {ContExp}
momentabout = element momentabout {ContExp}
logbase = element logbase {ContExp}

type = attribute type {text}
order = attribute order {"numeric" | "lexicographic"}
closure = attribute closure {text}


ContExp |= piecewise


piecewise = element piecewise {CommonAtt, DefEncAtt,(piece* & otherwise?)}

piece = element piece {CommonAtt, DefEncAtt, ContExp, ContExp}

otherwise = element otherwise {CommonAtt, DefEncAtt, ContExp}


DeprecatedContExp = reln | fn | declare
ContExp |= DeprecatedContExp

reln = element reln {ContExp*}
fn = element fn {ContExp}
declare = element declare {attribute type {xsd:string}?,
                           attribute scope {xsd:string}?,
                           attribute nargs {xsd:nonNegativeInteger}?,
                           attribute occurrence {"prefix"|"infix"|"function-model"}?,
                           DefEncAtt,
                           ContExp+}
```

```
interval.class = interval
ContExp |= interval.class


interval = element interval { CommonAtt, DefEncAtt,closure?, ContExp,ContExp}

unary-functional.class = inverse | ident | domain | codomain | image | ln | log | moment
ContExp |= unary-functional.class


inverse = element inverse { CommonAtt, DefEncAtt, empty}
ident = element ident { CommonAtt, DefEncAtt, empty}
domain = element domain { CommonAtt, DefEncAtt, empty}
codomain = element codomain { CommonAtt, DefEncAtt, empty}
image = element image { CommonAtt, DefEncAtt, empty}
ln = element ln { CommonAtt, DefEncAtt, empty}
log = element log { CommonAtt, DefEncAtt, empty}
moment = element moment { CommonAtt, DefEncAtt, empty}

lambda.class = lambda
ContExp |= lambda.class


lambda = element lambda { CommonAtt, DefEncAtt, BvarQ, DomainQ, ContExp}

nary-functional.class = compose
ContExp |= nary-functional.class


compose = element compose { CommonAtt, DefEncAtt, empty}

binary-arith.class = quotient | divide | minus | power | rem | root
ContExp |= binary-arith.class


quotient = element quotient { CommonAtt, DefEncAtt, empty}
divide = element divide { CommonAtt, DefEncAtt, empty}
minus = element minus { CommonAtt, DefEncAtt, empty}
power = element power { CommonAtt, DefEncAtt, empty}
rem = element rem { CommonAtt, DefEncAtt, empty}
root = element root { CommonAtt, DefEncAtt, empty}

unary-arith.class = factorial | minus | root | abs | conjugate | arg | real | imaginary | floor | ceiling | exp
ContExp |= unary-arith.class


factorial = element factorial { CommonAtt, DefEncAtt, empty}
abs = element abs { CommonAtt, DefEncAtt, empty}
conjugate = element conjugate { CommonAtt, DefEncAtt, empty}
arg = element arg { CommonAtt, DefEncAtt, empty}
real = element real { CommonAtt, DefEncAtt, empty}
imaginary = element imaginary { CommonAtt, DefEncAtt, empty}
floor = element floor { CommonAtt, DefEncAtt, empty}
ceiling = element ceiling { CommonAtt, DefEncAtt, empty}
exp = element exp { CommonAtt, DefEncAtt, empty}

nary-minmax.class = max | min
ContExp |= nary-minmax.class


max = element max { CommonAtt, DefEncAtt, empty}
min = element min { CommonAtt, DefEncAtt, empty}

nary-arith.class = plus | times | gcd | lcm
ContExp |= nary-arith.class
```

```
plus = element plus { CommonAtt, DefEncAtt, empty}
times = element times { CommonAtt, DefEncAtt, empty}
gcd = element gcd { CommonAtt, DefEncAtt, empty}
lcm = element lcm { CommonAtt, DefEncAtt, empty}

nary-logical.class = and | or | xor
ContExp |= nary-logical.class


and = element and { CommonAtt, DefEncAtt, empty}
or = element or { CommonAtt, DefEncAtt, empty}
xor = element xor { CommonAtt, DefEncAtt, empty}

unary-logical.class = not
ContExp |= unary-logical.class


not = element not { CommonAtt, DefEncAtt, empty}

binary-logical.class = implies | equivalent
ContExp |= binary-logical.class


implies = element implies { CommonAtt, DefEncAtt, empty}
equivalent = element equivalent { CommonAtt, DefEncAtt, empty}

quantifier.class = forall | exists
ContExp |= quantifier.class


forall = element forall { CommonAtt, DefEncAtt, empty}
exists = element exists { CommonAtt, DefEncAtt, empty}

nary-reln.class = eq | gt | lt | geq | leq
ContExp |= nary-reln.class


eq = element eq { CommonAtt, DefEncAtt, empty}
gt = element gt { CommonAtt, DefEncAtt, empty}
lt = element lt { CommonAtt, DefEncAtt, empty}
geq = element geq { CommonAtt, DefEncAtt, empty}
leq = element leq { CommonAtt, DefEncAtt, empty}

binary-reln.class = neq | approx | factorof | tendsto
ContExp |= binary-reln.class


neq = element neq { CommonAtt, DefEncAtt, empty}
approx = element approx { CommonAtt, DefEncAtt, empty}
factorof = element factorof { CommonAtt, DefEncAtt, empty}
tendsto = element tendsto { CommonAtt, DefEncAtt, type?, empty}

int.class = int
ContExp |= int.class


int = element int { CommonAtt, DefEncAtt, empty}

Differential-Operator.class = diff
ContExp |= Differential-Operator.class


diff = element diff { CommonAtt, DefEncAtt, empty}

partialdiff.class = partialdiff
ContExp |= partialdiff.class
```

```
partialdiff = element partialdiff { CommonAtt, DefEncAtt, empty}

unary-veccalc.class = divergence | grad | curl | laplacian
ContExp |= unary-veccalc.class


divergence = element divergence { CommonAtt, DefEncAtt, empty}
grad = element grad { CommonAtt, DefEncAtt, empty}
curl = element curl { CommonAtt, DefEncAtt, empty}
laplacian = element laplacian { CommonAtt, DefEncAtt, empty}

