Michael Kohlhase
Computer Science
International University Bremen
m.kohlhase@iu-bremen.de

An Open Markup Format for Mathematical Documents

OMDoc [Version 1.2]

August 11, 2009

This Document is an online version of the OMDoc 1.2 Specification published by Springer Verlag as number 4180 in the "Lecture Notes in Artificial Intelligence" (LNAI) series. It corrects all known errata; the corrections are marked up in the text and tabulated in the appendix F.

Source Information	revision 8723, last change August 11, 2009 by kohlhase
https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/doc/spec/spec.tex	

This work is licensed by the Creative Commons Share-Alike license http://creativecommons.org/licenses/by-sa/2.5/: the contents of this specification or fragments thereof may be copied and distributed freely, as long as they are attributed to the original author and source, derivative works (i.e. modified versions of the material) may be published as long as they are also licenced under the Creative Commons Share-Alike license.

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 1.2 of the OMDoc format, the final and mature release of OMDoc 1. 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 IV.
- 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 the International University Bremen have supported the author while working on revisions for versions 1.1 and 1.2.

Contents

Foreword IX
Preface XI

Part I Setting the Stage for Open Mathematical Documents

1 Document Markup for the Web 3
1.1 Structure vs. Appearance in Markup 3
1.2 Markup for the World Wide Web 5
1.3 XML, the eXtensible Markup Language 6

2 Markup for Mathematical Knowledge 13
2.1 Mathematical Objects and Formulae 14
2.2 Mathematical Texts and Statements 21
2.3 Large-Scale Structure and Context in Mathematics 22

3 Open Mathematical Documents 25
3.1 A Brief History of the OMDoc Format 25
3.2 Three Levels of Markup 28
3.3 Situating the OMDoc Format 29
3.4 The Future: An Active Web of (Mathematical) Knowledge 31

Part II An OMDoc Primer

4 Textbooks and Articles 37
4.1 Minimal OMDoc Markup 39
4.2 Structure and Statements 41
4.3 Marking up the Formulae 43
4.4 Full Formalization 48

5 OpenMath Content Dictionaries 53

6 Structured and Parametrized Theories 59

7 A Development Graph for Elementary Algebra 65

8 Courseware and the Narrative/Content Distinction 71
8.1 A Knowledge-Centered View 73
8.2 A Narrative-Structured View 77
8.3 Choreographing Narrative and Content OMDoc 79
8.4 Summary 80

9 Communication between Systems 81

Part III The OMDoc Document Format

10 OMDoc as a Modular Format 89
10.1 The OMDoc Namespaces 89
10.2 Common Attributes in OMDoc 90

11 Document Infrastructure 95
11.1 The Document Root 96
11.2 Metadata 97
11.3 Document Comments 98
11.4 Document Structure 100
11.5 Sharing Document Parts 100

12 Metadata 103
12.1 The Dublin Core Elements (Module DC) 104
12.2 Roles in Dublin Core Elements 107
12.3 Managing Rights 108
12.4 Inheritance of Metadata 110

13 Mathematical Objects 113
13.1 OpenMath 113
13.2 Content MathML 120
13.3 Representing Types in Content-MathML and OpenMath 123
13.4 Semantics of Variables 125
13.5 Legacy Representation for Migration 126

14 Mathematical Text 129
14.1 Multilingual Mathematical Vernacular 130
14.2 Formal Mathematical Properties 131
14.3 Text Fragments and their Rhetoric/Mathematical Roles 133
14.4 Phrase-Level Markup of Mathematical Vernacular 134

14.5 Technical Terms . . . 135
14.6 Rich Text Structure . . . 137

15 Mathematical Statements . . . 141
15.1 Types of Statements in Mathematics . . . 141
15.2 Theory-Constitutive Statements in OMDoc . . . 144
15.3 The Unassuming Rest . . . 150
15.4 Mathematical Examples in OMDoc . . . 155
15.5 Inline Statements . . . 157
15.6 Theories as Structured Contexts . . . 158

16 Abstract Data Types . . . 165

17 Representing Proofs . . . 169
17.1 Proof Structure . . . 171
17.2 Proof Step Justifications . . . 173
17.3 Scoping and Context in a Proof . . . 177
17.4 Formal Proofs as Mathematical Objects . . . 180

18 Complex Theories . . . 183
18.1 Inheritance via Translations . . . 183
18.2 Postulated Theory Inclusions . . . 187
18.3 Local/Required Theory Inclusions . . . 189
18.4 Induced Assertions . . . 191
18.5 Development Graphs . . . 192

19 Notation and Presentation . . . 199
19.1 Styling OMDoc Elements . . . 200
19.2 A Restricted Style Language . . . 202
19.3 Notation of Symbols . . . 204
19.4 Presenting Bound Variables . . . 210

20 Auxiliary Elements . . . 215
20.1 Non-XML Data and Program Code in OMDoc . . . 216
20.2 Applets and External Objects in OMDoc . . . 219

21 Exercises . . . 223

22 Document Models for OMDoc . . . 227
22.1 XML Document Models . . . 227
22.2 The OMDoc Document Model . . . 230
22.3 OMDoc Sub-Languages . . . 231

Part IV OMDoc Applications, Tools, and Projects

23 OMDoc resources . . . 237
23.1 The OMDoc Web Site, Wiki, and Mailing List . . . 237
23.2 The OMDoc distribution . . . 237
23.3 The OMDoc bug tracker . . . 238
23.4 An XML catalog for OMDoc . . . 238
23.5 External Resources . . . 239

24 Validating OMDoc Documents . . . 241
24.1 Validation with Document Type Definitions . . . 242
24.2 Validation with RelaxNG Schemata . . . 245
24.3 Validation with XML Schema . . . 246

25 Transforming OMDoc . . . 249
25.1 Extracting and Linking XSLT Templates . . . 249
25.2 Interfaces for Systems . . . 251
25.3 Presenting OMDoc to Humans . . . 254

26 Applications and Projects . . . 255
26.1 Introduction . . . 255
26.2 QMath Parser . . . 258
26.3 Sentido Integrated Environment . . . 261
26.4 MBase . . . 266
26.5 A Search Engine for Mathematical Formulae . . . 268
26.6 Semantic Interrelation and Change Management . . . 273
26.7 MathDox . . . 277
26.8 ActiveMath . . . 281
26.9 Authoring Tools for ActiveMath . . . 287
26.10SWiM – An OMDoc-based Semantic Wiki . . . 289
26.11Induction Challenge Problems . . . 293
26.12Maya . . . 296
26.13Hets . . . 301
26.14CPoint . . . 305
26.15sTeX: A LaTeX-Based Workflow for OMDoc . . . 310
26.16An Emacs mode for editing OMDoc Documents . . . 314
26.17Converting Mathematica Notebooks to OMDoc . . . 318
26.18Standardizing Context in System Interoperability . . . 321
26.19Proof Assistants in Scientific Editors . . . 326
26.20VeriFun . . . 330

Part V Appendix

A Changes to the specification . . . 337
A.1 Changes from 1.1 to 1.2 . . . 338
A.2 Changes from 1.0 to 1.1 . . . 346

B Quick-Reference 353

C Table of Attributes 361

D The RelaxNG Schema for OMDoc 369
D.1 The Sub-Language Drivers 369
D.2 Common Attributes 371
D.3 Module MOBJ: Mathematical Objects and Text 372
D.4 Module MTXT: Mathematical Text 372
D.5 Module DOC: Document Infrastructure 373
D.6 Module DC: Dublin Core Metadata 374
D.7 Module ST: Mathematical Statements 375
D.8 Module ADT: Abstract Data Types 378
D.9 Module PF: Proofs and Proof objects 378
D.10 Module CTH: Complex Theories 379
D.11 Module DG: Development Graphs 380
D.12 Module RT: Rich Text Structure 381
D.13 Module EXT: Applets and non-XML data 382
D.14 Module PRES: Adding Presentation Information 383
D.15 Module QUIZ: Infrastructure for Assessments 384

E The RelaxNG Schemata for Mathematical Objects 387
E.1 The RelaxNG Schema for OpenMath 387
E.2 The RelaxNG Schema for MathML 388

F The Errata 401

References 405

Index 421

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 [**?**] 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 locator**s (**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 [Bos98] 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 [**?**], an international interest group of universities and web industry) has developed a language framework for Internet markup languages called XML (eXtensible Markup Language) [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://www.mathweb.org/omdoc"
        xmlns:om="http://www.openmath.org/OpenMath">
  <CMP xml:lang='en'>
   The number
   <om:OMOBJ><om:OMS cd="nums1" name="pi"/><om:OMOBJ>
   is  irrational .
  </CMP>
</omtext>
```

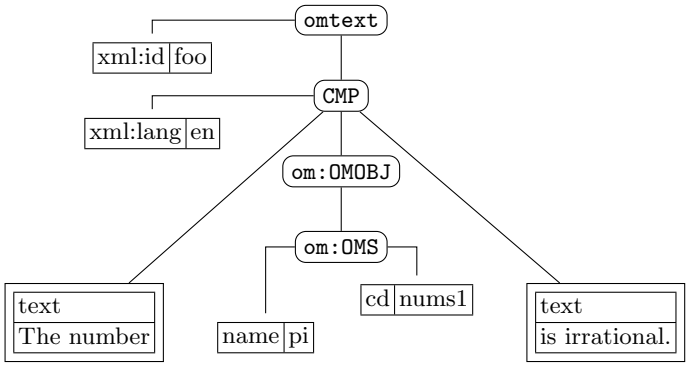


Fig. 1.1. An XML Document as a Tree

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 element**s (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 node**s — 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 **namespace**s [Bra99]. 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<b³]]>	≙	a<b³

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 24 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 V1.2//EN"
                       "http://omdoc.org/dtd/omdoc.dtd">
4 <omdoc xml:id="example-omdoc" xmlns="http://www.mathweb.org/omdoc">
...
</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 23.4 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 [GMMW03] 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 [**?**]). Fragment identifiers in the element() scheme [**?**] 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 [ABC+03a] and OpenMath [BCC+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 [BDD+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* [ABC+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.

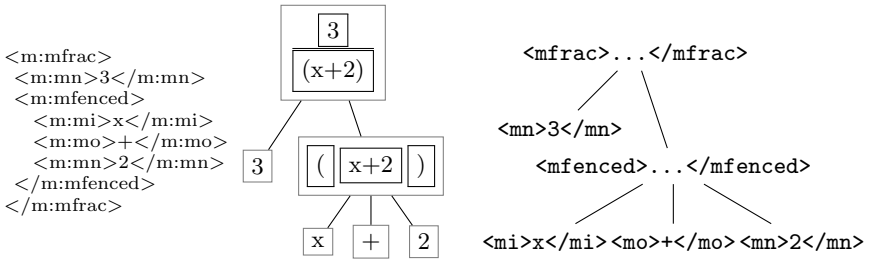


Fig. 2.2. The Layout Tree for the Formula $\frac{3}{x+2}$

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 [Sci]), 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).

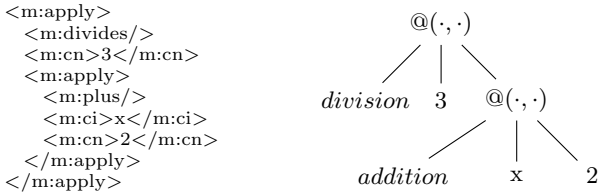


Fig. 2.3. The functional Structure of $\frac{3}{x+2}$

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.

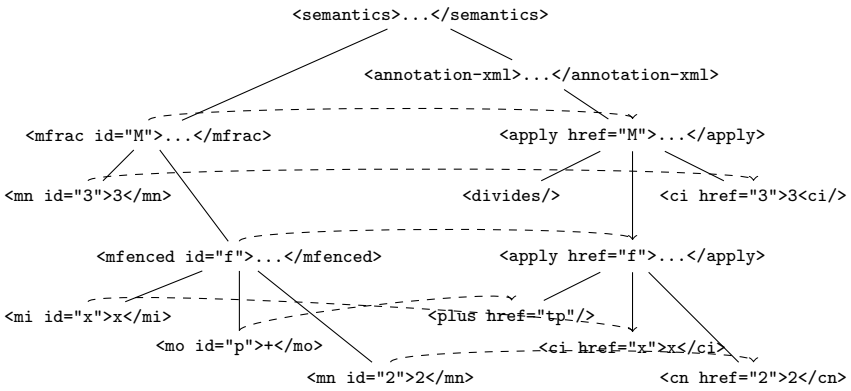


Fig. 2.4. Mixing Presentation and Content-MathML

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* [BCC+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 [OMC08]).

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 [DOM] or the or its infoset [Cow04] 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` [OMC08] 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> for all a,b | a + b = b + a </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 statement**s like the ones in the document fragment in Figure 2.5 are delimited by keywords (e.g. **Definition**, **Lemma** and □) 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. □

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 framework**s (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 [Miz08] 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 [dB80] NUPRL [CAB+86], HOL [GM93], MIZAR [Rud92] and ΩMEGA [BCF+97] which utilize strong logical systems that allow to express both mathematical statements and proofs as mathematical objects. Some systems like ISABELLE [PN90] and 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 [CoF04]) 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 [SBB+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 mathe-

matical 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 [Miz06] 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.

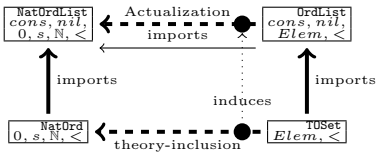
Level of Representation	OMDoc Example
Theory Level: Development Graph • Inheritance via symbol-mapping • Theory inclusion via proof-obligations • Local (one-step) vs. global links	
Statement Level: • Axiom, definition, theorem, proof, example,... • Structure explicit in statement forms and references	<definition for="plus" type="recursive"> <CMP>Addition is defined by recursion on the second argument </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 [BCC+04] and Content-MathML [ABC+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™ [CGG+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.

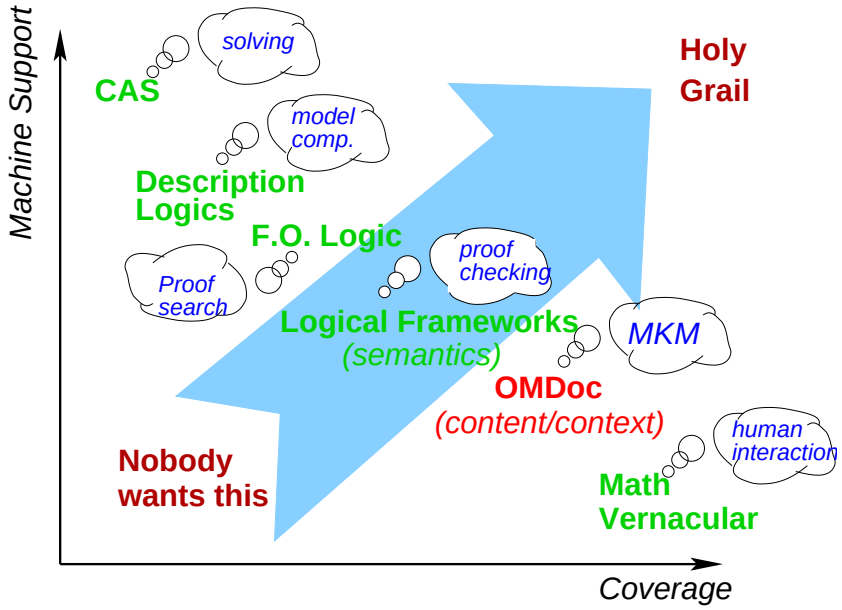


Fig. 3.2. Situating Content Markup: Math. Knowledge Management

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 some-

thing 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 26.4), 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 26.8 can reassemble 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 25), but also to copy

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 $\bot$ 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]

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 [CAB+86], HOL [GM93], MIZAR [Rud92] and OMEGA [BCF+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

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE omdoc PUBLIC "-//OMDoc//DTD OMDoc Basic V1.2//EN"
                "http://omdoc.org/dtd/omdoc-basic.dtd" []>

<omdoc xml:id="algebra1.omdoc" version="1.2" modules="@basic"
       xmlns:dc="http://purl.org/dc/elements/1.1/"
       xmlns:cc="http://creativecommons.org/ns"
       xmlns="http://www.mathweb.org/omdoc">
  <metadata>
    <dc:title>Laws of Composition</dc:title>
    <dc:creator role="trl">Michael Kohlhase</dc:creator>
    <dc:date action="created">2002-01-03T07:03:00</dc:date>
    <dc:date action="updated">2002-11-23T18:17:00</dc:date>
    <dc:description>
      A first migration step for a fragment of Bourbaki's Algebra
    </dc:description>
    <dc:source>
      Nicolas Bourbaki, Algebra, Springer Verlag 1989, ISBN 0-387-19373-1
    </dc:source>
    <dc:type>Text</dc:type>
    <dc:format>application/omdoc+xml</dc:format>
    <dc:rights>Copyright (c) 2005 Michael Kohlhase</dc:rights>
    <cc:license>
      <cc:permissions reproduction="permitted" distribution="permitted"
                      derivative_works="permitted"/>
      <cc:prohibitions commercial_use="permitted"/>
      <cc:requirements notice="required" copyleft="required" attribution="required"/>
    </cc:license>
  </metadata>

  <omtext xml:id="all">
    <CMP xml:lang="en">
      {\sc Definition 1.} Let $E$ be a set. A mapping $E \times E$ is called a law of
      ...
      mappings from $E$ into $E$ ({\emph{Set Theory}}, II, §5, no. 2).
    </CMP>
  </omtext>
</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.	24.1 p. 242
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 1.2) 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. 96
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.	10.1 p. 89
9–29	The metadata for the whole document in Dublin Core format	11.2 p. 98
10	The title of the document	12.1 p. 104
11	The document creator, here in the role of a translator	12.2 p. 107
12	The date and time of first creation of the document in ISO 8601 norm format.	12.1 p. 105
13	The date and time of the last update to the document in ISO 8601 norm format.	12.1 p. 105
14–16	A short description of the contents of the document	12.1 p. 105
17–19	Here we acknowledge that the OMDoc document is just a translation from an earlier work.	12.1 p. 106
20	The type of the document, this can be `Dataset` (un-ordered mathematical knowledge) or `Text` (arranged for human consumption).	12.1 p. 106

21	The format/MIME type [FB96] of the document, for OMDoc, this is `application/omdoc+xml`.	12.1 p. 106
22	The copyright resides with the creator of the OMDoc document	12.1 p. 106
23–28	The creator licenses the document to the world under certain conditions as specified in the Creative Commons license specified in this element.	12.3 p. 108
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.	12.3 p. 109
26	The `cc:prohibitions` can be used to prohibit certain uses of the document, but this one is unencumbered.	12.3 p. 109
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).	12.3 p. 109
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".	14.3 p. 133
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 p. 130
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. 96

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 $[32–36]^1_r$ in Listing 4.1, we will now have the representation in Listing 4.2. Err(1)

Listing 4.2. The segmented text

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

[1] Erratum! wrong reference (original text was: "19–25")

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

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

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

BErr(2)

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.	14.3 p. 133
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. 127
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.	14.3 p. 133

[2] Erratum: wrong cross-reference for "line 16"

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.	11.4 p. 100
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.	12.1 p. 104
18	The `type` attribute of this `omtext` element classifies this text fragment as an example.	14.3 p. 133

EErr(2)

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. BErr(3)

BErr(4)

BErr(5)

Listing 4.3. The definition of a magma with OpenMath objects