nary-setlist-constructor.class = set | \list
ContExp |= nary-setlist-constructor.class


set = element set { CommonAtt, DefEncAtt, type?, BvarQ*, DomainQ*, ContExp*}
\list = element \list { CommonAtt, DefEncAtt, order?, BvarQ*, DomainQ*, ContExp*}

nary-set.class = union | intersect | cartesianproduct
ContExp |= nary-set.class


union = element union { CommonAtt, DefEncAtt, empty}
intersect = element intersect { CommonAtt, DefEncAtt, empty}
cartesianproduct = element cartesianproduct { CommonAtt, DefEncAtt, empty}

binary-set.class = in | notin | notsubset | notprsubset | setdiff
ContExp |= binary-set.class


in = element in { CommonAtt, DefEncAtt, empty}
notin = element notin { CommonAtt, DefEncAtt, empty}
notsubset = element notsubset { CommonAtt, DefEncAtt, empty}
notprsubset = element notprsubset { CommonAtt, DefEncAtt, empty}
setdiff = element setdiff { CommonAtt, DefEncAtt, empty}

nary-set-reln.class = subset | prsubset
ContExp |= nary-set-reln.class


subset = element subset { CommonAtt, DefEncAtt, empty}
prsubset = element prsubset { CommonAtt, DefEncAtt, empty}

unary-set.class = card
ContExp |= unary-set.class


card = element card { CommonAtt, DefEncAtt, empty}

sum.class = sum
ContExp |= sum.class


sum = element sum { CommonAtt, DefEncAtt, empty}

product.class = product
ContExp |= product.class


product = element product { CommonAtt, DefEncAtt, empty}

limit.class = limit
ContExp |= limit.class
```

```
limit = element limit { CommonAtt, DefEncAtt, empty}

unary−elementary.class = sin | cos | tan | sec | csc | cot | sinh | cosh | tanh | sech | csch | coth | arcsin | arccos | arctan | arcc
ContExp |= unary−elementary.class


sin = element sin { CommonAtt, DefEncAtt, empty}
cos = element cos { CommonAtt, DefEncAtt, empty}
tan = element tan { CommonAtt, DefEncAtt, empty}
sec = element sec { CommonAtt, DefEncAtt, empty}
csc = element csc { CommonAtt, DefEncAtt, empty}
cot = element cot { CommonAtt, DefEncAtt, empty}
sinh = element sinh { CommonAtt, DefEncAtt, empty}
cosh = element cosh { CommonAtt, DefEncAtt, empty}
tanh = element tanh { CommonAtt, DefEncAtt, empty}
sech = element sech { CommonAtt, DefEncAtt, empty}
csch = element csch { CommonAtt, DefEncAtt, empty}
coth = element coth { CommonAtt, DefEncAtt, empty}
arcsin = element arcsin { CommonAtt, DefEncAtt, empty}
arccos = element arccos { CommonAtt, DefEncAtt, empty}
arctan = element arctan { CommonAtt, DefEncAtt, empty}
arccosh = element arccosh { CommonAtt, DefEncAtt, empty}
arccot = element arccot { CommonAtt, DefEncAtt, empty}
arccoth = element arccoth { CommonAtt, DefEncAtt, empty}
arccsc = element arccsc { CommonAtt, DefEncAtt, empty}
arccsch = element arccsch { CommonAtt, DefEncAtt, empty}
arcsec = element arcsec { CommonAtt, DefEncAtt, empty}
arcsech = element arcsech { CommonAtt, DefEncAtt, empty}
arcsinh = element arcsinh { CommonAtt, DefEncAtt, empty}
arctanh = element arctanh { CommonAtt, DefEncAtt, empty}

nary−stats.class = mean | sdev | variance | median | mode
ContExp |= nary−stats.class


mean = element mean { CommonAtt, DefEncAtt, empty}
sdev = element sdev { CommonAtt, DefEncAtt, empty}
variance = element variance { CommonAtt, DefEncAtt, empty}
median = element median { CommonAtt, DefEncAtt, empty}
mode = element mode { CommonAtt, DefEncAtt, empty}

nary−constructor.class = vector | matrix | matrixrow
ContExp |= nary−constructor.class


vector = element vector { CommonAtt, DefEncAtt, BvarQ, DomainQ, ContExp∗}
matrix = element matrix { CommonAtt, DefEncAtt, BvarQ, DomainQ, ContExp∗}
matrixrow = element matrixrow { CommonAtt, DefEncAtt, BvarQ, DomainQ, ContExp∗}

unary−linalg.class = determinant | transpose
ContExp |= unary−linalg.class


determinant = element determinant { CommonAtt, DefEncAtt, empty}
transpose = element transpose { CommonAtt, DefEncAtt, empty}

nary−linalg.class = selector
ContExp |= nary−linalg.class


selector = element selector { CommonAtt, DefEncAtt, empty}

binary−linalg.class = vectorproduct | scalarproduct | outerproduct
ContExp |= binary−linalg.class


vectorproduct = element vectorproduct { CommonAtt, DefEncAtt, empty}
```

```
scalarproduct = element scalarproduct { CommonAtt, DefEncAtt, empty}
outerproduct = element outerproduct { CommonAtt, DefEncAtt, empty}

constant−set.class = integers | reals | rationals | naturalnumbers | complexes | primes | emptyset
ContExp |= constant−set.class


integers = element integers { CommonAtt, DefEncAtt, empty}
reals = element reals { CommonAtt, DefEncAtt, empty}
rationals = element rationals { CommonAtt, DefEncAtt, empty}
naturalnumbers = element naturalnumbers { CommonAtt, DefEncAtt, empty}
complexes = element complexes { CommonAtt, DefEncAtt, empty}
primes = element primes { CommonAtt, DefEncAtt, empty}
emptyset = element emptyset { CommonAtt, DefEncAtt, empty}

constant−arith.class = exponentiale | imaginaryi | notanumber | true | false | pi | eulergamma | infinity
ContExp |= constant−arith.class


exponentiale = element exponentiale { CommonAtt, DefEncAtt, empty}
imaginaryi = element imaginaryi { CommonAtt, DefEncAtt, empty}
notanumber = element notanumber { CommonAtt, DefEncAtt, empty}
true = element true { CommonAtt, DefEncAtt, empty}
false = element false { CommonAtt, DefEncAtt, empty}
pi = element pi { CommonAtt, DefEncAtt, empty}
eulergamma = element eulergamma { CommonAtt, DefEncAtt, empty}
infinity = element infinity { CommonAtt, DefEncAtt, empty}
```

References

[ABD03] Andrea Asperti, Bruno Buchberger, and James Harold Davenport, eds. *Mathematical Knowledge Management, MKM'03*. LNCS 2594. Springer Verlag, 2003.

[Abe+08] Hal Abelson et al. *ccREL: The Creative Commons Rights Expression Language*. Tech. rep. Creative Commons, Mar. 3, 2008. URL: `http://wiki.creativecommons.org/images/d/d6/Ccrel-1.0.pdf` (visited on 10/22/2009) (cit. on p. 109).

[Adi+08] Ben Adida et al. *RDFa in XHTML: Syntax and Processing*. W3C Recommendation. World Wide Web Consortium (W3C), Oct. 2008. URL: `http://www.w3.org/TR/2008/REC-rdfa-syntax-20081014/` (cit. on pp. 105, 106).

[AK02] Andrea Asperti and Michael Kohlhase. "MathML in the MoWGLI Project". In: *Second International Conference on MathML and Technologies for Math on the Web*. Chicago, USA, 2002. URL: `http://www.mathmlconference.org/2002/presentations/asperti/` (cit. on p. 176).

[AKSC03] Andrea Asperti, Michael Kohlhase, and Claudio Sacerdoti Coen. *Prototype n. D2.b Document Type Descriptors: OMDoc Proofs*. MoWGLI Deliverable. The MoWGLI Project, 2003 (cit. on p. 176).

[And02] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. second. Kluwer Academic Publishers, 2002 (cit. on p. 50).

[Asp+01] Andrea Asperti et al. "HELM and the Semantic Math-Web". In: *Theorem Proving in Higher Order Logics: TPHOLs'01*. Ed. by Richard. J. Boulton and Paul B. Jackson. LNCS 2152. Springer Verlag, 2001, pp. 59–74 (cit. on p. 176).

[Aus+03a] Ron Ausbrooks et al. *Mathematical Markup Language (MathML) Version 2.0 (second edition)*. W3C Recommendation. World Wide Web Consortium (W3C), 2003. URL: `http://www.w3.org/TR/MathML2` (cit. on pp. 14, 15, 29, 121, 128, 214).

[Aus+03b] Ron Ausbrooks et al. *Mathematical Markup Language (MathML) Version 2.0 (second edition)*. W3C Recommendation. World Wide Web Consortium (W3C), 2003. URL: `http://www.w3.org/TR/MathML2` (cit. on p. 128).

[Aut+00] Serge Autexier et al. "Towards an Evolutionary Formal Software-Development Using CASL". In: *Proceedings Workshop on Algebraic Development Techniques, WADT-99*. Ed. by C. Choppy and D. Bert. LNCS 1827. Springer Verlag, 2000, pp. 73–88 (cit. on p. 219).

[Bar80] Hendrik P. Barendregt. *The Lambda-Calculus: Its Syntax and Semantics*. North-Holland, 1980 (cit. on p. 50).

[Bau99] Judith Baur. "Syntax und Semantik mathematischer Texte – ein Prototyp". MA thesis. Saarbrücken, Germany: Fachrichtung

Computerlinguistik, Universität des Saarlandes, 1999 (cit. on p. 179).

[BB01] P. Baumgartner and A. Blohm. "Automated deduction techniques for the management of personalized documents". In: *Electronic Proceedings of the First International Workshop on Mathematical Knowledge Management: MKM'2001*. Ed. by Bruno Buchberger and Olga Caprotti. 2001. URL: `http://www.risc.uni-linz.ac.at/institute/conferences/MKM2001/Proceedings/` (cit. on p. 37).

[BC01] Henk Barendregt and Arjeh M. Cohen. "Electronic communication of mathematics and the interaction of computer algebra systems and proof assistants". In: *Journal of Symbolic Computation* 32 (2001), pp. 3–22 (cit. on p. 175).

[Ben+97] Christoph Benzmüller et al. "ΩMEGA: Towards a mathematical assistant". In: *Proceedings of the 14^{th} Conference on Automated Deduction*. Ed. by William McCune. LNAI 1249. Townsville, Australia: Springer Verlag, 1997, pp. 252–255. URL: `http://kwarc.info/kohlhase/papers/Omega97-CADE.pdf` (cit. on pp. 23, 38, 180).

[Ber91] Paul Bernays. *Axiomatic Set Theory*. Dover Publications, 1991 (cit. on p. 146).

[BHL99] *Namespaces in XML*. W3C Recommendation. World Wide Web Consortium (W3C), 1999. URL: `http://www.w3.org/TR/REC-xml-names` (cit. on p. 7).

[Bir09] Mark Birbeck. *Proposal for 'URIs everywhere'*. e-mail to `public-rdf-in-xhtml-tf@w3.org`. Nov. 25, 2009. URL: `http://lists.w3.org/Archives/Public/public-rdf-in-xhtml-tf/2009Nov/0081.html` (cit. on p. 107).

[BL98] Tim Berners-Lee. *The Semantic Web*. W3C Architecture Note. 1998. URL: `http://www.w3.org/DesignIssues/Semantic.html` (cit. on p. 32).

[BLFM98] Tim Berners-Lee, Roy T. Fielding, and Larry. Masinter. *Uniform Resource Identifiers (URI), Generic Syntax*. RFC 2717. Internet Engineering Task Force (IETF), 1998. URL: `http://www.ietf.org/rfc/rfc2717.txt` (cit. on pp. 5, 134).