```
<!DOCTYPE omdoc PUBLIC "-//OMDoc//DTD OMDoc CD V1.2//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>
      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>
```

[3] Erratum: `for` attribute on `definition` should be of type `NCNames`

[4] Erratum: should be "definiendum" not "definiens"

[5] Erratum: should be definiendum-applied not definiens-applied

```
21        </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"/>
26            <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>
31          <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"/>
36              <om:OMV name="E"/><om:OMV name="E"/>
              </om:OMA>
            </om:OMA>
          </om:OMOBJ> is called the
          <term cd="magmas" name="law_of_composition"
41                      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>.
46      </CMP>
      </definition>
...
</theory>
...
```

EErr(5)
EErr(4)
EErr(3)

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. 232
5	The start tag of a theory. We need this, since symbols and definitions can only appear inside `theory` elements.	15.6 p. 158

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. 159
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. 144
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. 144
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.148
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. 114
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. 114
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. 114
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.	14.5 p. 136

[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
        <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>
          <om:OMBIND><om:OMS cd="fns1" name="lambda"/>
```

```
            <om:OMBVAR>
              <om:OMV name="X"/><om:OMV name="Y"/>
35          </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"/>
40            </om:OMA>
              <om:OMA><om:OMS cd="sets" name="intersection"/>
                <om:OMV name="X"/>
                <om:OMV name="Y"/>
              </om:OMA>
45          </om:OMA>
          </om:OMBIND>
        </om:OMOBJ>
        are <term cd="magmas" name="law_of_composition>laws of composition</term>
        on the set of subsets of a set
50      <om:OMOBJ><om:OMS cd="magmas" name="E"/></om:OMOBJ>.
      </CMP>
    </example>

    <example xml:id="e2.magma" for="law_of_composition" type="for">
55    <CMP>
        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
60      <term cd="nats" name="power">exponentiation</term> are ...
      </CMP>
    </example>
  </omgroup>
</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

```
1   <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"
6       <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` BErr(6) (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>
    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>.
  </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>
```

[6] Erratum: `for` attribute on `definition` should be of type `NCNames`

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 $\lambda E, F.set(E) \wedge F : E \times E \to E$[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 $(\lambda E, F.set(E) \wedge F : E \times E \to E)(A, B)$ which in turn reduces[3] to $set(A) \wedge B : A \times A \to A$ 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.

EErr(6)

BErr(7)

BErr(8)

Listing 4.7. The formal definition of a magma

```
<definition xml:id="magma.def" for="magma" type="implicit">
 <CMP> A set with a law of composition is called a
   <term cd="magmas" name="magma" role="definiendum">magma</term>.
 </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"/>
               <om:OMV name="F"/>
             </om:OMA>
           </om:OMA>
         </om:OMBIND>
       </om:OMA>
```

[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.

[7] Erratum: `for` attribute on `definition` should be of type `NCNames`

[8] Erratum: should be "definiendum" not "definiens"

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

EErr(8)
EErr(7)

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> 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>.
  </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 [BCC+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 [OMC08] 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>
```

```
  <Name> lcm </Name>
15  <Description>
      The symbol to represent the n-ary function to return the least common
      multiple of its arguments.
    </Description>

20  <CMP> lcm(a,b) = a*b/gcd(a,b) </CMP>
    <FMP>... </FMP>

    <CMP>
      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
      Integer and lcm(a,b) > c.
    </CMP>
    <FMP>...</FMP>
    ...
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.

BErr(9)

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

```
<?xml version="1.0" encoding="utf-8"?>
<omdoc xml:id="arith1.omdoc" modules="@cd"
       xmlns:dc="http://purl.org/dc/elements/1.1/">

5   <metadata>
      <dc:title>The OpenMath Content Dictionary arith1.ocd in OMDoc Form</dc:title>
      <dc:creator role="trl">Michael Kohlhase</dc:creator>
      <dc:creator role="ant">The OpenMath Society</dc:creator>
      <dc:date action="updated"> 2004-01-17T09:04:03Z </dc:date>
10    <dc:source>
        Derived from the OpenMath CD http://www.openmath.org/cd/arith1.ocd.
      </dc:source>
```

[9] Erratum: `for` attribute on `definition` should be of type `NCNames`

```
    <dc:type>Text</dc:type>
    <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>We define <OMOBJ><OMR href="#lcm-def.O"/></OMOBJ>
        as <OMOBJ><OMR href="#lcm-def.1"/></OMOBJ></CMP>
      <CMP xml:lang="de">
        Wir definieren <OMOBJ><OMR href="#lcm-def.O"/></OMOBJ>
        als <OMOBJ><OMR href="#lcm-def.1"/></OMOBJ></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>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>.
      </CMP>
      <CMP xml:lang="de">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>.
      </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>
```

EErr(9)

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 25.3.

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 [CoF04]) standard to the Pvs theorem prover [ORS92], since it uses most language elements without becoming too unwieldy to present.

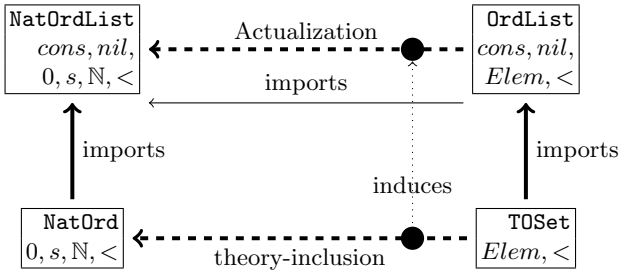


Fig. 6.1. A Structured Specification of Lists (of Natural Numbers)

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.

BErr(10)

```
<theory xml:id="TOSet">
  <symbol name="set"/>
  <symbol name="ord"/>
  <axiom xml:id="toset"><CMP>ord is a total order on set.</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>≤ exists.</CMP></assertion>
  <assertion xml:id="leq.unique"><CMP>≤ is unique</CMP></assertion>
  <assertion xml:id="leq.TO"><CMP>≤ is a total order on Nat.</CMP></assertion>
</theory>
```

EErr(10)

[10] Erratum: `for` attribute on `definition` should be of type `NCNames`

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 $\leq$. BErr(11)

```
<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>
```

[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.

[11] Erratum: `for` attribute on `definition` should be of type `NCNames`

```
      <constructor name="nil"/>
    </sortdef>
  </adt>

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

EErr(11)

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 18.5 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"/>
```

```
8     <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.

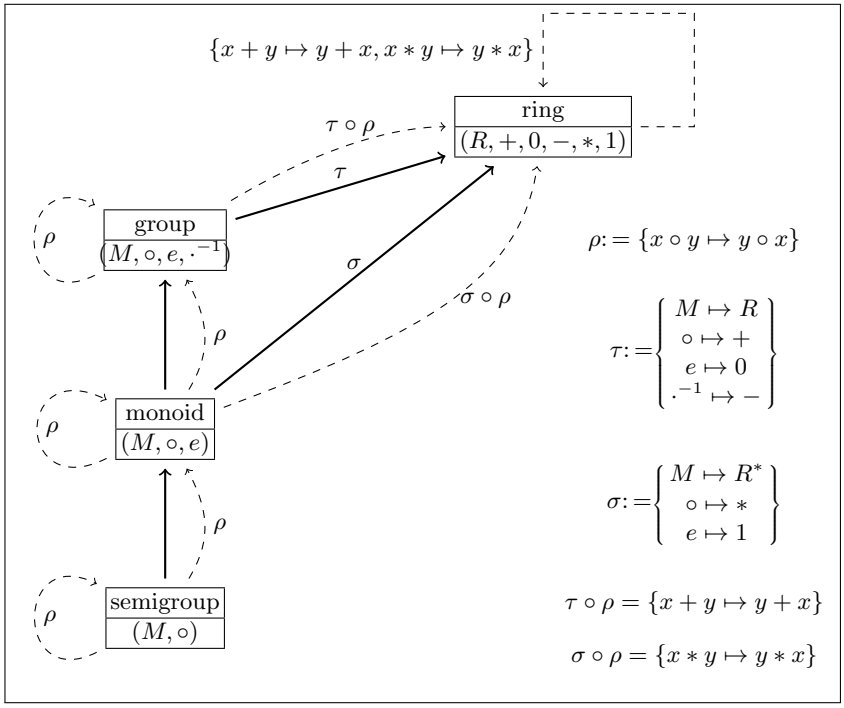


Fig. 7.1. A Development Graph for Elementary Algebra

We start off with the theory for *semigroup*s. 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 *monoid*s 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 $\circ$.

BErr(12)

```
<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>
```

EErr(12)

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-

[12] Erratum: `for` attribute on `definition` should be of type `NCNames`

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. BErr(13)

```
<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>
```

EErr(13)

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"/>
```

[13] Erratum: `for` attribute on `definition` should be of type `NCNames`

```
</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].

BErr(14)

```
<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>
```

EErr(14)

Again, we have to establish some axiom inclusions to justify the theory self-inclusion 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">
```

[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.

[14] ERRATUM: FOR ATTRIBUTE ON DEFINITION SHOULD BE OF TYPE NCNAMES, TOTALLY REWORKED EXAMPLE

```
  <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>
<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 26.8) and can be produced e.g. by **C**Point (see Section 26.14).

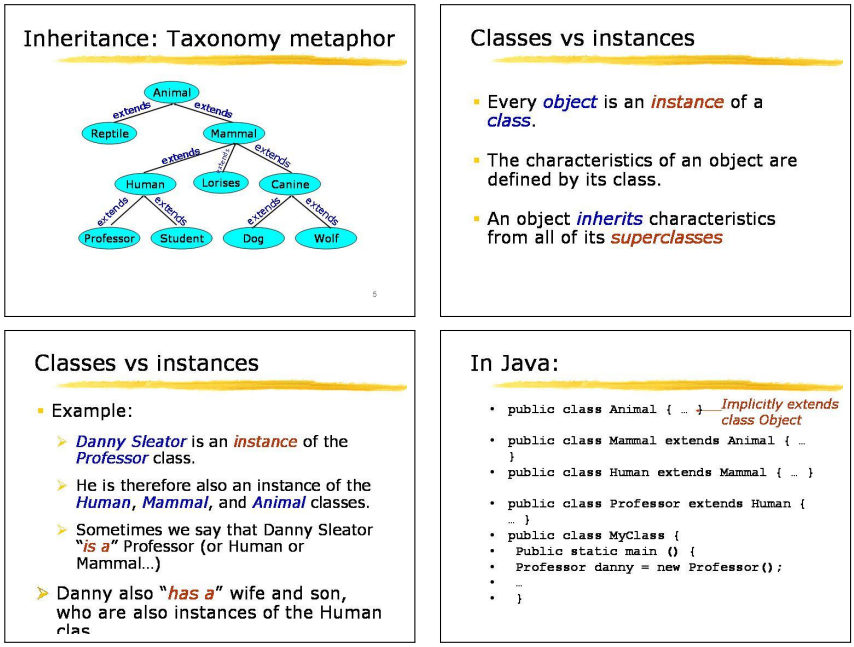


Fig. 8.1. Three slides from 15-211

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 26.4) or knowledge presentation system like ActiveMath (see Section 26.8) 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>Humans are Animals.</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.

BErr(15)

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>Every <phrase style="font-style:italic;color:blue">object</phrase>
      is an <phrase style="font-style:italic; color:red">instance</phrase>
      of a <phrase style="font-style:italic; color:blue">class</phrase>.
    </CMP>
  </axiom>

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

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

EErr(15)

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. 155). 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,

[15] Erratum: `for` attribute on `axiom` should be of type `NCNames`

[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]. BErr(16)

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><phrase style="font-style:italic;color:blue">Danny Sleator</phrase>
      is an <phrase style="font-style:italic; color:red">instance</phrase>
      of the <phrase style="font-style:italic; color:blue">Professor</phrase>
      class.
    </CMP>
  </axiom>

  <assertion xml:id="dannys-classes" type="theorem">
    <CMP>He is therefore also an instance of the
      <phrase style="font-style:italic; color:blue">Human</phrase>,
      <phrase style="font-style:italic; color:blue">Mammal</phrase>,
      <phrase style="font-style:italic; color:blue">Animal</phrase> classes.
    </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>Sometimes we say that Danny Sleator
       &#x201C;<phrase style="font-style:italic;color:red">is a</phrase>&#x201D;
       Professor (or Human or Mammal&#x2026;)
     </CMP>
  </definition>
</theory>
```

EErr(16)

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 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).

[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

[16] Erratum: `for` attribute on `definition` should be of type `NCNames`

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.

Err(17)

The $[\text{just-by}]_r^{17}$ `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.

BErr(18)

Listing 8.4. An Alternative Representation Using `example` Elements

```
1  ...
<example xml:id="danny-mammal" type="for" assertion="#dannys-mammal-thm"
         for="#cvi.instance #cvi.class #animal-taxonomy.mammal">
  <CMP>Danny Sleator is an instance of the
    <phrase style="font-style:italic; color:blue">Mammal</phrase> class.
6 </CMP>
  <OMOBJ><OMS cd="cvi-examples" name="danny"/></OMOBJ>
</example>

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

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

[17] Erratum! forgot to thread through attribute renaming (original text was: "proofs")

[18] Erratum: The attribute on the `assertion` element should be `just-by`, not `proofs`. We were also missing some fragment identifiers.

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

EErr(18)

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>
3   <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>
8   <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 `ref` elements, whose `xref` attribute points 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 `ref` element 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. 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>
      <omlet data="#tax-image" style="width:540;height:366"
             action="display" show="embed"/>
    </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)">
    <ref style="position:30% 10%" xml:id="obj" xref="slide1_content.omdoc#ax1"/>
    <ref style="position:55% 10%" xml:id="class" xref="slide1_content.omdoc#ax2"/>
    <ref 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>Example:</CMP></omtext>
    <omgroup type="itemize" style="list-style-type:url(triangle.gif)">
      <ref style="position:400% 15%"
           xml:id="danny" xref="slide1_content.omdoc#danny-professor"/>
      <ref style="position:55% 15%"
           xml:id="inst" xref="slide1_content.omdoc#dannys-classes"/>
      <ref 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>
        Danny also &#x201C;<phrase style="font-style:italic;color:red">has
        a</phrase>&#x201D; wife and son, who are also instances of the Human class
      </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>Implicitly extends class object</CMP>
```

```
    </omtext>
    <omtext xml:id="slide-850.t2">
      <CMP><omlet data="#cvic-code1" action="display" show="embed"/></CMP>
    </omtext>
    <omtext xml:id="slide-850.t3">
      <CMP><omlet data="#cvic-code2" action="display" show="embed"/></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 $\mathcal{N}$.