[BM04] Paul V. Biron and Ashok Malhotra. *XML Schema Part 2: Datatypes Second Edition*. W3C Recommendation. World Wide Web Consortium (W3C), Oct. 28, 2004. URL: `http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/` (cit. on pp. 110, 114).

[BM09] Mark Birbeck and Shane McCarron. *CURIE Syntax 1.0. A syntax for expressing Compact URIs*. W3C Candidate Recommendation. World Wide Web Consortium (W3C), Jan. 16, 2009. URL: `http://www.w3.org/TR/2009/CR-curie-20090116` (cit. on p. 107).

[Bos+98] *Cascading Style Sheets, level 2; CSS2 Specification*. W3C Recommendation. World Wide Web Consortium (W3C), 1998. URL: `http://www.w3.org/TR/1998/REC-CSS2-19980512` (cit. on pp. 6, 92, 93, 225).

[Bou74] Nicolas Bourbaki. *Algebra I*. Elements of Mathematics. Springer Verlag, 1974 (cit. on pp. 37, 38).

[BPSM97] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. *Extensible Markup Language (XML)*. W3C Recommendation. World Wide Web Consortium (W3C), Dec. 1997. URL: `http://www.w3.org/TR/PR-xml.html` (cit. on pp. 6, 8, 9).

[Bra+02] R. Bradford et al. "Reasoning About the Elementary Functions of Complex Analysis." In: *Annals of Mathematics and Artificial Intelligence* 36 (2002), pp. 303 –318 (cit. on p. 197).

[Bra+04] Tim Bray et al. *Extensible Markup Language (XML) 1.1*. W3C Recommendation REC-xml11-20040204. World Wide Web Consortium (W3C), 2004. URL: `http://www.w3.org/TR/2004/REC-xml11-20040204/` (cit. on p. 153).

[Bru80] Nicolaas Govert de Bruijn. "A Survey of the Project AUTOMATH". In: *To H.B. Curry: Essays in Combinator Logic, Lambda Calculus and Formalisms*. Ed. by R. Hindley and J. Seldin. Academic Press, 1980, pp. 579–606 (cit. on p. 23).

[Bru94] Nicolaas Govert de Bruijn. "The Mathematical Vernacular, A Language for Mathematics with Typed Sets". In: *Selected Papers on Automath*. Ed. by R. P Nederpelt, J. H. Geuvers, and R. C. de Vrijer. Vol. 133. Studies in Logic and the Foundations of Mathematics. Elsevier, 1994, pp. 865 –935 (cit. on p. 137).

[Bus+04] Stephen Buswell et al. *The Open Math Standard, Version 2.0*. Tech. rep. The OpenMath Society, 2004. URL: `http://www.openmath.org/standard/om20` (cit. on pp. 14, 18, 29, 53, 121, 123, 125, 169, 170, 214, 218, 273).

[Bus+99] Stephen Buswell et al. *Mathematical Markup Language (MathML) 1.01 Specification*. W3C Recommendation. World Wide Web Consortium (W3C), 1999. URL: `http://www.w3.org/TR/REC-MathML` (cit. on p. 14).

[CD99] James Clark and Steve DeRose. *XML Path Language (XPath) Version 1.0*. W3C Recommendation. World Wide Web Consortium (W3C), Nov. 1999. URL: `http://www.w3.org/TR/1999/REC-xpath-19991116` (cit. on p. 10).

[Cha+92] Bruce W. Char et al. *First leaves: a tutorial introduction to Maple V*. Berlin: Springer Verlag, 1992 (cit. on p. 29).

[Cla+03] Edmund Clarke et al. "System Description: Analytica 2". In: *Proceedings of the 11^{th} Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus 2003)*. Ed. by Thérèse Hardin and Renaud Rioboo. Rome, Italy, Sept. 2003, pp. 69–74. URL: http://kwarc.info/kohlhase/papers/calculemus03.pdf (cit. on p. 203).

[Cla99a] *Associating Style Sheets with XML Documents Version 1.0*. W3C Recommendation. World Wide Web Consortium (W3C), 1999. URL: http://www.w3.org/TR/xml-stylesheet (cit. on p. 93).

[Cla99b] *XSL Transformations (XSLT) Version 1.0*. W3C Recommendation. World Wide Web Consortium (W3C), 1999. URL: http://www.w3.org/TR/xslt (cit. on p. 214).

[Com] Userland Com. *XML Remote Procedure Call Specification*. http://www.xmlrpc.com/. URL: http://www.xmlrpc.com/ (cit. on p. 81).

[Con+86] Robert L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1986 (cit. on pp. 23, 38, 181).

[Cor] Microsoft Corp. *Microsoft Internet Explorer*. http://www.microsoft.com/windows/ie/. URL: http://www.microsoft.com/windows/ie/ (cit. on p. 16).

[Cre] Creative Commons, ed. *Creative Commons*. URL: http://creativecommons.org (visited on 01/17/2012) (cit. on p. 118).

[CT04] *XML Information Set (Second Edition)*. W3C Recommendation. World Wide Web Consortium (W3C), Feb. 4, 2004. URL: http://www.w3.org/TR/2004/REC-xml-infoset-20040204 (cit. on p. 18).

[Dah01] Ingo Dahn. "Slicing Book Technology – Providing Online Support for Textbooks". In: *The 20th ICDE World Conference on Open Learning and Distance Education*. 2001 (cit. on p. 37).

[DeR+01] Steve DeRose et al. *XML Linking Language (XLink Version 1.0)*. W3C Recommendation. World Wide Web Consortium (W3C), 2001. URL: http://www.w3.org/TR/2000/REC-xlink-20010627/ (cit. on p. 206).

[DMJ03] Steven DeRose, Eve Maler, and Ron Daniel Jr. *XPointer xpointer() Scheme*. W3C Recommendation. World Wide Web Consortium (W3C), 2003. URL: http://www.w3.org/TR/xptr-xpointer (cit. on p. 10).