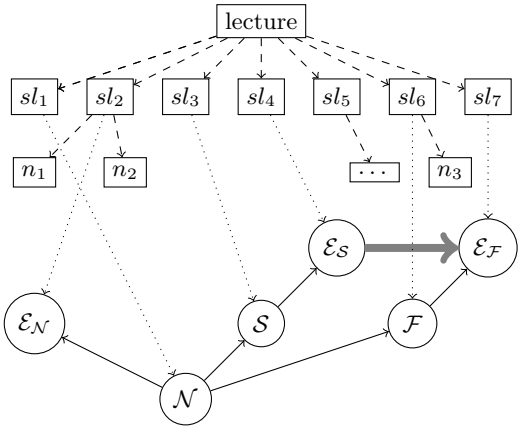


Fig. 8.2. An Introduction of a Concept via a Straw-Man Theory

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 `ref` elements 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 [MBG+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 26.8 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 service*s), 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 [GHMN03] (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
Content−Type: application/soap+xml
Content−Length: 990

<?xml version='1.0'?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap−envelope">
  <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>
      <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>
      </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>
      <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>
        <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://www.mathweb.org/omdoc"
            theory="http://mbase.mathweb.org:8080/RPC2#lovelife">
14        <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"/>
19              <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"/>
24                </om:OMA>
              </om:OMBIND>
            </om:OMOBJ>
          </omdoc:FMP>
        </omdoc:assertion>
29      <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 26.4 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://www.mathweb.org/omdoc">
      <symbol name="peter"/><symbol name="mary"/>
      <symbol name="love"/><symbol name="hate"/>
      <axiom xml:id="opposite">
        <CMP>Peter hates everybody Mary loves</CMP>
        <FMP>∀x.loves(mary, x) ⇒ hates(peter, x)</FMP>
      </axiom>
      <axiom xml:id="mary−loves−peter">
        <CMP>Mary loves Peter</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://www.mathweb.org/omdoc">
      <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 1.2 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

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 Figure 10.1 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 first four modules in Figure 10.1 are required (mathematical documents without them do not really make sense), the other ones 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).

10.1 The OMDoc Namespaces

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

In an OMDoc document, the OMDoc namespace must be specified either using a namespace declaration of the form `xmlns="http://www.mathweb.org/omdoc"` 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://www.mathweb.org/omdoc"` on some element dominating the OMDoc element in question (see Section 1.3 for an introduction). OMDoc also uses the following namespaces[1]:

Format	namespace URI	see
Dublin Core	`http://purl.org/dc/elements/1.1/`	Sections 12.1 and 12.2
Creative Commons	`http://creativecommons.org/ns`	Section 12.3
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

Thus a typical document root of an OMDoc document looks as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<omdoc xml:id="test.omdoc" version="1.2"
  xmlns="http://www.mathweb.org/omdoc"
  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>
```

10.2 Common Attributes in OMDoc

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.

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,

[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.

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.

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) [Bos98], 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 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 [Bos98] 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>Here comes something
      <phrase style="font-weight:bold;color:green" class="emphasize">stylish</phrase>!
    </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 List-

[2] The treatment of the CSS attributes has changed from OMDoc 1.1, see the discussion on page 338.

ing 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 [Cla99] 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 [Bos98]. 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 `phrase` element. As a consequence, the word "stylish" would appear in green, bold italics.

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

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			
DC	Dublin Core Metadata	yes	Sections 12.1 and 12.2
*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 12.3
Licenses for text use			
RT	Rich Text Structure	no	Section 14.6
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			
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 18.5
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, . . .)			
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			
QUIZ	Infrastructure for Assessments	no	Chapter 21
Markup for exercises integrated into the OMDoc *document model*			

Fig. 10.1. The OMDoc Modules

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 text-

book 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 26.8) 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 10.2.

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 `1.2` 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 10.1), 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 11.4. Both have an optional `type` attribute; we will discuss its values and meaning in Section 11.4.

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]$^{19}_{r}$ 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. Err(19) BErr(20) BErr(21)

Element	Attributes		D	Content
	Required	Optional	C	
`omdoc`	`version, xmlns`	`xml:id, type, class, style, version, modules, theory`	+	(⟪*top-level*⟫)*
`omgroup`		`xml:id, modules, type, class, style, theory`	+	(⟪*top-level*⟫)*
`metadata`		`xml:id, inherits, class, style`	–	⟪*MDelt*⟫*
`ref`	`xref`	`xml:id, type, class, style`	–	
`ignore`		`type, comment`	–	`ANY`
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.

EErr(21) EErr(20)

11.2 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-

[19] Erratum! Typo: "Backus Naur form" instead of "Bachus Naur Form" (original text was: "Bachus")

[20] Erratum: ref does permit an xml:id attribute (and this should remain, as that is important for talking about refs from an RDF point of view)

[21] Erratum: omdoc and omgroup can have an optional theory attribute as well

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 $[]^{22}_{d}$ [Resource Description Framework]$^{23}_{r}$ (RDF [LS99]). We use one of the best-known metadata schemata for documents – the *Dublin Core* (cf. Sections 12.1 and 12.2). 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.

Err(22)
Err(23)

`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 12.3), 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.

11.3 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 can-

[22] Erratum! RDF as a general data model is independent from XML; RDF/XML is just one of its possible serializations. (deleted "XML")

[23] Erratum! correct name (original text was: "resource description format")

not 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 14.3) 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>This does not make sense yet, in particular, the equation
      <OMOBJ>...</OMOBJ> cannot be true, think of <OMOBJ>...</OMOBJ>
    </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>The proof is trivial</CMP>
    <CMP xml:lang="de">Der Beweis ist extrem schwer</CMP>
  </assertion>
</ignore>
```

For another use of the `ignore` element, see Figure 11.2 in Section 11.5.

11.4 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 10.2).

`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 12.1 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.

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.

11.5 Sharing and Referring to Document Parts

ref

As the document structure need not be a tree in hypertext documents, `omgroup` elements also allow empty `ref` elements whose `xref` attribute can be used to reference OMDoc elements defined elsewhere. The optional `xml:id`

[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 14.6 for details. Furthermore, Ver-

(its value must be document-unique) attribute identifies it and can be used for building reference labels for the included parts. Even though this attribute is optional, it is highly recommended to supply it. The `type` attribute can be used to describe the reference type. Currently OMDoc supports two values: `include` (the default) for in-text replacement and `cite` for a proper reference. The first kind of reference requires the OMDoc application to process the document as if the `ref` element were replaced with the OMDoc fragment specified in the `xref`. The processing of the type `cite` is application specific. It is recommended to generate an appropriate label and (optionally) supply a hyper-reference. There may be more supported values for `type` in time.

Let R be a `ref` element of type `include`. We call the element the URI in the `xref` points to its **target** unless it is an `omdoc` element; in this case, the target is an `omgroup` element which has the same children as the original `omdoc` element[3].

We call the process of replacing a `ref` element by its target in a document **`ref`-reduction**, and the document resulting from the process of systematically and recursively reducing all the `ref` elements 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 `ref` elements 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.

The `ref` elements also allow to "flatten" the tree structure in a document into a list of leaves and relation declarations (see Figure 11.2 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.

---

sion 1.1 of OMDoc allowed the value `narrative`, which was synonymous with `sequence`.

[3] This transformation is necessary, since OMDoc does not allow to nest `omdoc` elements, which would be the case if we allowed verbatim replacement for `omdoc` elements. As we have stated above, the `omdoc` has an implicit `omgroup` element, and thus behaves like one.

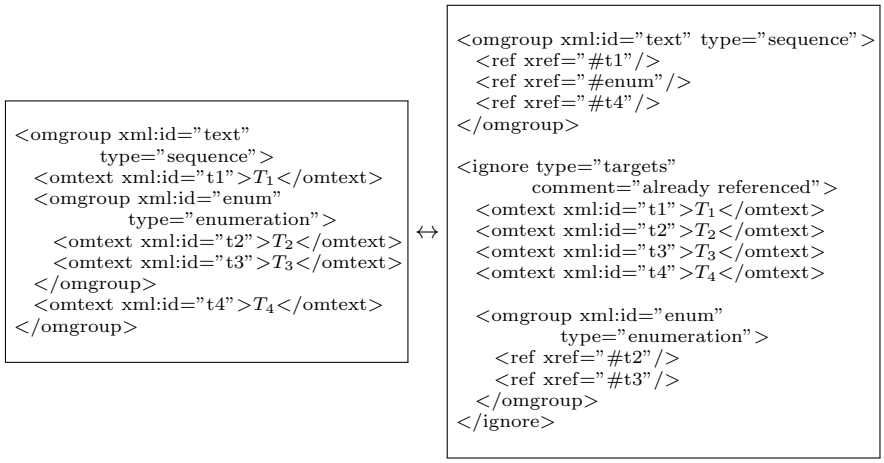


Fig. 11.2. Flattening a Tree Structure

While the OMDoc approach to specifying document structure is a much more flexible (database-like) approach to representing structured documents[4] than the tree model, it puts a much heavier load on a system for presenting the text to humans. In essence the presentation system must be able to recover the left representation from the right one in Figure 11.2. 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 cross-references specified in `ref` elements. In particular, the text fragment corresponding to the element with `xml:id="text"` in the right OMDoc of Figure 11.2 is just the one on the left.

In Section 10.2 we have introduced the CSS attributes `style` and `class`, which are present on all OMDoc elements. In the case of the `ref` element, 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 `ref` element, and then determine the values of the CSS properties of the `ref` element by inheritance.

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

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. BErr(24)

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

Fig. 12.1. Dublin Core Metadata in OMDoc

EErr(24)

[24] Erratum: The content Model for `dc:creator` and `cd:contributor` is simple text

In the following we will describe the variant of Dublin Core metadata elements used in OMDoc[1]. 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 [MAR03]) on the authorship elements and with a rights management system based on the Creative Commons Initiative.

12.1 The Dublin Core Elements (Module DC)

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.

`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.

dc:creator

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 12.2.

`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

dc:contributor

[1] 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

attributes plus a `dc:lang` attribute that specifies the target language in case the contribution is a translation.

`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:subject`

`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 `CMP`s (then it can be used as an abstract). `dc:description`

`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:publisher`

`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 [BM01] 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 $[\texttt{action}]^{25}_{r}$ 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:date` Err(25)

`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, arti-*

[25] Erratum! wrong attribute name (original text was: "`dc`")

cles, 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` 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.

`dc:rights`

OMDoc supplies specialized elements for the Creative Commons licenses to support the sharing of mathematical content. We will discuss them in Section 12.3.

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 appro-

priate for the mathematical content of OMDoc, which is largely independent of time and geography.

12.2 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 [MAR03].

`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 12.1 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 12.1. 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 12.2 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 12.2. A Formalization with Scientific Antecedent (`ant`)

```
<omdoc xml:id="NNat" version="1.2" 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>
```

12.3 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 `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.

`cc:license`

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 [Crea] 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[2]. 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. 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`

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

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 [Creb] which defines the meaning, and provides an RDF [LS99] based implementation. As we have discussed in Section 11.2, 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 12.3 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 12.3. 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>
```

12.4 Inheritance of Metadata

The `metadata` elements can be added to many of the OMDoc elements, including grouping elements that can contain others that contain `metadata`. To avoid duplication, OMDoc assumes a priority-union semantics for the Dublin Core elements `dc:creator`, `dc:contributor`, `dc:date`, `dc:type`, `dc:format`, `dc:source`, `dc:language`, and `dc:rights`. A Dublin Core element, e.g. `dc:creator` that is missing in lower `metadata` declaration (i.e. there is no element of the same name) is inherited from the upper ones. So in Figure 12.3, the two boxes are equivalent, since the metadata in theory `th1` and in definition `d1` is inherited from the main declaration in the top-level `omdoc` element. If there is a metadata element of the same name present, the closer one takes precedence.

BErr(26)

EErr(26)

[26] Erratum: `for` attribute on `definition` should be of type `NCNames`

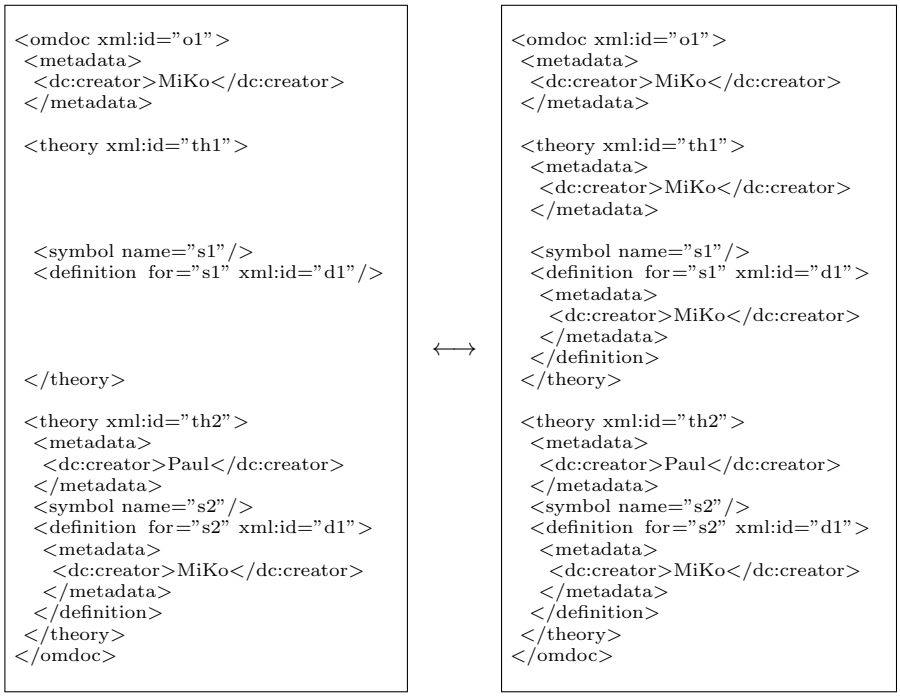


Fig. 12.3. Inheritance of Metadata

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 [BCC+04] and the MathML 2.0 recommendation (second edition) [ABC+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). BErr(27) EErr(27)

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>
```

[27] Erratum(clarification): It should be made clear that this inheritance mechanism is extended by the OMDoc format. See section **3.1** of the errata document for details

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 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:OMBVAR

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.

om:OMATTR

om:OMATP

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"/>
```

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

```
  </OMA>
</OMATTR>
```

A special application of the `om:OMATTR` element is associating non-OpenMath objects with OpenMath objects. For this, OpenMath2 allows to use 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 19.4 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 [BCC+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.

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

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 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.

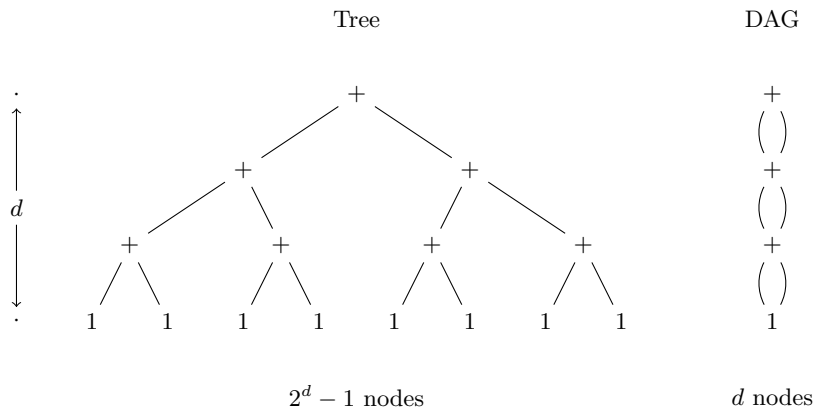


Fig. 13.3. Structure Sharing by Directed Acyclic Graphs

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

[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 OMDoc1.0 did now know structure sharing, OMDoc1.1 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 OMDoc1.2, 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.

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.

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

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

Listing 13.2. A Simple Cycle

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

$$\cfrac{1}{1+\cfrac{1}{1+\cdots}}$$

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 [ABC+03a], a maintenance release for the MathML 2.0 recommendation [ABC+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 [ABC+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).

Element	Attributes		Content
	Required	Optional	
m:math		id, xlink:href	⟪*CMel*⟫+
m:apply		id, xlink:href	m:bvar?,⟪*CMel*⟫*
m:csymbol	definitionURL	id, xlink:href	m:EMPTY
m:ci		id, xlink:href	#PCDATA
m:cn		id, xlink:href	([0-9]|,|.)(*|e([0-9]|,|.)*)?
m:bvar		id, xlink:href	m:ci|m:semantics
m:semantics		id, xlink:href, definitionURL	⟪*CMel*⟫,(m:annotation | m:annotation-xml)*
m:annotation		definitionURL, encoding	#PCDATA
m:annotation-xml		definitionURL, encoding	ANY
where ⟪*CMel*⟫ is m:apply|m:csymbol|m:ci|m:cn|m:semantics			

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

`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).

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.

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 [OMC08]. 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.

`<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

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 variables 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"/>
```

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><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"/>
              </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

[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).