[DUB03a] The DCMI Usage Board. *DCMI Metadata Terms*. DCMI Recommendation. Dublin Core Metadata Initiative, 2003. URL: http://dublincore.org/documents/dcmi-terms/ (cit. on pp. 27, 113, 223).

[DUB03b] The DCMI Usage Board. *DCMI Type Vocabulary*. DCMI Recommendation. Dublin Core Metadata Initiative, 2003. URL: http://dublincore.org/documents/dcmi-type-vocabulary/ (cit. on p. 115).

[DuC97] Bob DuCharme. "Formatting Documents with DSSSL Specifications and Jade". In: *The SGML Newsletter* 10.5 (1997), pp. 6–10 (cit. on p. 6).

[DW05] Mark Davis and Ken Whistler. *Unicode Collation Algorithm.* Unicode Technical Standard #10. 2005. URL: `http://www.unicode.org/reports/tr10/` (cit. on p. 142).

[Far93] William M. Farmer. "Theory Interpretation in Simple Type Theory". In: *HOA'93, an International Workshop on Higher-order Algebra, Logic and Term Rewriting.* LNCS 816. Amsterdam, The Netherlands: Springer Verlag, 1993 (cit. on p. 189).

[FB96] N. Freed and N. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types.* RFC 2046: `http://www.faqs.org/rfcs/rfc2046.html`. 1996. URL: `http://www.faqs.org/rfcs/rfc2046.html` (cit. on pp. 42, 115, 125, 203, 249).

[FGT93] William M. Farmer, Joshua D. Guttman, and F. Javier Thayer. "IMPS: An Interactive Mathematical Proof System". In: *Journal of Automated Reasoning* 11.2 (Oct. 1993), pp. 213–248 (cit. on p. 194).

[FH97] Amy P. Felty and Douglas J. Howe. "Hybrid Interactive Theorem Proving using NuPRL and HOL". In: *Proceedings of the 14^{th} Conference on Automated Deduction.* Ed. by William McCune. LNAI 1249. Townsville, Australia: Springer Verlag, 1997, pp. 351–365 (cit. on p. 26).

[Fie97] Armin Fiedler. "Towards a Proof Explainer". In: *Proceedings of the First International Workshop on Proof Transformation and Presentation.* Ed. by J. Siekmann, F. Pfenning, and X. Huang. Schloss Dagstuhl, Germany, 1997, pp. 53–54 (cit. on p. 179).

[FK99] Andreas Franke and Michael Kohlhase. "System Description: MathWeb, an Agent-Based Communication Layer for Distributed Automated Theorem Proving". In: *Automated Deduction — CADE-16.* Ed. by Harald Ganzinger. LNAI 1632. Springer Verlag, 1999, pp. 217–221. URL: `http://kwarc.info/kohlhase/papers/cade99.pdf` (cit. on pp. 26, 81).

[Gen35] Gerhard Gentzen. "Untersuchungen über das logische Schließen I & II". In: *Mathematische Zeitschrift* 39 (1935), pp. 176–210, 572–595 (cit. on pp. 182, 183).

[GM93] M. J. C. Gordon and T. F. Melham. *Introduction to HOL – A theorem proving environment for higher order logic.* Cambridge University Press, 1993 (cit. on pp. 23, 38).

[Gog+03] George Goguadze et al. "Problems and Solutions for Markup for Mathematical Examples and Exercises". In: *Mathematical Knowledge Management, MKM'03.* Ed. by Andrea Asperti, Bruno Buchberger, and James Harold Davenport. LNCS 2594. Springer Verlag, 2003, pp. 80–93 (cit. on p. 209).

[Gol90] C. F. Goldfarb. *The SGML Handbook.* Oxford University Press, 1990 (cit. on p. 6).

[Gro+03a] Paul Grosso et al. *W3C XPointer Framework.* W3C Recommendation. World Wide Web Consortium (W3C), Mar. 25, 2003.

URL: http://www.w3.org/TR/2003/REC-xptr-framework-20030325/ (cit. on p. 10).

[Gro+03b] Paul Grosso et al. *XPointer element() Scheme*. W3C Recommendation. World Wide Web Consortium (W3C), 2003. URL: http://www.w3.org/TR/xptr-element (cit. on p. 10).

[Gro99] The Open eBook Group. *Open eBook[tm] Publication Structure 1.0*. Draft Recommendation. The OpenEBook Initiative, 1999. URL: http://www.openEbook.org (cit. on p. 116).

[Gud+03] Martin Gudgin et al. *SOAP 1.2 Part 2: Adjuncts*. W3C Recommendation. 2003. URL: http://www.w3.org/TR/2003/REC-soap12-part2-20030624 (cit. on p. 81).

[Har+] Jens Hartmann et al. *Ontology Metadata Vocabulary – OMV*. URL: http://omv2.sourceforge.net (visited on 01/12/2010) (cit. on p. 110).

[Har01] Eliotte Rusty Harold. *XML Bible*. Gold Edition. Hungry Minds, 2001 (cit. on pp. 3, 6).

[Har03] Eliotte Rusty Harold. “Effective XML”. In: Addison Wesley, 2003. Chap. 15 (cit. on p. 214).

[HC09] Aidan Hogan and Richard Cyganiak. *Frequently Observed Problems on the Web of Data*. Tech. rep. Version v0.3. Pedantic Web Group, Nov. 13, 2009. URL: http://pedantic-web.org/fops.html (cit. on p. 108).

[Her+08] Ivan Herman et al. *Team Comment on ccREL: The Creative Commons Rights Expression Language Member Submission*. W3C Team Comment. World Wide Web Consortium (W3C), Feb. 2008. URL: http://www.w3.org/Submission/2008/02/Comment (cit. on p. 110).

[Her+13] Ivan Herman et al. *RDFa 1.1 Primer – Second Edition. Rich Structured Data Markup for Web Documents*. W3C Working Goup Note. World Wide Web Consortium (W3C), Apr. 19, 2013. URL: http://www.w3.org/TR/xhtml-rdfa-primer/ (cit. on p. 108).