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).

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 19.4 for a discussion.

13.5 Legacy Representation for Migration

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

[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 19.4 for a discussion of the OMDoc solution to this problem).

convert all mathematical objects and formulae into OpenMath or ContentMathML form. For this situation OMDoc provides the `legacy` element, which can contain arbitrary math markup[11]. The `legacy` element can occur 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:

`legacy`

- `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">
2   \lambda{x}{x}
</legacy>
```

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

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]. BErr(28) BErr(29)

Element	Attributes		D	Content
	Required	Optional	C	
CMP		xml:id, xml:lang	−	⟪*math vernacular*⟫
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)
phrase		xml:id, class, style, index, verbalizes, type	−	⟪*math vernacular*⟫
term	cd, name	cdbase, role, xml:id, class, style	−	⟪*math vernacular*⟫
omtext		xml:id, type, for, from, class, style, verbalizes	+	CMP+, FMP*

Fig. 14.1. The OMDoc Elements for Specifying Mathematical Properties

EErr(29) EErr(28)

[1] The term "mathematical vernacular" was first introduced by Nicolaas Govert de Bruijn in the 1970s (see [de 94] 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.

[28] Erratum: added the attribute xml:id to the CMP element; added the attribute from to the omtext element

[29] Erratum: added the attribute cdbase to the term element

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, such as the `phrase` and `term` elements introduced in this chapter, or link it to the document structure as the `ref` or `ignore` elements introduced above. 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 needs to talk about code fragments (see Section 20.1 and [Koha]), chemistry vernacular about chemical formulae (e.g. represented in Chemical Markup Language [ea07]).

Module	Elements	Comment	see
MOBJ	`om:OMOBJ`, `m:math`, `legacy`	mathematical Objects	p. 113
MTXT	`phrase`, `term`	phrase-level markup	below
DOC	`ref`, `ignore`	document structure	p. 95
RT	`p`, `ol`, `ul`, `dl`, `table`, `link`, `note`, `idx`	rich text structure	p. 137
EXT	`omlet`	for applets, images, ...	p. 219

Fig. 14.2. OMDoc Modules Contributing to Mathematical Vernacular

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.

BErr(30)

[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 [KK06] for a discussion in the context of "communities of practice"). Then a generalized uniqueness condition must be observed in `CMP` groups, so that systems can choose between the supplied variants.

[30] Erratum: should be "definiendum" not "definiens"

Listing 14.1. A Multilingual Group of `CMP` Elements

```
<CMP>
  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>.
</CMP>
<CMP xml:lang="de">
  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>.
</CMP>
<CMP xml:lang="fr">
  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>.
</CMP>
```

EErr(30)

Listing 14.1 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.

14.2 Formal Mathematical Properties

`FMP`

An `FMP`[4] element is the general element for representing formal mathematical content in the form of OpenMath objects. `FMP`s always appear in groups, which can differ in the value of their `logic` attribute, which specifies the logical formalism. The value of this attribute specifies the logical system used in formalizing the content. All members of the group have to formalize the same mathematical object or property, i.e. they have to be translations of each other, like siblings `CMP`s, we speak of a **multi-logic** `FMP` **group** in this

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

case. Furthermore, if an FMP group has CMP siblings, all must express the same content.

In Listing 14.2 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 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 14.2. A multi-logic FMP group for Listing 14.1.

```
<omtext xml:id="binop−def" type="definition">
  ... the content of Listing 14.1 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 to refer to them by ref elements in 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 14.3 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). For details about the phrase element see Section 14.4 below.

Listing 14.3. Representing Vernacular as an FMP Sequent

```
<CMP>If a ∈ U and a ∈ V, then a ∈ U ∩ V or
  <phrase index="moon_cheese">the moon is made of green cheese</phrase>.
</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>
```

14.3 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 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`, [`transition`]$_a^{31}$ 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.

`omtext`

Err(31)

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 12.1).

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.

[31] Erratum! `note` (added text)

`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.

BErr(32)
BErr(33)

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 `CMP`s (if that is feasible).

EErr(33)
EErr(32)

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).

14.4 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". In OMDoc we adhere to this intuition and restrict the `phrase` element to phrases in this sense. The `term` element is naturally restricted to phrases by construction. The `phrase` element is a general wrapper for sentence-level phrases that allows to mark their specific properties.

`phrase`

The `phrase` element allows the same content as the `CMP` element, so that it

[32] Erratum: omtext can also be an assumption, obligation or rule as all of these can be expressed in informal as well as formal way

[33] Erratum: And there should also be <omtext type="assertion"> for generic assertions, corresponding to the <assertion> element without a type.

can be transparently nested. It has the optional attribute `xml:id` for referencing the text fragment and the CSS attributes `style` and `class` to associate presentation information with it (see the discussion in Sections 10.2 and 19.1). 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 14.3. Furthermore, the `phrase` element allows the attribute `index` for parallel multilingual markup: Recall that sibling `CMP` elements form multilingual groups of text fragments. We can use the `phrase` element to make the correspondence relation on text fragments more fine-grained: `phrase` 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 14.7 for an example at the discourse level.

Finally, the `phrase` element can carry a `verbalizes` attribute whose value is a whitespace-separated list of URI references that act as pointers to other OMDoc elements. This has two applications: 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 14.4. Parallel Markup between Formal and Informal

```
<CMP>
  If <phrase verbalizes="#isaG">⟨G, ∘⟩ is a group</phrase>, then of course
    <phrase verbalizes="#isaM">it is a monoid</phrase> by construction.
</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.

14.5 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 25), 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>...
```

`term`

Err(34)

The `term` element has two required attributes: `cd` and `name`[, and optionally `cdbase`]$^{34}_{a}$, 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.

> 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.*

BErr(35)

EErr(35)

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.

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

[34] Erratum! also need `cdbase` for identifying (added text)

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

[35] Erratum: Should be Deffiniendum instead of deffiniens

Pending a thorough linguistic investigation we will mark up such occurrences with `definiens-applied`, for instance in BErr(36)

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.
3 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.
```

EErr(36)

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

14.6 Rich Text Structure (Module RT)

The infrastructure for mathematical vernacular introduced above assumed the `CMP` elements as atomic fragments of mathematical vernacular allowing for very little discourse-level structure below the level of `CMP`. This would be sufficient, if the `CMP` were only used for text, but as we have seen above, the `CMP` element is also used for mathematical text fragments that correspond to mathematical statements like definitions or theorems, which might have internal text structure and therefore required corresponding structural elements in OMDoc.

Element	Attributes		D	Content
p		xml:id, style, class, index, verbalizes	+	⟨*math vernacular*⟩
ol		xml:id, style, class, index, verbalizes	+	li*
ul		xml:id, style, class, index, verbalizes	+	li*
li		xml:id, style, class, index, verbalizes	+	⟨*math vernacular*⟩
dl		xml:id, style, class, index, verbalizes	+	di*
di		xml:id, style, class, index, verbalizes	+	dt*,dd*
dt		xml:id, style, class, index, verbalizes	+	⟨*math vernacular*⟩
dd		xml:id, style, class, index, verbalizes	+	⟨*math vernacular*⟩
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*⟩
table		xml:id, style, class, index, verbalizes	+	tr*
tr		xml:id, style, class, index, verbalizes	+	td*
td		xml:id, style, class, index, verbalizes	+	⟨*math vernacular*⟩
th		xml:id, style, class, index, verbalizes	+	⟨*math vernacular*⟩
link	href	xml:id, style, class, index, verbalizes	−	⟨*math vernacular*⟩
note		type, xml:id, style, class, index, verbalizes	+	⟨*math vernacular*⟩

Fig. 14.3. Rich Text Format OMDoc

In this section we will discuss the OMDoc rich text structure module RT, which introduces text structuring elements for mathematical text below the

[36] Erratum: should be "definiendum" not "definiens"

level of mathematical statements. The elements in this module are loosely patterned after elements from the XHTML specification [Gro00], and can occur as part of mathematical vernacular. Where we do not explicitly discuss the content, it is mathematical vernacular as well. The module RT provides five classes of elements, which we will show in context in Listing 14.6.

Listing 14.6. An Example of Rich Text Structure

```
<CMP>
  <p style="color:red" xml:id="p1">All <idx><idt>animals are dangerous</idt>
    <idp>dangerous</idp><idp seealso="creature">animal</idp></idx>!
    (which is a highly <phrase class="emphasis">unfounded</phrase>
    statement). In reality only some animals are, for instance:</p>
  <ul xml:id="l1">
    <li>sharks (they bite) and </li>
    <li>bees (they sting).</li>
  </ul>
  <p>If we measure danger by the number of deaths, we obtain</p>
  <table>
    <tr>          <th>Culprits</th> <th>Deaths</th> <th>Action</th></tr>
    <tr>          <td>sharks</td> <td>312</td> <td>bite</td></tr>
    <tr xml:id="bn"> <td>bees</td> <td>23</td> <td>sting</td></tr>
    <tr>          <td>cars</td>  <td>7500</td> <td>crash</td></tr>
  </table>
  <p>So, if we do the numbers <note xml:id="n1" type="ednote">check the
  numbers again</note> we see that animals are dangerous, but they are
  less so than cars but much more photogenic as we can see
  <link href="http://www.yellowpress.com/killerbee.jpg">here</link>.</p>

  <note type="footnote">From the International Journal of Bee−keeping; numbers only
  available for 2002.</note>
</CMP>
```

`p`

Paragraphs `p` elements can be used as children in a `CMP` to divide the text into paragraphs.

`ol`

`li`

Ordered Lists The `ol` element is a constructor for ordered lists, which has `li` elements as children that represent the items. These contain mathematical vernacular as content and are presented as consecutively numbered.

`ul`

Unordered Lists `ul` is the constructor for unordered or bulleted lists, the in the presentation, list items are indicated by some sort of bullet.

`dl`

`di`

`dt`

`dd`

Description Lists Finally, `dl` is a constructor for description lists, which have `di` elements as children. The `di` elements contain an optional `dt` element (description title) followed by a (possibly empty) list of `dd` elements that contain the descriptions.

`table`

`tr`

`td`

Tables To mark up simple tables we use the `table` element. Just as in XHTML, it has an arbitrary number of `tr` (table row) elements that contain `td` (table data) and `th` (table header) elements, which contain mathematical vernacular. Note that OMDoc does not support advanced formatting attributes of XHTML, but as tables are mathematical text in the module RT it does support nested tables.

`th`

`link`

Hyperlinks The `link` element is equivalent to the XHTML `a` element, and

carries a required `href`[7] attribute that points to an arbitrary resource in form of a URI reference.

Index Markup 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. 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]$^{37}_{a}$ `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).

idx idt ide Err(37) idp

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

note

All elements in the RT module carry an optional `xml:id` attribute for identification and an `index` attribute for parallel multilingual markup (e.g. Section 14.4 for an explanation and Listing 14.7 for a translation example).

Listing 14.7. Multilingual Parallel Markup

```
<omtext xml:id="animals.overview">
  <CMP>
    <p index="intro">Consider the following animals:</p>
    <ul index="animals">
      <li index="first">a tiger,</li>
      <li index="second">a dog.</li>
    </ul>
  </CMP>
  <CMP xml:lang="de">
```

[7] It is anticipated that future versions of OMDoc may use simple links from xlink [DMOT01] for such cross-referencing tasks, but at the moment we keep in style to the rest of the specification.

[37] Erratum! the index attribute should be optional (added text)

```
    <p index="intro">Betrachte die folgenden Tiere:</p>
    <ul index="animals">
      <li index="first">Ein Tiger</li>
      <li index="second">Ein Hund</li>
    </ul>
  </CMP>
</omtext>
```

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. BErr(38)

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

EErr(38)

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.

[38] Erratum: "Definiendum" and "Definiens" should switched

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 [BPSM+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]$^{39}_{a}$. Finally, the optional attribute `role` that can take the values[3] Err(39)

`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.

---

[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.

[39] Erratum! scope is deprecated (added text)

[3] The first six values come from the OpenMath2 standard. They are specified in content dictionaries; therefore OMDoc also supplies them.

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 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 $[\text{names}]_r^{40}$ 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.

Err(40)

BErr(41)

Listing 15.2. An OMDoc `axiom`

```
<axiom xml:id="mon.ax" for="monoid">
  <CMP>If (M, ∗) is a semigroup with unit e, then (M, ∗, e) is a monoid.</CMP>
```

[40] Erratum! the `for` attribute in the `axiom` element must reference symbol names (original text was: "URI references")

[41] Erratum: `for` attribute on `axiom` should be of type `NCNames`

</axiom>

EErr(41)

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** $e:t$ 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**.

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 $e:t$ 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]. BErr(42)

`type`

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 $e:t$, 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. EErr(42)

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).

[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.

[42] Erratum: examples reference wrong listings

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.[

Err(43)

]$^{43}_{a}$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.

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 `axioms`. 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`.

BErr(44)

Listing 15.3. A Simple OMDoc `definition`.

```
<symbol name="one"/>
<definition xml:id="one.def" for="one" type="simple">
 <CMP><OMOBJ><OMS cd="nat" name="one"/></OMOBJ> is the successor of
      <OMOBJ><OMS cd="nat" name="zero"/></OMOBJ>.</CMP>
 <OMOBJ>
   <OMA>
     <OMS cd="nat" name="suc"/>
     <OMS cd="nat" name="zero"/>
   </OMA>
 </OMOBJ>
</definition>
```

EErr(44)

`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.

BErr(45)

[43] Erratum! Note that this use of the `for` attribute is different from the other usages, which are URI references. (added text)

[44] Erratum: `for` attribute on `definition` should be of type `NCNames`, also corrected cd attribute.

[45] Erratum: `for` attribute on `definition` should be of type `NCNames`

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>
    There is at most one differentiable function that solves the
    differential equation in definition <ref type="cite" xref="#exp-def"/>.
  </CMP>
</assertion>
<assertion xml:id="exp-exists">
  <CMP>
    The differential equation in <ref type="cite" xref="#exp-def"/> is solvable.
  </CMP>
</assertion>
```

EErr(45)

`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.

BErr(46)

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>Addition is defined by recursion on the second argument.</CMP>
  <requation>x + 0 ⇝ x</requation>
  <requation>x + s(y) ⇝ s(x + y)</requation>
</definition>
```

EErr(46)

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`

[46] ERRATUM: FOR ATTRIBUTE ON DEFINITION SHOULD BE OF TYPE NCNAMES

`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. Since 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.

BErr(47)

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

EErr(47)

15.3.1 Assertions

`assertion`

OMDoc uses the `assertion` element for all statements (proven or not) about

[47] Erratum: deleted spurious `for` attribute on the `assertion` element, `alternative` should have the same content as definition

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 op-

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

tional `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).

Listing 15.6. An OMDoc Assertion About Semigroups

```
<assertion xml:id="ida.c6s1p4.l1" type="lemma">
  <CMP> A semigroup has at most one unit.</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.

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.

BErr(48)

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">
```

[48] Erratum: `for` attribute on `definition` should be of type `NCNames`

`status`	`just-by` points to
`tautology` *All $\mathcal{T}$-interpretations satisfy $\mathcal{A}$ and some $\mathcal{C}_i$*	Proof of $\mathcal{F}$
`tautologous-conclusion` *All $\mathcal{T}$-interpretations satisfy some $\mathcal{C}_j$*	Proof of $\mathcal{F}_c$.
`equivalent` *$\mathcal{A}$ and $\mathcal{C}$ have the same $\mathcal{T}$-models (and there are some)*	Proofs of $\mathcal{F}$ and $\mathcal{F}^{-1}$
`theorem` *All $\mathcal{T}$-models of $\mathcal{A}$ (and there are some) satisfy some $\mathcal{C}_i$*	Proof of $\mathcal{F}$
`satisfiable` *Some $\mathcal{T}$-models of $\mathcal{A}$ (and there are some) satisfy some $\mathcal{C}_i$*	Model of $\mathcal{A}$ and some $\mathcal{C}_i$
`contradictory-axioms` *There are no $\mathcal{T}$-models of $\mathcal{A}$*	Refutation of $\mathcal{A}$
`no-consequence` *Some $\mathcal{T}$-models of $\mathcal{A}$ (and there are some) satisfy some $\mathcal{C}_i$, some satisfy $\overline{\mathcal{C}}$*	$\mathcal{T}$-model of $\mathcal{A}$ and some $\mathcal{C}_i$, $\mathcal{T}$-model of $\mathcal{A} \cup \overline{\mathcal{C}}$.
`counter-satisfiable` *Some $\mathcal{T}$-models of $\mathcal{A}$ (and there are some) satisfy $\overline{\mathcal{C}}$*	Model of $\mathcal{A} \cup \overline{\mathcal{C}}$
`counter-theorem` *All $\mathcal{T}$-models of $\mathcal{A}$ (and there are some) satisfy $\overline{\mathcal{C}}$*	Proof of $\overline{\mathcal{C}}$ from $\mathcal{A}$
`counter-equivalent` *$\mathcal{A}$ and $\overline{\mathcal{C}}$ have the same $\mathcal{T}$-models (and there are some)*	Proof of $\overline{\mathcal{C}}$ from $\mathcal{A}$ and proof of $\mathcal{A}$ from $\overline{\mathcal{C}}$
`unsatisfiable-conclusion` *All $\mathcal{T}$-interpretations satisfy $\overline{\mathcal{C}}$*	Proof of $\overline{\mathcal{C}}$
`unsatisfiable` *All $\mathcal{T}$-interpretations satisfy $\mathcal{A}$ and $\overline{\mathcal{C}}$*	Proof of $\neg\mathcal{F}$

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 $\mathcal{T}$

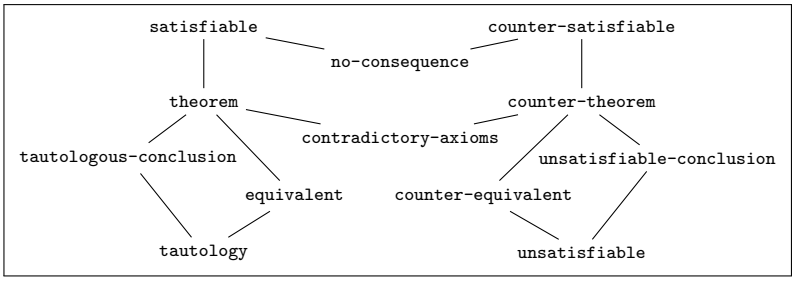


Fig. 15.7. Relations of Assertion States

```
  <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>
```

EErr(48)

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 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]$^{49}_{r}$. 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.

`alternative`

Err(49)

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,

[49] Erratum! fixed the target of the for attribute (original text was: "primary definition or assertion")

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 counter-examples for conjectures. Therefore examples are given status as primary objects in OMDoc. Conceptually, we model an example $\mathcal{E}$ as a pair $(\mathcal{W}, \mathbf{A})$, where $\mathcal{W} = (\mathcal{W}_1, \ldots, \mathcal{W}_n)$ is an n-tuple of mathematical objects 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`,]$^{50}_{a}$[`alternative`]$^{51}_{a}$, or `assertion` elements. Err(50)

`type` specifying the aspect, the value is one of `for` or `against` Err(51)

`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.

[50] Erratum! added the `axiom` element to the list; cf. discussion on `omdoc-dev` on May 16, 2008 (added text)

[51] Erratum! added the `alternative` element to the list (added text)

`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.

BErr(52)

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>(A*, ::, ϵ) is a monoid.</CMP>
  <FMP>mon(A*, ::, ϵ)</FMP>
</assertion>
...
<example xml:id="mon.ex1" for="monoid" type="for"
         assertion="string.struct.monoid">
  <CMP>The set of strings with concatenation is a monoid.</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>Monoids are groups.</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>The set of strings with concatenation is not a group.</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>(A*, ::, ϵ) is a monoid, but there is no inverse function for it.</CMP>
</assertion>
```

EErr(52)

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

[52] Erratum: `for` attribute on `definition` should be of type `NCNames`

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: BErr(53)

Listing 15.9. A Simple-Minded Representation of **Theorem 3.12** BErr(54)

```
<assertion type="theorem" style="display=flow">
  <CMP>In a monoid M, the left unit and the right unit coincide,</CMP>
</assertion>
<definition for="unit" style="display:flow">
  <CMP>we call it the <term role="definiendum" name="unit">unit</term> of M</CMP>
</definition>
```

EErr(54)

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 taken as stand-alone, but only makes sense in 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: BErr(55)

EErr(53)

BErr(56)

[53] Erratum: `for` attribute on `definition` should be of type `NCNames`
[54] Erratum: should be "definendum" not "definiens"
[55] Erratum: `for` attribute on `definition` should be of type `NCNames`
[56] Erratum: should be "definiendum" not "definiens"

Listing 15.10. An Inline Definition

```
<assertion xml:id='unit−unique' type="theorem" >
  <CMP>In a monoid M, the left unit and the right unit coincide,
    <phrase verbalizes="#unit−def">we call it the unit of M</phrase>.</CMP>
</assertion>
<symbol name="unit"/>
<definition xml:id="unit−def" for="unit" just−by='#unit−unique'>
  <CMP>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.</CMP>
</definition>
```

EErr(56)
EErr(55)

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`

Err(57)

Theories are specified by the `theory` element in OMDoc, which has [an optional]$^{57}_{r}$ `xml:id` attribute for referencing the theory. Furthermore, the `theory` element can have the `cdbase` attribute that allows to specify the `cdbase` 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.

`tgroup`

Theories can be structured like documents e.g. into sections and the like (see Section 11.4 for a discussion) via the `tgroup` element, which behaves exactly like the `omgroup` element introduced in Section 11.4 except that it also allows theory-constitutive elements, but does not allow a `theory` attribute, since this information is already given by the dominating `theory` element.[6]

[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

[57] Erratum! the `xml:id` attribute on the `theory` element should be optional (original text was: "a required")

[6] This element has been introduced to keep OMDoc validation manageable: We cannot directly use the `omgroup` element,since there is no simple, context-free way to determine whether an `omgroup` is dominated by a `theory` 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	+	
tgroup		xml:id, modules, type, class, style	+	(⟪*top+thc*⟫)*
where ⟪*top+thc*⟫ stands for top-level (but no omgroup) 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.

The meaning of the imports element is determined by two attributes: imports

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 BErr(58)

BErr(59)

```
<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

[58] Erratum: The symbol name af should be aa

[59] Erratum: for attribute on definition should be of type NCNames

```
<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>
```

EErr(59)

EErr(58)

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[7]. For theories that are imported 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.

[7] 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`

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. BErr(60)

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>neut is a left unit for op.</CMP><FMP>∀x ∈ set.op(x, neut) = x</FMP>
  </axiom>
  <definition xml:id="setstar.def" for="setstar" type="implicit">
    <CMP>·* subtracts the unit from a set </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>For every X ∈ set there is an inverse inv(X) wrt. op.</CMP>
  </axiom>
</theory>
```

EErr(60)

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

[60] Erratum: `for` attribute on `definition` should be of type `NCNames`

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 [BCC+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
BErr(61) 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
EErr(61) `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

[61] Erratum(clarification): This specification of the inheritance mechanism is to wishy washy. See section **3.1** of the errata document for a clarification.

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 [BCC+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 [CoF04]) 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 26.20 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`. $[]_d^{62:1}$ 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`

BErr(62)

EErr(62)

Listing 16.1 introduces the sort symbols `pos-nats` (positive natural numbers) and `nats` (natural numbers) , the symbol names are given by the re-

[62] Erratum: The `for` attribute contains a URI reference according to the RelaxNG schema; the locality restriction here contradicts that and needs to be removed.

[62:1] deleted: Note that the sort ⟪*name*⟫ must be declared in a `sortdef` in the same `adt` element.

quired `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` 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`).

`argument`

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`.

`selector`

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`.

`recognizer`

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. [BC01a]). 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 [APCS01]. As a consequence, the PF module has been redesigned [AKC03] 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 [Coe05] 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. BErr(63)

Listing 17.1. An OMDoc Representation of Figure 17.1.

```
<assertion xml:id="a1">
  <CMP>There are infinitely many prime numbers.</CMP>
</assertion>
<proof xml:id="p" for="#a1">
  <omtext xml:id="intro">
    <CMP>We need to prove that the set $P$ of all prime numbers is not finite.</CMP>
  </omtext>
  <derive xml:id="d1">
    <CMP>We proceed by assuming that $P$ is finite and reaching a contradiction.</CMP>
    <method>
      <proof xml:id="p1">
        <hypothesis xml:id="h2"><CMP>Let $P$ be finite.</CMP></hypothesis>
        <derive xml:id="d3">
          <CMP>Then $P = \{p_1, \ldots, p_n\}$ for some $p_i$.</CMP>
          <method><premise xref="#h2"/></method>
        </derive>
        <symbol name="q"/>
        <definition xml:id="d4" for="q" type="informal">
          <CMP>Let $q \stackrel{def}{=} p_1 \cdots p_n + 1$</CMP>
        </definition>
        <derive xml:id="d5">
          <CMP> Since for each $p_i \in P$ we have $q > p_i$, we conclude $q \notin P$.</CMP>
        </derive>
        <omtext xml:id="c6">
          <CMP>We prove the absurdity by showing that $q$ is prime:</CMP>
        </omtext>
        <derive xml:id="d7">
          <CMP>For each $p_i \in P$ we have $q = p_i k + 1$ for some
            natural number $k$, so $p_i$ can not divide $q$;</CMP>
          <method><premise xref="#d4"/></method>
        </derive>
        <derive xml:id="d8">
          <CMP>$q$ must be prime as $P$ is the set of all prime numbers.</CMP>
          <method><premise xref="#d7"/></method>
        </derive>
        <derive xml:id="d9">
          <CMP>Thus we have contradicted our assumption</CMP>
          <method><premise xref="#d5"/><premise xref="#d8"/></method>
        </derive>
      </proof>
    </method>
  </derive>
  <derive xml:id="d10" type="conclusion">
    <CMP>This proves the assertion.</CMP>
  </derive>
</proof>
```

EErr(63)

Proofs are specified by `proof` elements in OMDoc that have the optional

`proof`

[63] Erratum: `for` attribute on `definition` should be of type `NCNames`

attributes `xml:id` and `theory` and the required attribute `for`. The `for` attribute 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).
BErr(64) 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

EErr(64)

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
`derive` 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.

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.

[64] Erratum: made the for attribute in the `proofobject` element required; added the rank attribute to the `premise` element

[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.

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.

`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.

[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 [BCF+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 [Koh05c]. 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.

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 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 [CAB+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`). BErr(65)

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>By <ref type="cite" xref="#A2"/>, <ref type="cite" xref="#A4"/>, and
      <ref type="cite" xref="#A5"/> we have z + (a + (−a)) = (z + a) + (−a).</CMP>
    <FMP>z + (a + (−a)) = (z + a) + (−a)</FMP>
    <method xref="nk−sorts.omdoc#NK−Sorts.forallstar">
      <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="forallstar">
    <metadata>
      <dc:description>Repeated Universal Instantiation></dc:description>
    </metadata>
  </symbol>
  <definition xml:id="forallstar.def" for="forallstar" type="informal">
    <CMP>Given n parameters, the inference rule ∀I* instantiates
      the first n universal quantifications in the antecedent with them.</CMP>
  </definition>
  ...
</theory>
```

EErr(65)

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.

[65] Erratum: `for` attribute on `definition` should be of type `NCNames`

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>If a ∈ U or a ∈ V, then a ∈ U ∪ V.</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> We prove the assertion by a case analysis.</CMP>
  </omtext>
  <derive xml:id="t1_p1_l1">
    <CMP>If a ∈ U, then a ∈ U ∪ V.</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>If a ∈ V, then a ∈ U ∪ V.</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> We have considered both cases, so we have a ∈ U ∪ V.</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.

[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.

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 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. BErr(66)

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>
```

EErr(66)

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

[66] ERRATUM: FOR ATTRIBUTE ON DEFINITION SHOULD BE OF TYPE NCNAMES

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 step**s, 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.

BErr(67)

Listing 17.5. A top-down Representation of the Proof in Figure 17.1.

```
1  <assertion xml:id="a1">
     <CMP>There are infinitely many prime numbers.</CMP>
   </assertion>
```

[67] Erratum: `for` attribute on `definition` should be of type `NCNames`

```
<proof for="#a1">
  <omtext xml:id="c0">
    <CMP>We need to prove that the set $P$ of all prime numbers is not finite.</CMP>
  </omtext>
  <derive xml:id="d1">
    <CMP> We proceed by assuming that $P$ is finite and reaching a contradiction.</CMP>
    <method xref="nk.omdoc#NK.by-contradiction">
      <proof>
        <hypothesis xml:id="h2"><CMP>Let $P$ be finite.</CMP></hypothesis>
        <derive xml:id="d3"><CMP>Then $P = \{p_1, \ldots, p_n\}$ for some $n$</CMP></derive>
        <symbol name="q"/>
        <definition xml:id="d4" for="q" type="informal">
          <CMP>Let $q \stackrel{def}{=} p_1 \cdots p_n + 1$</CMP>
        </definition>
        <derive xml:id="d5a">
          <CMP>For each $p_i \in P$ we have $q > p_i$</CMP>
          <method xref="#Trivial"><premise xref="#d4"/></method>
        </derive>
        <derive xml:id="d5b">
          <CMP>$q \notin P$</CMP>
          <method xref="#Trivial"><premise xref="#d5"/></method>
        </derive>
        <derive xml:id="d6">
          <CMP>We show absurdity by showing that $q$ is prime</CMP>
          <FMP>$\bot$</FMP>
          <method xref="#Contradiction">
            <premise xref="#d5b"/>
            <proof>
              <derive xml:id="d7a">
                <CMP>
                  For each $p_i \in P$ we have $q = p_i k + 1$ for a given natural number $k$.
                </CMP>
                <method xref="#By_Definition"><premise xref="#d1"/></method>
              </derive>
              <derive xml:id="d7b">
                <CMP>Each $p_i \in P$ does not divide $q$</CMP>
              </derive>
              <derive xml:id="d8">
                <CMP>$q$ is prime</CMP>
                <method xref="#Trivial">
                  <premise xref="#h2"/>
                  <premise xref="#p4"/>
                </method>
              </derive>
            </proof>
          </method>
        </derive>
      </proof>
    </method>
  </derive>
</proof>
```

EErr(67)

`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 `PLOND` 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.

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.

BErr(68)

Listing 17.6. A Theory for Propositional Natural Deduction

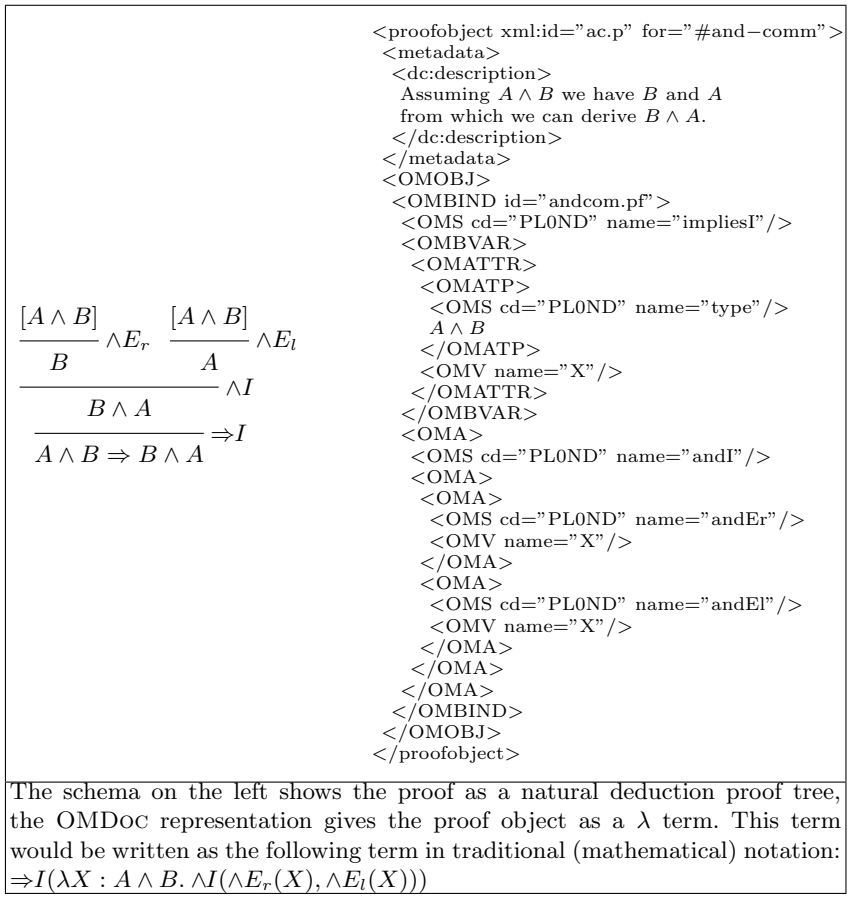
```
<theory xml:id="PL0ND">
  <metadata>
    <dc:description>The Natural Deduction Calculus for Propositional Logic</dc:description>
  </metadata>
  ...
  <symbol name="andI">
    <metadata><dc:subject>Conjunction Introduction</dc:subject></metadata>
    <type system="prop-as-types">A → B → (A ∧ B)</type>
  </symbol>

  <definition xml:id="andI.def" for="andi">
    <CMP>Conjunction introduction, if we can derive A and B,
      then we can conclude A ∧ B.</CMP>
  </definition>
  ...
</theory>
```

EErr(68)

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>
```

[68] Erratum: `for` attribute on `definition` should be of type `NCNames`



$$\dfrac{\dfrac{\dfrac{[A \wedge B]}{B}\wedge E_r \quad \dfrac{[A \wedge B]}{A}\wedge E_l}{B \wedge A}\wedge I}{A \wedge B \Rightarrow B \wedge A}\Rightarrow I$$

```
<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

Like `proofs`, `proofobjects` 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 (Modules CTH and DG)

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 "theory inclusions", 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. BErr(69) EErr(69)

18.1 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

[69] Erratum: changed the order of type and hiding attributes in the `morphism` element; removed the consistency and consistency-just attributes from the `morphism`, `inclusion`, `theory-inclusion`, and `axiom-inclusion` elements; changed the contents of the `theory-inclusion` element to (morphism?, obligation*); changed the contents of the `morphism` element to (requation+, measure?, ordering?); added the element `obligation`

Element	Attributes		D	Content
	Required	Optional	C	
theory		xml:id, class, style	+	((⟪top-level⟫ | imports | inclusion)*
imports	from	xml:id, type, class, style, conservativity, conservativity-just	+	morphism?
morphism		xml:id, base, class, style, hiding, type, consistency, exhaustivity	−	requation+, measure?, ordering?
inclusion	via	xml:id	−	EMPTY
theory-inclusion	from, to	xml:id, class, style	+	morphism?, obligation*
axiom-inclusion	from, to	xml:id, class, style	+	morphism?, obligation*
obligation	induced-by-assertion	xml:id	−	EMPTY

Fig. 18.1. Complex Theories in OMDoc

unit 0 and inverse operation $-$ and $(R^*, *)$ is a monoid with unit 1 and base set $R^* := \{r \in R | r \neq 0\}$. 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).