[HF96] Xiaorong Huang and Armin Fiedler. “Presenting Machine-Found Proofs”. In: *Proceedings of the 13th Conference on Automated Deduction*. Ed. by M. A. McRobbie and J. K. Slaney. LNAI 1104. New Brunswick, NJ, USA: Springer Verlag, 1996, pp. 221–225 (cit. on p. 179).

[HHA08] Michael Hausenblas, Ivan Herman, and Ben Adida. *RDFa – Bridging the Web of Documents and the Web of Data*. 2008. URL: http://www.w3.org/2008/Talks/1026-ISCW-RDFa/ (visited on 11/26/2009) (cit. on p. 108).

[HKW96] Reiner Hähnle, Manfred Kerber, and Christoph Weidenbach. *Common Syntax of DFG-Schwerpunktprogramm “Deduktion”*. Interner Bericht 10/96. Universität Karlsruhe, Fakultät für Informatik, 1996 (cit. on p. 26).

[Hut00] Dieter Hutter. "Management of Change in Verification Systems". In: *Proceedings 15th IEEE International Conference on Automated Software Engineering, ASE-2000*. IEEE Computer Society, 2000, pp. 23–34 (cit. on pp. 193, 195, 219).

[Ian] *Root-Zone Whois Information*. `http://www.iana.org/cctld/cctld-whois.htm`. URL: `http://www.iana.org/cctld/cctld-whois.htm` (cit. on p. 118).

[IL10] Toby A. Inkster and Christoph Lange. *RDFa Host Languages*. Feb. 23, 2010. URL: `http://rdfa.info/wiki/?title=RDFa_Host_Languages&oldid=1032` (visited on 08/27/2010) (cit. on p. 106).

[Inc03] Unicode Inc., ed. *The Unicode Standard, Version 4.0*. Addison-Wesley, 2003 (cit. on pp. 6, 214).

[JFF02] Dean Jackson, Jon Ferraiolo, and Jun Fujisawa. *Scalable Vector Graphics (SVG) 1.1 Specification*. W3C Candidate Recommendation. World Wide Web Consortium (W3C), Apr. 2002. URL: `http://www.w3.org/TR/2002/CR-SVG11-20020430` (cit. on pp. 125, 134).

[Joh05] Pete Johnston. *MARC Relator Properties in Dublin Core Metadata*. Tech. rep. UKOLN, Dec. 2005. URL: `http://www.ukoln.ac.uk/metadata/dcmi/marcrel-ex/` (cit. on p. 109).

[Jom] *JOMDoc Project — Java Library for OMDoc documents*. URL: `http://jomdoc.omdoc.org` (visited on 05/07/2011) (cit. on p. 199).

[KA03] Michael Kohlhase and Romeo Anghelache. "Towards Collaborative Content Management And Version Control For Structured Mathematical Knowledge". In: *Mathematical Knowledge Management, MKM'03*. Ed. by Andrea Asperti, Bruno Buchberger, and James Harold Davenport. LNCS 2594. Springer Verlag, 2003, pp. 147–161. URL: `http://kwarc.info/kohlhase/papers/mkm03.pdf` (cit. on p. 216).

[KD03a] Michael Kohlhase and Stan Devitt. *Bound Variables in MathML*. W3C Working Group Note. 2003. URL: `http://www.w3.org/TR/mathml-bvar/` (cit. on p. 134).

[KD03b] Michael Kohlhase and Stan Devitt. *Structured Types in MathML 2.0*. W3C Note. 2003. URL: `http://www.w3.org/TR/mathml-types/` (cit. on p. 131).

[KK06a] Andrea Kohlhase and Michael Kohlhase. "An Exploration in the Space of Mathematical Knowledge". In: *Mathematical Knowledge Management, MKM'05*. Ed. by Michael Kohlhase. LNAI 3863. Springer Verlag, 2006, pp. 17–32. URL: `http://kwarc.info/kohlhase/papers/mkm05.pdf` (cit. on pp. 197, 214).

[KK06b] Andrea Kohlhase and Michael Kohlhase. "Communities of Practice in MKM: An Extensional Model". In: *Mathematical Knowledge Management (MKM)*. Ed. by Jon Borwein and William M.

Farmer. LNAI 4108. Springer Verlag, 2006, pp. 179–193. URL: http://kwarc.info/kohlhase/papers/mkm06cp.pdf (cit. on p. 138).

[KMR08] Michael Kohlhase, Christine Müller, and Florian Rabe. "Notations for Living Mathematical Documents". In: *Intelligent Computer Mathematics*. 9th International Conference, AISC, 15th Symposium, Calculemus, 7th International Conference MKM (Birmingham, UK, July 28, 2008–Aug. 1, 2008). Ed. by Serge Autexier et al. LNAI 5144. Springer Verlag, 2008, pp. 504–519. URL: http://omdoc.org/pubs/mkm08-notations.pdf (cit. on p. 199).

[Knu84] Donald E. Knuth. *The TEXbook*. Addison Wesley, 1984 (cit. on p. 4).

[Koh] Michael Kohlhase. "CodeML: An Open Markup Format the Content and Presentatation of Program Code". URL: https://github.com/OMDoc/CodeML/doc/spec/codeml.pdf (cit. on pp. 138, 202).

[Koh06a] Michael Kohlhase, ed. *Mathematical Knowledge Management, MKM'05*. LNAI 3863. Springer Verlag, 2006.

[Koh06b] Michael Kohlhase. *OMDoc – An open markup format for mathematical documents [Version 1.2]*. LNAI 4180. Springer Verlag, Aug. 2006. URL: http://omdoc.org/pubs/omdoc1.2.pdf (cit. on p. 109).

[KR93] Hans Kamp and Uwe Reyle. *From Discourse to Logic: Introduction to Model-Theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Dordrecht: Kluwer, 1993 (cit. on p. 149).

[KR96] Hans Kamp and Uwe Reyle. "A Calculus for First Order Discourse Representation Structures". In: *Journal of Logic, Language and Information* 5.3-4 (1996), pp. 297–348 (cit. on p. 144).

[Lam94] Leslie Lamport. *LaTeX: A Document Preparation System, 2/e*. Addison Wesley, 1994 (cit. on p. 4).