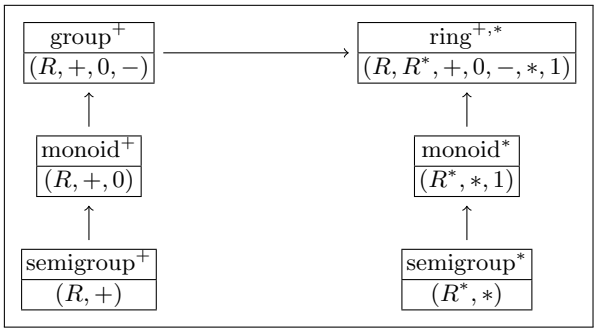


Fig. 18.2. A Theory of Rings via Simple Inheritance

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

[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.

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].

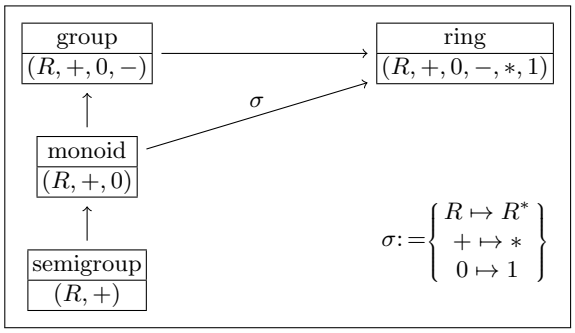


Fig. 18.3. A Theory of Rings via Morphisms

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 $\sigma: \mathcal{S} \to \mathcal{T}$ 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 $\sigma(\forall x.x + 0 = x) = \forall x.x * 1 = x$.

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`.

`morphism`

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

[2] On any monoid $M = (S, \circ, e)$, we have the $\cdot^*$ operator, which converts a set $S \subseteq M$ in to $S^* := \{r \in S \,|\, r \neq e\}$

[3] which may in turn be inherited from other theories

```
<theory xml:id="ring">
  <symbol name="times"/><symbol name="one"/>
  <imports xml:id="add.import" from="#group" type="global"/>
  <imports xml:id="mult.import" from="#monoid" type="global">
    <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><OMOBJ><OMS cd="semigroup" name="op"/></OMOBJ> distributes over
      <OMOBJ><OMS cd="ring" name="times"/></OMOBJ>
    </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 [CoF04, 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.]$^{70}_{a}$ 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

Err(70)

[70] Erratum! noted special case (added text)

— 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 `theory-inclusion`, `axiom-inclusion`, and `imports` 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 theory inclusion]$_r^{71}$. Err(71)

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 Postulated Theory Inclusions

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 **theory inclusion** 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**

[71] Erratum! Clarified wording (original text was: "`globals="...#`⟨*ref1*⟩`#`⟨*ref2*⟩`..."` 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")

[4] Mathematical objects that can be represented using the only symbols of the source theory $\mathcal{S}$.

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 $\sigma: \mathcal{S} \to \mathcal{T}$, 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 inclusion**s), we have to establish the theory inclusion property by 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.

Err(72) Err(73) Err(74) Err(75)

We can specify conservativity of a theory inclusion via the `conservativity`[attribute]$^{72}_{a}$. The values [`conservative`]$^{73}_{r}$ and [`definitional`]$^{74}_{r}$ are used for the two cases discussed above. There is a third value: [`monomorphism`]$^{75}_{r}$, which

[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.

[72] ERRATUM! added missing word (added text)

[73] ERRATUM! Fixed value of the `conservativity` attribute (original text was: "`conservative`")

[74] ERRATUM! Fixed value of the `conservativity` attribute (original text was: "`conservative`")

we will not explain here, but refer the reader to [MAH06].

OMDoc implements the concept of postulated theory inclusions in the top-level `theory-inclusion` 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.

`theory-inclusion`

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 `theory-inclusion`) is provable in the target theory. Note that a `theory-inclusion` element must contain `obligation` elements for all theory-constitutive elements (inherited or local) of the source theory to be correct.

`obligation`

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>
<theory−inclusion 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"/>
</theory−inclusion>
```

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

[75] Erratum! Fixed value of the `conservativity` attribute (original text was: "`conservative`")

posited as theory inclusions, so checking these conditions is a purely structural matter, and comes into the realm of OMDoc's structural methods.

`inclusion`

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.

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`.

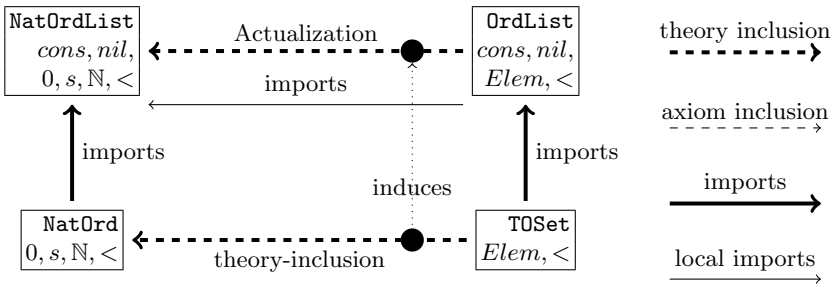


Fig. 18.4. A Structured Specification of Lists (of Natural Numbers)

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 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.

[6] This example is covered in detail in Chapter 6.

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.

Furthermore, the CTH module introduces the `axiom-inclusion` element for local theory inclusions. This has the same attributes as `theory-inclusion`: `from` to specify source theory, `to` for the target theory. It also allows `obligation` elements as children.

`axiom-inclusion`

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. [BCD+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 [**?**] 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 feed on these structures without committing to a particular formalization or foundation of mathematics.

18.5 Development Graphs (Module DG)

The OMDoc module DG for **development graph**s complements module CTH with high-level justifications for the theory inclusions. Concretely, the module provides an infrastructure for dealing efficiently with the proof obligations induced by theory inclusions and forms the basis for a management of theory change. We anticipate that the elements introduced in this chapter will largely be hidden from the casual user of mathematical software systems, but will form the basis for high-level document- and mathematical knowledge management services.

[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.

18.5.1 Introduction

As we have seen in the example in Listing 18.2, the burden of specifying an `obligation` element for each theory-constitutive element of the source theory can make the establishment of a theory inclusion quite cumbersome — theories high up in inheritance hierarchies can have a lot (often hundreds) of inherited, theory-constitutive statements. Even more problematically, such obligations are a source of redundancy and non-local dependencies, since many of the theory-constitutive elements are actually inherited from other theories.

Consider for instance the situation in Figure 18.5, where we are interested in the top theory inclusion Γ. On the basis of theories $\mathcal{T}_1$ and $\mathcal{T}_2$, theory $\mathcal{C}_1$ is built up via theories $\mathcal{A}_1$ and $\mathcal{B}_1$. Similarly, theory $\mathcal{C}_2$ is built up via $\mathcal{A}_2$ and $\mathcal{B}_2$ (in the latter, we have a non-trivial non-trivial morphism σ). Let us assume for the sake of this argument that for $\mathcal{X}_i \in \{\mathcal{A}, \mathcal{B}, \mathcal{C}\}$ theories $\mathcal{X}_1$ and $\mathcal{X}_2$ are so similar that axiom inclusions (they are indicated by thin dashed arrows in Figure 18.5 and have the formula-mappings α, β, and γ) are easy to prove[8].

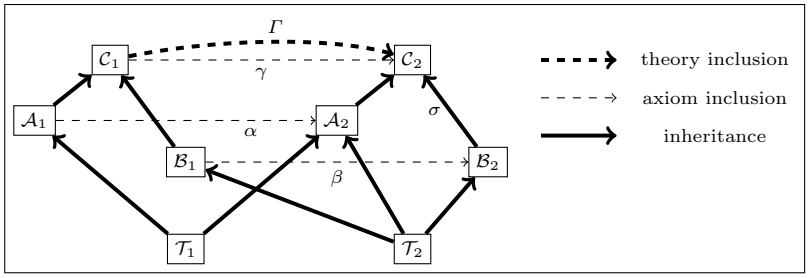


Fig. 18.5. A Development Graph with Theory Inclusions

To justify Γ, we must prove that the Γ-translations of all the theory-constitutive statements of $\mathcal{C}_1$ are provable in $\mathcal{C}_2$. So let statement $\mathbf{B}$ be theory-constitutive for $\mathcal{C}_1$, say that it is local in $\mathcal{B}_1$, then we already know that $\beta(\mathbf{B})$ is provable in $\mathcal{B}_2$ since β is an axiom inclusion. Moreover, we know that $\sigma(\beta(\mathbf{B}))$ is provable in $\mathcal{C}_2$, since σ is a (definitional, global) theory inclusion. So, if we have $\Gamma = \sigma \circ \beta$, then we are done for $\mathbf{B}$ and in fact for all local statements of $\mathcal{B}_1$, since the argument is independent of $\mathbf{B}$. Thus, we have established the existence of an axiom inclusion from $\mathcal{B}_1$ to $\mathcal{C}_2$ simply by finding suitable inclusions and checking translation compatibility.

We will call a situation, where a theory $\mathcal{T}$ can be reached by an axiom inclusion with a subsequent chain of theory inclusions a **local chain** (with

[8] A common source of situations like this is where the $\mathcal{X}_2$ are variants of the $\mathcal{X}_1$ theories. Here we might be interested whether $\mathcal{C}_2$ still proves the same theories (and often also in the converse theory inclusion Γ^{-1} that would prove that the variants are equivalent).

morphism $\tau := \sigma_n \circ \cdots \circ \sigma_1 \circ \sigma$), if $\mathcal{S} \xrightarrow{\sigma} \mathcal{T}_1$ is an axiom inclusion or (local theory import) and $\mathcal{T}_i \xrightarrow{\sigma_i} \mathcal{T}_{i+1}$ are theory inclusions (or local theory import).

$$\mathcal{S} \overset{\tau = \sigma_n \circ \cdots \circ \sigma_1 \circ \sigma}{\overset{\sigma}{\dashrightarrow}} \mathcal{T}_1 \overset{\sigma_1}{\dashrightarrow} \mathcal{T}_2 \overset{\sigma_2}{\dashrightarrow} \cdots \overset{\sigma_{n-1}}{\dashrightarrow} \mathcal{T}_n \overset{\sigma_n}{\dashrightarrow} \mathcal{T}$$

Note that by an argument like the one for **B** above, a local chain justifies an axiom inclusion from $\mathcal{S}$ into $\mathcal{T}$: all the τ-translations of the local theory-constitutive statements in $\mathcal{S}$ are provable in $\mathcal{T}$.

In our example in Figure 18.5 — given the obvious compatibility assumptions on the morphisms which we have not marked in the figure, — we can justify four new axiom inclusions from the theories $\mathcal{T}_1$, $\mathcal{T}_2$, $\mathcal{A}_1$, and $\mathcal{B}_1$ into $\mathcal{C}_2$ by the following local chains[9].

$$\mathcal{T}_2 \longrightarrow \mathcal{B}_2 \xrightarrow{\sigma} \mathcal{C}_2 \qquad \mathcal{B}_1 \overset{\beta}{\dashrightarrow} \mathcal{B}_2 \xrightarrow{\sigma} \mathcal{C}_2$$

$$\mathcal{T}_1 \longrightarrow \mathcal{A}_2 \longrightarrow \mathcal{C}_2 \qquad \mathcal{A}_1 \overset{\alpha}{\dashrightarrow} \mathcal{A}_2 \longrightarrow \mathcal{C}_2$$

Thus, for each theory $\mathcal{X}$ that $\mathcal{C}_1$ inherits from, there is an axiom inclusion into $\mathcal{C}_2$. So for any theory-constitutive statement in $\mathcal{C}_1$ (it must be local in one of the $\mathcal{X}$) we know that it is provable in $\mathcal{C}_2$; in other words Γ is a theory inclusion if it is compatible with the morphisms of these axiom inclusions. We have depicted the situation in Figure 18.6.

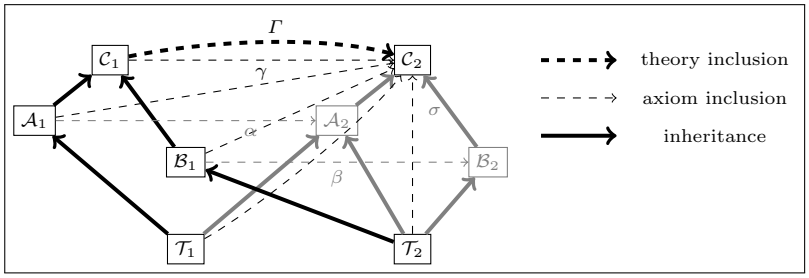


Fig. 18.6. A Decomposition for the theory inclusion Γ

We call a situation where we have a formula mapping $\mathcal{S} \xrightarrow{\sigma} \mathcal{T}$, and an axiom inclusion $\mathcal{X} \xrightarrow{\sigma_{\mathcal{X}}} \mathcal{T}$ for every theory $\mathcal{X}$ that $\mathcal{S}$ inherits from a **decomposition** for σ, if the $\sigma_{\mathcal{X}}$ and σ are compatible. As we have seen in the example above, a decomposition for σ can be used to justify that σ a theory inclusion: all theory-constitutive elements in $\mathcal{S}$ are local in itself or one of the theories $\mathcal{X}$

[9] Note for the leftmost two chains use the fact that theory inclusions (in our case definitional ones) are also axiom inclusions by definition.

it inherits from. So if we have axiom inclusions from all of these to $\mathcal{T}$, then all obligations induced by them are justified and σ is indeed a theory inclusion.

18.5.2 An OMDoc Infrastructure for Development Graphs (Module DG)

decomposition

The DG module provides the `decomposition` element to model justification by decomposition situations. This empty element can occur at top-level or inside a `theory-inclusion` element.