[LS99] Ora Lassila and Ralph R. Swick. *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation. World Wide Web Consortium (W3C), 1999. URL: http://www.w3.org/TR/1999/REC-rdf-syntax (cit. on pp. 100, 119).

[MAH06] Till Mossakowski, Serge Autexier, and Dieter Hutter. "Development Graphs – Proof Management for Structured Specifications". In: *Journal of Logic and Algebraic Programming* 67.1–2 (2006), pp. 114–145 (cit. on pp. 192, 194, 219).

[Mat] *MathPlayer ¡Display MathML in your browser¿*. http://www.dessci.com/en/products/mathplayer. URL: http://www.dessci.com/en/products/mathplayer (cit. on p. 16).

[McC10] *XHTML^{tm} Modularization 1.1 - Second Edition*. W3C Recommendation. World Wide Web Consortium (W3C), 2010. URL:

http://www.w3.org/TR/xhtml-modularization (cit. on pp. 139, 141).

[McC97] William McCune, ed. *Proceedings of the 14^{th} Conference on Automated Deduction.* LNAI 1249. Townsville, Australia: Springer Verlag, 1997.

[Mei00] Andreas Meier. "System Description: TRAMP: Transformation of Machine-Found Proofs into ND-Proofs at the Assertion Level". In: *Automated Deduction – CADE-17.* Ed. by David McAllester. LNAI 1831. Springer Verlag, 2000, pp. 460–464 (cit. on p. 85).

[Mel+03] Erica Melis et al. "Knowledge Representation and Management in ActiveMath". In: *Annals of Mathematics and Artificial Intelligence* 38.1–3 (2003), pp. 47–64 (cit. on p. 80).

[Mit03] Nilo Mitra. *SOAP 1.2 Part 0: Primer.* W3C Recommendation. 2003. URL: http://www.w3.org/TR/2003/REC-soap12-part0-20030624 (cit. on p. 81).

[Miz] *Mizar Mathematical Library.* Web Page at http://www.mizar.org/library. seen May 2008. URL: http://www.mizar.org/library (cit. on p. 22).

[Mos04] P. D. Mosses, ed. CASL *Reference Manual.* LNCS 2960 (IFIP Series). Springer Verlag, 2004 (cit. on pp. 23, 59, 171, 192, 219).

[MR+] Peter Murray-Rust et al. *Chemical Markup Language (CML).* seen January 2007. URL: http://cml.sourceforge.net/ (visited on 01/08/2007) (cit. on p. 138).

[MR03] *MARC code list for Relators, Sources, Description Conventions.* 2003. URL: http://www.loc.gov/marc/relators (cit. on pp. 113, 116).

[MSLK01] M. Murata, S. St. Laurent, and D. Kohn. *XML Media Types.* RFC 3023. Jan. 2001. URL: ftp://ftp.isi.edu/in-notes/rfc3023.txt (cit. on p. 115).

[Mül10] Christine Müller. "Adaptation of Mathematical Documents". PhD thesis. Jacobs University Bremen, 2010. URL: http://kwarc.info/cmueller/papers/thesis.pdf (cit. on p. 199).

[MVW05] Jonathan Marsh, Daniel Veillard, and Norman Walsh. xml:id *Version 1.0.* W3C Recommendation. World Wide Web Consortium (W3C), Sept. 9, 2005. URL: http://www.w3.org/TR/2005/REC-xml-id-20050909/ (cit. on pp. 10, 27, 92, 223).

[NS81] Alan Newell and Herbert A. Simon. "Computer Science as empirical inquiry: Symbols and search". In: *Communications of the Association for Computing Machinery* 19 (1981), pp. 113–126 (cit. on p. 152).

[Odl95] A. M. Odlyzko. "Tragic loss or good riddance? The impending demise of traditional scholarly journals". In: *International Journal of Human-Computer Studies* 42 (1995), pp. 71–122 (cit. on p. XI).

[OMCD] OpenMath *Content Dictionaries*. http://www.openmath.org/cd/. seen June2008. URL: http://www.openmath.org/cd/ (cit. on pp. 18, 20, 53, 130).

[OMDoc] Michael Kohlhase. OMDoc*: An open markup format for mathematical documents (latest released version)*. URL: http://www.omdoc.org/pubs/spec.pdf (cit. on p. 90).

[Org] The Mozilla Organization. *Mozilla*. http://www.mozilla.org. URL: http://www.mozilla.org (cit. on p. 16).

[ORS92] S. Owre, J. M. Rushby, and N. Shankar. "PVS: A Prototype Verification System". In: *Proceedings of the 11th Conference on Automated Deduction*. Ed. by D. Kapur. LNCS 607. Saratoga Springs, NY, USA: Springer Verlag, 1992, pp. 748–752 (cit. on p. 59).

[Orw49] George Orwell. *Nineteen Eighty-Four*. London: Secker & Warburg, 1949 (cit. on p. 110).

[Pal+09] Raúl Palma et al. "Change Representation For OWL 2 Ontologies". In: *OWL: Experiences and Directions (OWLED)*. Ed. by Rinke Hoekstra and Peter F. Patel-Schneider. Oct. 2009 (cit. on p. 110).

[Pau94] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*. LNCS 828. Springer Verlag, 1994 (cit. on p. 181).

[Pfe01] Frank Pfenning. "Logical Frameworks". In: *Handbook of Automated Reasoning*. Ed. by Alan Robinson and Andrei Voronkov. Vol. I and II. Elsevier Science and MIT Press, 2001 (cit. on p. 21).

[Pfe91] Frank Pfenning. "Logic Programming in the LF Logical Framework". In: *Logical Frameworks*. Ed. by Gérard P. Huet and Gordon D. Plotkin. Cambridge University Press, 1991 (cit. on p. 23).

[Pie80] John R. Pierce. *An Introduction to Information Theory. Symbols, Signals and Noise*. Dover Publications Inc., 1980 (cit. on p. 13).

[PN90] Lawrence C. Paulson and Tobias Nipkow. *Isabelle Tutorial and User's Manual*. Tech. rep. 189. Computer Laboratory, University of Cambridge, Jan. 1990 (cit. on p. 23).

[PS08] Eric Prud'hommeaux and Andy Seaborne. *SPARQL Query Language for RDF*. W3C Recommendation. World Wide Web Consortium (W3C), Jan. 15, 2008. URL: http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/ (cit. on p. 107).

[Rei87] Glenn C. Reid. *PostScript, Language, Program Design*. Addison Wesley, 1987 (cit. on p. 4).

[RHJ98] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. *HTML 4.0 Specification*. W3C Recommendation REC-html40. World Wide Web Consortium (W3C), Apr. 1998. URL: http://www.w3.org/TR/PR-xml.html (cit. on p. 5).

[Rud92] Piotr Rudnicki. "An Overview of the MIZAR Project". In: *Proceedings of the 1992 Workshop on Types and Proofs as Programs*. 1992, pp. 311–332 (cit. on pp. 23, 38).

[SC06] Claudio Sacerdoti Coen. “Explanation in Natural Language of $\overline{\lambda}\mu\overline{\mu}$-terms”. In: *Mathematical Knowledge Management, MKM'05*. Ed. by Michael Kohlhase. LNAI 3863. Springer Verlag, 2006 (cit. on p. 176).

[Sie+00] Jörg Siekmann et al. “Adaptive Course Generation and Presentation”. In: *Proceedings of ITS-2000 workshop on Adaptive and Intelligent Web-Based Education Systems*. Ed. by P. Brusilovski and Chrisoph Peylo. Montreal, 2000 (cit. on p. 95).

[Sie+02] Jörg Siekmann et al. “Proof Development with *Ω*MEGA”. In: *Automated Deduction — CADE-18*. Ed. by Andrei Voronkov. LNAI 2392. Springer Verlag, 2002, pp. 144–149 (cit. on p. 25).

[SSY94] Geoff Sutcliffe, Christian Suttner, and Theodor Yemenis. “The TPTP Problem Library”. In: *Proceedings of the 12th Conference on Automated Deduction*. Ed. by Alan Bundy. LNAI 814. Nancy, France: Springer Verlag, 1994 (cit. on p. 26).

[SZS04] G. Sutcliffe, J. Zimmer, and S. Schulz. “TSTP Data-Exchange Formats for Automated Theorem Proving Tools”. In: *Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems*. Ed. by W. Zhang and V. Sorge. Frontiers in Artificial Intelligence and Applications 112. IOS Press, 2004, pp. 201–215 (cit. on p. 159).

[TDO07] Giovanni Tummarello, Renaud Delbru, and Eyal Oren. “Sindice.com: Weaving the Open Linked Data”. In: *ISWC/ASWC*. 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007 (Busan, Korea, Nov. 11, 2007–Nov. 15, 2007). Ed. by Karl Aberer et al. LNCS 4825. Springer Verlag, 2007, pp. 552–565. ISBN: 978-3-540-76297-3. DOI: 10.1007/978-3-540-76298-0_40 (cit. on p. 106).

[The02] The W3C HTML Working Group. *XHTML 1.0 The Extensible HyperText Markup Language (Second Edition) – A Reformulation of HTML 4 in XML 1.0*. W3C Recommendation. World Wide Web Consortium (W3C), Aug. 1, 2002. URL: http://www.w3.org/TR/2002/REC-xhtml1-20020801 (cit. on pp. 206, 214, 220).

[Tho91] Simon Thompson. *Type Theory and Functional Programming*. International Computer Science Series. Addison-Wesley, 1991 (cit. on p. 185).

[Urla] http://creativecommons.org/worldwide (cit. on p. 119).

[Urlb] *Creative Commons Worldwide*. http://creativecommons.org/worldwide. URL: http://creativecommons.org/worldwide (cit. on p. 118).

[Urlc] *Document Object Model DOM*. http://www.w3.org/DOM/. URL: http://www.w3.org/DOM/ (cit. on p. 18).

[Urld] *Mizar Language*. http://mizar.org/language. seen III 2006. URL: http://mizar.org/language (cit. on p. 27).

[Vat] Irène Vatton. *Welcome to Amaya*. http://www.w3.org/Amaya. URL: http://www.w3.org/Amaya (cit. on p. 16).

[Vli03] Eric van der Vlist. *Relax NG*. O'Reilly, 2003 (cit. on p. 8).

[Vor02] Andrei Voronkov, ed. *Automated Deduction — CADE-18*. LNAI 2392. Springer Verlag, 2002.

[W3c] *W3 Consortium*. seen February 2007. URL: http://www.w3.org (visited on 01/18/2010) (cit. on p. 6).

[Wei97] Christoph Weidenbach. "SPASS: Version 0.49". In: *Journal of Automated Reasoning* 18.2 (1997). Special Issue on the CADE-13 Automated Theorem Proving System Competition, pp. 247–252 (cit. on p. 81).

[WM08] Norman Walsh and Leonard Muellner. *DocBook 5.0: The Definitive Guide*. O'Reilly, 2008 (cit. on p. 220).

[Wol02] Stephen Wolfram. *The Mathematica Book*. Cambridge University Press, 2002 (cit. on pp. 14, 29).

[WR10] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*. 2nd ed. Vol. I. Cambridge, UK: Cambridge University Press, 1910 (cit. on p. 21).

[Xml] *XML Schema*. Web page at http://www.w3.org/XML/Schema. URL: http://www.w3.org/XML/Schema (cit. on pp. 8, 9).

[ZK02] Jürgen Zimmer and Michael Kohlhase. "System Description: The MathWeb Software Bus for Distributed Mathematical Reasoning". In: *Automated Deduction — CADE-18*. Ed. by Andrei Voronkov. LNAI 2392. Springer Verlag, 2002, pp. 247–252. URL: http://kwarc.info/kohlhase/papers/cade02.pdf (cit. on pp. 26, 81).