The `decomposition` element can occur as a child to a `theory-inclusion` element and carries the required attribute `links` that contains a whitespace-separated list of URI references to the `axiom-` and `theory-inclusion` elements that make up the decomposition situation justifying the parent `theory-inclusion` element. Note that the order of references in `links` is irrelevant. If the `decomposition` appears on top-level, then the optional `for` attribute must be used to point to the `theory-inclusion` it justifies. In this situation the `decomposition` element behaves towards a `theory-inclusion` much like a `proof` for an `assertion`. BErr(76)

Element	Attributes		D	Content
	Required	Optional	C	
decomposition	links	for	–	EMPTY
path-just	local, globals	for	–	EMPTY
theory-inclusion	from, to	xml:id, class, style	+	morphism?, (decomposition* | obligation*)
axiom-inclusion	from, to	xml:id, class, style	+	morphism?, (path-just* | obligation*)

Fig. 18.7. Development Graphs in OMDoc

EErr(76)

Furthermore module DG provides `path-just` elements as children to the `axiom-inclusion` elements to justify that this relation holds, much like a `proof` element provides a justification for an `assertion` element for some property of mathematical objects.

path-just

A `path-just` element justifies an `axiom-inclusion` by reference to other `axiom-` or `theory-inclusion`s. Local chains are encoded in the empty `path-just` element via the required attributes `local` (for the first `axiom-inclusion`) and the attribute `globals` attribute, which contains a whitespace-separated list of URI references to `theory-inclusion`s. Note that the order of the references in the `globals` matters: they are ordered in order of the path in the local

[76] Erratum: added the optional for attribute for the decomposition element; removed the by attribute from the `theory-inclusion` element; changed the contents of the `theory-inclusion` element to (morphism?, (decomposition* — obligation*))

chain, i.e if we have `globals="... #ref1 #ref2 ..."` there must be theory inclusions σ_i with `xml:id="refi"`, such that the target theory of σ_1 is the source theory of σ_2.

Like the `decomposition` element, `path-just` can appear at top-level, if it specifies the `axiom-inclusion` it justifies in the (otherwise optional) `for` attribute.

Let us now fortify our intuition by casting the situation in Listings 18.4 to 18.5.2 in OMDoc syntax. Another — more mathematical — example is carried out in detail in Chapter 7.

Listing 18.4. The OMDoc representation of the theories in Figure 18.5.

```
<theory xml:id="t1">...</theory>        <theory xml:id="t2">...</theory>

<theory xml:id="a1">                    <theory xml:id="b1">
  <imports xml:id="ima1" from="#t1"/>     <imports xml:id="imb1" from="#t2"/>
  <axiom xml:id="axa11">...</axiom>       <axiom xml:id="axb11">...</axiom>
  <axiom xml:id="axa12">...</axiom>      </theory>
</theory>

<theory xml:id="a2">                    <theory xml:id="b2">
  <imports xml:id="im1a2" from="#t1"/>     <imports xml:id="imb2" from="#t2"/>
  <imports xml:id="im2a2" from="#t2"/>
  <axiom xml:id="axa21">...</axiom>         <axiom xml:id="axb21">...</axiom>
</theory>                               </theory>

<theory xml:id="c1">                    <theory xml:id="c2">
  <imports xml:id="im1c1" from="#a1"/>     <imports xml:id="im1c2" from="#a2"/>
  <imports xml:id="im2c1" from="#b1"/>     <imports xml:id="im2c2" from="#b2"/>
  <axiom xml:id="axc11">...</axiom>        <axiom xml:id="axc21">...</axiom>
</theory>                               </theory>
```

Here we set up the theory structure with the theory inclusions given by the `imports` elements (without morphism to simplify the presentation). Note that these have `xml:id` attributes, since we need them to construct axiom- and theory inclusions later. We have also added axioms to induce proof obligations in the axiom inclusions:

Listing 18.5. The OMDoc Representation of the Inclusions in Figure 18.5.

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

<axiom-inclusion xml:id="bib" from="#b1" to="#b2">
  <obligation induced-by="#axb11" assertion="#th-axb1"/>
</axiom-inclusion>

<axiom-inclusion xml:id="cic" from="#c1" to="#c2">
  <obligation induced-by="#axc11" assertion="#th-axc1"/>
</axiom-inclusion>
```

We leave out the actual assertions that justify the `obligations` to conserve space. From the axiom inclusions, we can now build four more via path justifications:

Listing 18.6. The Induced Axiom Inclusions in Figure 18.5.

```
<axiom−inclusion xml:id="t1ic" from="#t1" to="#c2">
 <path−just local="#im1a2" globals="#im1c2"/>
</axiom−inclusion>

<axiom−inclusion xml:id="t2ic" from="#t2" to="#c2">
 <path−just local="#imb2" globals="#im2c2"/>
</axiom−inclusion>

<axiom−inclusion xml:id="aic" from="#a1" to="#c2">
 <path−just local="#aia" globals="#im1c2"/>
</axiom−inclusion>

<axiom−inclusion xml:id="bic" from="#b1" to="#c2">
 <path−just local="#bib" globals="#im2c2"/>
</axiom−inclusion>
```

Note that we could also have justified the axiom inclusion `t2ic` with two local paths: via the theory $\mathcal{A}_2$ and via $\mathcal{B}_2$ (assuming the translations work out). These alternative justifications make the development graph more robust against change; if one fails, the axiom inclusion still remains justified. Finally, we can assemble all of this information into a decomposition that justifies the theory inclusion Γ:

```
<theory−inclusion xml:id="tcic" from="#c1" to="#c2">
  <decomposition links="#t1ic #t2ic #aic #bic #cic"/>
</theory−inclusion>
```

19

Notation and Presentation (Module PRES)

As we have seen, OMDoc is concerned mainly with the content and structure of mathematical documents, and offers a complex infrastructure for dealing with that. However, mathematical texts often carry typographic conventions that cannot be determined by general principles alone. Moreover, nonstandard presentations of fragments of mathematical texts sometimes carry meanings that do not correspond to the mathematical content or structure proper. In order to accommodate this, OMDoc provides a limited functionality for embedding style information into the document. BErr(77)

Element	Attributes		Content
	Required	Optional	
omstyle	element	for, xml:id, xref, class, style	(style|xslt)*
presentation	for	xml:id, xref, fixity, role, lbrack, rbrack, separator, bracket-style, class, style, precedence, crossref-symbol	CMP*, (use | xslt | style)*
xslt	format	xml:lang, requires, xref	XSLT fragment
use	format	xml:lang, requires, fixity, precedence lbrack, rbrack, separator, element, attributes, crossref-symbol	(element | text | recurse | map | value-of)*

Fig. 19.1. The OMDoc Elements for Notation Information

EErr(77)

The normal (but of course not the only) way to generate presentation from XML documents is to use XSLT style sheets (see Chapter 25 for other applications). XSLT [XSL99] is a general transformation language for XML. XSLT programs (often called **style sheet**s) consist of a set of **templat**es (rules for the transformation of certain nodes in the XML tree). These templates are recursively applied to the input tree to produce the desired output.

[77] Erratum: added CMP* to content of `presentation` element

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

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

The OMDoc format allows to generate specialized style sheets that are tailored to the presentation of (collections of) OMDoc documents. The mechanism will be discussed in Chapter 25, here we only concern ourselves with the OMDoc primitives for representing the necessary data. In the next section, we will address the specification of style information for OMDoc elements by `omstyle` elements, and then the question of the specification of notation for mathematical symbols in `presentation` elements.

19.1 Specifying Style Information for OMDoc Elements

`omstyle`

OMDoc provides the `omstyle`[1] elements for specifying style information for OMDoc elements. An `omstyle` element has the attributes

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

`for` This optional attribute allows to further restrict the OMDoc element to a single instance. The value of this attribute is a URI reference to a single element.

`xref` This optional attribute can be used to refer to another existing `omstyle` element (in another document via a URI reference), sometimes avoiding double specification: If an `omstyle` element carries an `xref` attribute, its attributes and content is disregarded, and those of the target `omstyle` element is considered instead.

`class` This optional attribute is an additional parameter that controls the output style. Remember that all OMDoc elements that have `xml:id` attributes also carry a `class` attribute, which allows to specify different

[1] This element would perhaps be more aptly be named `omclass`, since its function is more similar to the CSS class concept, but we keep the name `omstyle` for backwards compatibility in OMDoc 1.2.

notational conventions (see Section 10.2): In the presentation of an OMDoc element only those `omstyle` elements are taken into account that have the same value in the `class` attribute.
Note that the choice of notational style is not a content-carrying feature, and should not be depended on, indeed the value of the `class` need not be respected by output routines, but can be overwritten.

In the presentation process described in Section 25.3 the information specified in the body of this element is then used to generate XSLT templates that are included then into the generated style sheets. This information is either given directly in XSLT using the `xslt` element, or in a `style` element using an OMDoc-internal equivalent of a small subset of XSLT. The latter is used if the full power of XSLT is not needed, and has the advantage that it can be transformed into the input of other formatting engines. The `xslt` and `style` elements share the following attributes:

`xslt`

`style`

`format` This required attribute specifies the output format. Its value is a set of format specifiers divided by the `|` character. We use the specifiers `TeX` for TEX and LATEX, `pmml` for Presentation-MathML, `cmml` for Content-MathML, `html` for HTML, `mathematica` for Mathematica® notebooks. Other formats can be specified at liberty. Finally, there is the pseudo format-specifier `default`, which will be taken, if no other format is defined. Note that case matters in these specifiers, so `TeX` is not the same as `tex`. Furthermore, `default` is not a regular format specifier, so it cannot appear in the disjunctions.

`xml:lang` This specifies the languages for which this notation is used. Note that it is used differently than e.g. in the `CMP` element: on `omstyle`, the attribute `xml:lang` contains a whitespace-separated list of language specifiers and it does not have a default value `en`, if the attribute is not present, this means that this element is not language-specific.

`requires` This attribute contains a URI reference that points to a `code` element that contains a code fragment that is needed to be included for the presentation engine. For instance, the body of the `omstyle` element may contain TEX macros that need to be defined. Their definitions would need to be included in the output document by the presentation style sheet before they can be used.

Listing 19.1 shows a very simple example, where a `phrase` element is used to mark a text passage as "important". Its `class` attribute is picked up by the `omstyle` element to prompt special treatment in the output.

Listing 19.1. Specifying Style Information with the `phrase` Element.

```
<CMP>
  I want to mark <phrase xml:id="w1" class="important">this important
  text</phrase> as special.<phrase class="linebreak"/>
</CMP>

<omstyle element="omdoc:phrase" class="important">
  <style format='html|pmml'><element name="em"><recurse/></element></style>
```

```
  <xslt format='TeX' xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:text>\emph{</xsl:text>
      <xsl:apply-templates/>
    <xsl:text>}</xsl:text>
  </xslt>
</omstyle>

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

19.2 A Restricted Style Language

Let us now have a closer look at the presentation-language used in `style` elements. In the first `omstyle` element in Listing 19.1 we see that the content of an `xslt` element is an XSLT fragment. Note that when referring to OMDoc elements, the XSLT must use the full qualified name (i.e. including the namespace) of the elements for the presentation to work.[2]

Element	Attributes		Content
	Required	Optional	
`style`	`format`	`xml:lang, requires, xref`	`(element | text | recurse | map | value-of)*`
`element`	`name`	`crid, cr, ns`	`(attribute | element | text | value-of | recurse | map)*`
`attribute`	`name`		`(value-of | text)*`
`text`			`(#PCDATA)`
`value-of`	`select`		`EMPTY`
`recurse`		`select`	`EMPTY`
`map`		`select`	`separator?, (element | text | recurse | map)`
`separator`			`(element | text | recurse | map)`

Fig. 19.2. The OMDoc Elements for Styling

Let us analyze the example to see the presentation in action before we define it. In the first `style` element in the `omstyle` for `linebreak` in Listing 19.1 we see that the `element` element can be used to insert an XML element into the output; in this case it is the empty HTML element `<br/>`. In the second `style` child the `text` element (it does not have attributes) allows to add arbitrary text into the output (in this case some TEX macros). In the first `omstyle` element, we see that the `element` element may be non-empty, it contains the element `recurse`, which corresponds to the directive to continue presentation generation recursively over the children of the element specified

[2] For DTD validation the XSLT fragments must be encoded using the `xsl:` namespace prefix, unless the DTD has been adapted to a different prefix by setting the appropriate parameter entity.

in the dominating `omstyle` element. The effect of this is that the content of the first `phrase` element is encased in the HTML `em` element.

Textual material can be added to the output in two ways: by copying it from the source, or supplying it in the transformation. For the latter, OMDoc supplies the `text` element (it does not have attributes), which allows to add arbitrary text (its body) into the output. For the former, we have the `value-of` element, an empty element that carries the required attribute `select`, whose value is an XPath expression. It adds the value (a string) to the XML node specified by the expression to the output.

`text`

`value-of`

The `element` element allows to generate XML elements. It has a required attribute `name`, which contains its (local) name, and the optional attribute `ns` to specify the namespace. Attributes of the resulting element can be specified by the `attribute` element: any `attribute` element adds an attribute-value pair of the form ⟪*name*⟫`="`⟪*value*⟫`"` to the output element specified by the enclosing `element` element, where the local part ⟪*name*⟫ is the value of the `name` attribute (its namespace URI given by the value the optional `ns` attribute), and ⟪*value*⟫ is either the result of presentation on the content of the `attribute` element or (iff that is empty), the value of the XPath expression in the optional `select` attribute.

`element`

`attribute`

To navigate the OMDoc structure to be transformed, we have two elements: the `recurse` allows to specify a fragment continues presentation on a sub-element, and the `map` element that maps directives over a set of sub-elements. The `recurse` element is empty, and can have the attribute `select`, which contains an XPath [CD99] expression specifying a set of OMDoc elements the presentation should continue with recursively. If this attribute is missing, presentation continues on the children as in Listing 19.1. The `map` element (see Listing 19.3 for an example) has the optional attribute `select` and contains a combination of the transformation directive elements `element`, `text`, `recurse`, `map` after an optional `separator` child. The `map` element directs the presentation engine to map the body directives[3] over the list of elements specified by the XPath expression in the `select`, between any two elements, the result of styling the body of the `separator` element is inserted between the result node sets. In Listing 19.3 the `map` element recursively styles the children of the `om:OMBVAR` element and separates them by commata. Furthermore, the `map` element can have the attributes `precedence`, `lbrack`, and `rbrack` to specify brackets (with precedence-based elision) around the result. This is useful for generating argument groups.

`recurse`

`map`

Note that this OMDoc-internalized subset of XSLT restricts the expressivity of the presentation style by leaving out the computational features of XSLT. Firstly, the infrastructure for iteration, recursion, variable declaration, ... is not present, and secondly, path expressions are restricted to pure XPath [CD99], leaving out all XSLT extensions (e.g. functions calls), again leaving us with a more declarative subset of XSLT.

[3] i.e. those elements after the `separator` element

19.3 Specifying the Notation of Symbols

In this section we discuss the problem of specifying the notation of mathematical symbols in OMDoc. The approach taken is very similar to the one for OMDoc elements presented in the previous section. The mathematical concepts and symbols introduced in an OMDoc document (by `symbol` elements or implicitly by abstract data types) often carry typographic conventions that cannot be determined by general principles alone. Therefore, these need to be specified, so that pleasing presentations can be generated.

We have already seen the use of `style` and `xslt` elements for specifying the presentation of general OMDoc elements in the last section. Here we will present yet another way to specify presentation information that is specialized to notations of mathematical symbols. The main idea is to specify the properties of mathematical symbols in relation to the representations of their children and siblings.

19.3.1 Specifying Notation via Templates

Let us build up our intuition by an example: For the notation information for the universal quantifier we would use an XSLT template like the one shown in Listing 19.2.

Listing 19.2. An XSLT Template for the Universal Quantifier

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

The XPath expression in the `match` attribute (the **template head**) specifies that this template acts as a presentation rule for `om:OMBIND` elements, where the first child is of the form `<OMS cd="quant1" name="forall"/>`. Applied to such a node, the body of the template will be executed: it will print the quantifier ∀, then the bound variables as a comma-separated list (for each of the children of `om:OMBVAR` it recursively applies XSLT templates from the style sheet), print a dot, and then recurse on the third child of the `om:OMBIND` element. Thus this template will print the OpenMath expression below as $\forall P, Q.P \vee Q \Rightarrow Q \vee P$ assuming appropriate templates for implication and disjunction.

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

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

presentation

To annotate a symbol with notation information OMDoc supplies the `presentation` element. It is a top-level element whose `for` attribute points to the symbol in question. It contains a multilingual `CMP` group that allows to specify the notation[4]. Like the `omstyle` element, it has children that specify the presentation: The `xslt` element can be used to literally include the body of the template, and the `style` can express the presentation directives natively in OMDoc. In Listing 19.3 we have juxtaposed the presentational content from Listing 19.2 in `xslt` and `style` elements. Note that the directives in their body share much of the structure; the directives in the `style` are somewhat more succinct. The main difference to the XSLT template in Listing 19.2 is the specification of the template head: the attributes in the `presentation` element carry all the information necessary to identify the application conditions.

Listing 19.3. A Simple `presentation` Element for the Universal Quantifier

```
<presentation for="#quant1.forall" role="binding">
  <CMP>We write
    <OMOBJ>
      <OMBIND><OMS cd="quant1" name="forall"/>
        <OMBVAR><OMV name="X></OMBVAR>
        <OMV name="A"/>
      </OMBIND>
    </OMOBJ>
    for the phrase "A holds for all X".
  </CMP>
  <xslt format="default" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:text>∀</xsl:text>
    <xsl:for-each select="OMBVAR"/>
     <xsl:apply-templates/>
     <xsl:if test="position()!=last()">,</xsl:if>
    </xsl:for-each>
    <xsl:text>.</xsl:text>
    <xsl:apply-templates select="*[3]"/>
  </xslt>
  <style format="html">
    <text>&amp;#8704;</text>
    <map select="OMBVAR/*">
      <separator><text>,</text></separator>
      <recurse/>
    </map>
    <text>.</text>
    <recurse select="*[3]"/>
  </style>
  <style format="pmml">
    <element crid="." name="mrow" ns="http://www.w3.org/1998/Math/MathML">
      <element crid="*[1]" cr="yes" name="mo"><text>&amp;#8704;</text></element>
      <element name="mrow" crid="*[2]">
```

[4] Of course in the content markup in OMDoc, this looks somewhat awkward, since the representation relies on the fact that it will be rendered in the correct way. In the source, the whole markup looks somewhat circular.

```
      <map select="OMBVAR/*">
        <separator>
          <element name="mo" cr="yes">
            <attribute name="separator"><text>true</text></attribute>
            <text>,</text>
          </element>
        </separator>
        <recurse/>
      </map>
    </element>
    <recurse select="*[3]"/>
  </element>
 </style>
</presentation>
```

The `element` element can have the `crid` attribute which specifies the role of the generated element in parallel markup of mathematical formulae (see Subsection 2.1.1). The value of this element (if present) must be a XPath fragment (see [CD99]) pointing to the element in the source that semantically corresponds to the generated element (see Listing 19.3[5]). Finally, the `element` element can carry the `cr` attribute, which (if its value is `yes`) instructs the presentation system to to set an `xlink:href` attribute on the result element that acts as a cross-reference to the symbol declaration.

19.3.2 Specifying Notation via Syntactic Roles

Note that hand-coding XSLT-templates is a tedious and error-prone process, and that we need a template for each output format (e.g. LaTeX, HTML, Presentation-MathML, ASCII), and even various output languages (for instance the greatest common divisor of two integers is expressed by the symbol *gcd* in English but *ggT* ("größter gemeinsamer Teiler") in German). Obviously, the respective templates for all of these transformations share a great deal of structure (in our example, they only differ in the representation of the glyph for the quantifier itself).

Therefore OMDoc goes another step and supplies a set of abbreviations that are sufficient for most presentation applications via the `use` elements that can occur as children of `presentation` elements. The user only needs to specify the relevant information in the `use` elements and a separate translation process generates the needed XSLT templates from that (see Chapter 25). The `use` elements make use of the same symbolic attributes and specialize (overdefine) these attributes according to the respective format and language. The following set of attributes are particular to the `presentation`, since they are independent of the language and the output format.

use

`for`, `xref`, `class` (see the specification for `omstyle` in the last section)

`role` This attribute specifies to which roles of the symbol the `presentation` element applies. The value of this attribute can be one of

[5] There the top-level generated `mrow` element corresponds to the application as specified by the path ".", whereas its first child corresponds to the quantifier symbol, and the bound variables correspond to each other.

`applied` for situations, where the symbol occurs as a function symbol that is applied to a list of arguments, i.e. as the first child of an `om:OMA` or an `m:apply` element.

`binding` for situations, where the symbol occurs as a binding symbol, i.e as the first child of an `om:OMBIND` element or an `m:apply` element that is followed by an `m:bvar` element.

`key` for situations, where the symbol occurs as a key in an attribution, i.e. as a child of an `om:OMATTR` element at an odd position (ContentMathML does not have the attribution construct).

In the examples in Figure 19.4 we have assumed the head to be an `om:OMA` element (for functional application). It can also be an `om:OMBIND` as in the case of a quantifier in Figure 19.5.

`fixity` This optional attribute can be one of the keywords `prefix` (the default), `infix`, `postfix`, and `assoc`. The value `assoc` has two variants: `infixl` and `infixr`, which have the same presentation; `infixl` is used for a binary infix operator that associates to the left like the list constructor in Standard ML, `infixr` is the right-leaning analogon.

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

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

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

Figure 19.4 shows some combinations of attributes and their results on the function style.

`precedence` allows us to specify the operator precedence in order to elide unnecessary brackets. The OMDoc presentation system orients itself on the Prolog standard: lower precedences mean stronger binding, and brackets can be omitted. If we set the default precedence to 1000, and other precedences as specified in Figure 19.3, then the formulae below are presented as $(x+2)^2$ and $x+y^2$, respectively.

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

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

Precedence	Operators	Comment
200	+,-	unary
200	^	exponentiation
400	$*, \wedge, \cap$	multiplicative
500	$+, -, \vee, \cup$	additive
600	/	fraction
700	$=, \neq, \leq, <, >, \geq$	relation

Fig. 19.3. Common Operator Precedences

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

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

`separator` This specifies the separator in the argument list of a function. The default for `separator` is the comma. See Figure 19.4 for some combinations.

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

Fig. 19.4. Attribute-Combination and Function Style

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

symbol, `lbrack`, `rbrack`, `brackets`, only on the left/right/both brackets, `separator`, on the separator, and finally `all` on all presentation parts.
In Figure 19.5, the effect of the default `yes` can be seen in the lower part of the figure: the LaTeX and the HTML presentations have attached hyperlinks to the representation of the universal quantifier.

Notation specification	Example
<presentation for="#forall" role="binding" separator="."> <use format="TeX">\forall</use> <use format="html">∀</use> </presentation>	<OMBIND> <OMS cd="quant1" name="forall"/> <OMBVAR> <OMV name="X"/> </OMBVAR> <OMS cd="logic1" name="true"/> </OMBIND>

Using XSLT templates induced from the `presentation` element on the OpenMath expression yields $\forall X.\mathsf{true}$, where the glyph $\forall$ carries a hyperlink[6] to it definition, as the `crossref-symbol` on the `presentation` element has the default value `yes`. Internally, the hyperlinks are format-dependent, we have:
LaTeX: `\href{../ocd/logic1.ps#true}{\forall}X.`
`\href{../ocd/logic1.ps#true}{{\sf true}}`
HTML: `<a href="../ocd/logic1.html#forall">∀</a> X.`
`<a href="../ocd/logic1.html#true"><b>true</b></a>`	

Fig. 19.5. Notation for `forall` (cf. Listing 19.2) using `presentation`

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

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

`element`, `attributes`, `bracket-style` These attributes simplify the specification of notations in XML-based formats like MathML. The `element` attribute contains the name and the `attributes` the attribute declarations of an XML element that takes the place of the brackets specified in the attributes `lbrack` and `rbrack`. If the attribute `fixity` is used on a `use` element in conjunction with the `element` and `attributes` attributes, then it specifies the position of the element brackets rather than the brackets specified in the `lbrack` and `rbrack` attributes.
For instance, the binomial coefficient is some presented as $\binom{n}{m}$ (spoken "n choose m") and represented as

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

in Presentation-MathML. The first `presentation` element in Listing 19.4 shows a `presentation` element that has this effect. The second `presentation` element in Listing 19.4 shows a notation declaration, which applied to

```
<OMA><OMS cd="arith" name="power"/>
  <OMI>3</OMI><OMI>5</OMI>
</OMA>
```

would yield `3<sup>5</sup>` for the target `html`.

Listing 19.4. Presentation for Binomial Coefficients

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

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

Conceptually, the attributes of the `presentation` and `use` elements form a meta-language for XSLT style sheets that aims at covering the most common notations succinctly and legibly. In situations, where this language does not suffice, we must fall back to to `style` or even `xslt` elements.

19.4 Presenting Bound Variables

As we have seen in Section 13.4, the presentation approaches for symbols do not work for (bound) variables[7], as there is no independent place to put the `presentation` element. In this section, we will present the OMDoc solution to this problem. The main idea is simply to annotate defining occurrences of variables with notation information. Without this, we are forced to use the ASCII variable name in OpenMath and a translation of the Presentation-MathML in the `m:ci` element for other formats in MathML. This is hardly adequate for modern mathematics, where variables are numbered, decorated with primes or change marks, and cast in other colors or font families for better recognition.

In OMDoc we follow the spirit of the OpenMath standard [BCC+04] which suggests to annotate (via `om:OMATTR` parts of) the OpenMath objects with notation information by `presentation` elements. Unlike OpenMath, we restrict this practice to defining occurrences of bound variables, since all the other constructs can be handled with the methods introduced above. We

[7] We say that an `om:OMBIND` element binds a 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"/>`). For content MathML, the definition is analogous, only that an `m:apply` element with `m:bvar` child takes the role of the `om:OMBIND` and `om:OMBVAR` elements.

use the symbol `<OMS cd="omdoc" name="notation"/>` symbol to identify the following object as a notation declaration and the `om:OMFOREIGN` element to hold it. BErr(78)

Listing 19.5. Notation for Bound Variables in OpenMath

```
<OMOBJ>
  <OMBIND>
    <OMS cd="quant1" name="forall"/>
    <OMBVAR>
      <OMATTR>
        <OMATP>
          <OMS cd="omdoc" name="notation"/>
          <OMFOREIGN encoding="application/omdoc+xml">
            <presentation for="#X">
              <use format="TeX">X_4</use>
              <use format="pmml">
                <msub><mi>X</mi><mn>4</mn></msub>
              </use>
              <use format="html">X<sub>4</sub></use>
            </presentation>
          </OMFOREIGN>
        </OMATP>
        <OMV name="X4"/>
      </OMATTR>
    </OMBVAR>
    <OMA><OMS cd="relation1" name="eq"/>
      <OMV name="X4"/>
      <OMV name="X4"/>
    </OMA>
  </OMBIND>
</OMOBJ>
```

To represent binding objects in Content-MathML we follow a very similar strategy, using the `m:semantics` element to associate the defining occurrence of the bound variable with its notation declaration, which is embedded into the `m:annotation-xml` child.

Listing 19.6. Notation for Bound Variables in Content-MathML

```
<m:math>
  <m:apply>
    <m:forall/>
    <m:bvar>
      <m:semantics>
        <m:ci><m:msub><m:mi>X</m:mi><m:mn>4</m:mn></m:msub></m:ci>
        <m:annotation-xml encoding="application/xml+OMDoc"
          definitionURL="http://omdoc.org/omdoc.omdoc#notation">
          <presentation for="#X4">
            <use format="TeX">X_4</use>
            <style format="pmml">
              <element name="msub" ns="http://www.w3.org/1998/Math/MathML">
                <element name="mi" ns="http://www.w3.org/1998/Math/MathML">
                  <text>X</text>
                </element>
                <element name="mn" ns="http://www.w3.org/1998/Math/MathML">
                  <text>4</text>
                </element>
              </element>
            </style>
```

[78] Erratum: The `for` attribute should be `#X4` instead of `#X` in listings 19.5 and 19.6

```
            <style format="html">
              <text>X</text>
              <element name="sub" ns="http://www.w3.org/1999/xhtml">
24              <text>4</text>
              </element>
            </style>
          </presentation>
        </m:annotation-xml>
29    </m:semantics>
  </m:bvar>
  <m:apply><m:eq/><m:cn>4</m:cn><m:cn>4</m:cn></m:apply>
 </m:apply>
</m:math>
```

EErr(78)

With these declarations, all the variables in the scope of the universal quantifier would be represented as X_4, yielding $\forall X_4.X_4 = X_4$ which is exactly what we wanted. Note that if we want to specify notations for function variables (OMDoc does not prevent the user from doing this), we need to also specify notations for the non-applied occurrences of the symbol — otherwise a fallback using the variable name has to be used. For instance, to make the (false) conjecture that all relations are symmetric we could use the following representation:

Listing 19.7. Notation for bound variables in OpenMath

```
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
2  <OMBIND>
    <OMS cd="quant1" name="forall"/>
    <OMBVAR>
     <OMATTR>
      <OMATP>
7      <OMS cd="omdoc" name="notation"/>
       <OMFOREIGN encoding="application/omdoc+xml">
        <presentation xmlns="http://www.mathweb.org/omdoc"
                      for="#R" role="applied" precedence="500" fixity="infix">
         <use format="TeX">\prec</use>
12       <use format="pmml|html">&#x022DE;</use>
        </presentation>
        <presentation xmlns="http://www.mathweb.org/omdoc" for="#R">
         <use format="TeX">{}\prec{}</use>
         <use format="pmml|html">&#x022DE;</use>
17      </presentation>
       </OMFOREIGN>
      </OMATP>
      <OMV name="R"/>
     </OMATTR>
22   <OMV name="X"/>
    </OMBVAR>
    <OMA><OMV name="R"/><OMV name="X"/><OMV name="X"/></OMA>
  </OMBIND>
</OMOBJ>
```

This would give us the presentation $\forall \prec, X.X \prec X$. Here, the first occurrence of the variable $\prec$ is handled by the second notation declaration (it does not occur in applied position), the second occurrence of $\prec$ is in applied position, so the second notation declaration governs this and puts it in to infix position. Note that while OMDoc allows to specify this kind of notation declarations, they should be used with great care and discretion. In this particular case, the infix notation of $\prec$ de-emphasizes the variable nature, and might lead

to confusion; moreover, the particular choice of the glyph $\prec$ may suggest irreflexivity, which may or may not be intended.

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 document**s. 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 26.4 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 [Koha], 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 [CKOS03] 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. BErr(79)

Listing 20.1. Reformulating Mathematical Knowledge

```
<assertion xml:id="ALGX0">
  <CMP>If a, b, c, d are numbers, then we have a + b(c + d) = a + bc + bd.</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>
```

EErr(79)

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

[79] Erratum: The reference `reformulates="ALGX0"` should be a URI reference, i.e. `#ALGX0`

[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

as the transformation process can then choose the most suitable one for the target.

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>None.</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>None: the applet handles input itself.</CMP></input>
  <output><CMP>The result of the integration.</CMP></output>
  <effect><CMP>None.</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. BErr(80)

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

EErr(80)

omlet

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.

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

[5] In particular, the JavaScript references the HTML DOM, which in our model is created by a presentation engine on the fly.

[80] Erratum: Wrong Content Model for `omlet`

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 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 [Gro00], 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.

[6] These latter two attributes are modeled after the XLink [DMOT01] attributes `show` and `actuate`.

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>
```

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>
    <p>Let practice integration!</p>
    <p><omlet data="#callMint" action="execute" style="width:320;height:200">
        No plug-in found for callMint!
      </omlet></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> 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>
</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 runtime. 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 26.8 and [GMUC03] 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

exercise

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).

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>
    <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
      <phrase style="font-family:fixed">\hbox</phrase> 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>
  </CMP> <hint>
    <CMP>Go back and forth between math mode and horizontal mode.</CMP>
  </hint>
  <solution>
    <CMP>
      <phrase style="font-family:fixed">
       $\{\,p\mid p$~and $p+2$ are prime$\,\}$
      </phrase>,
      assuming that <phrase style="font-family:fixed">\mathsurround</phrase> is
      zero. The more difficult alternative '<phrase style="font-family:fixed">
      $\{\,p\mid p\\ {\rm and}\ p+2\rm\ are\ prime\,\}$</phrase>'
      is not a solution, because line breaks do not occur at
      <phrase style="font-family:fixed">\␣</phrase> (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.
    </CMP>
  </solution>
</exercise>
```

mc

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).

choice

`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.

`hint` (optional) for a hint to the user, see above for a description.

answer

`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.

Listing 21.2. A Multiple-Choice Exercise in OMDoc

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

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 [BCC+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 [ABC+03a] or XHTML [Gro00] specify models in form of the intended screen presentation, while still others like the XSLT [XSL99] 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 [**?**] 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. BErr(81)

Listing 22.1. An OMDoc Definition

```
<definition xml:id="comm-def" for="comm">
  <CMP xml:lang="en">
    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>.
  </CMP>
  <CMP xml:lang="de">
    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>.
  </CMP>
</definition>
```

EErr(81)

Listing 22.2. An XML-equal serialization for Listing 22.1 BErr(82)

```
<definition for="comm" xml:id="comm-def" >
  ...
  <CMP xml:lang='de'> <!-- Note the unabbreviated empty element -->
   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>.
  </CMP>
</definition>
```

EErr(82)

[81] Erratum: `for` attribute on `definition` should be of type `NCNames`

[82] Erratum: `for` attribute on `definition` should be of type `NCNames`

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.

BErr(83)

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">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>.
  </CMP>
  <CMP xml:lang="en">
    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>.
  </CMP>
</definition>
```

EErr(83)

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

[83] Erratum: `for` attribute on `definition` should be of type `NCNames`

#	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` `ref`
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

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-structured** and specify this by the value `Text` (also see the discussion in Section 12.1)

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 IV 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.

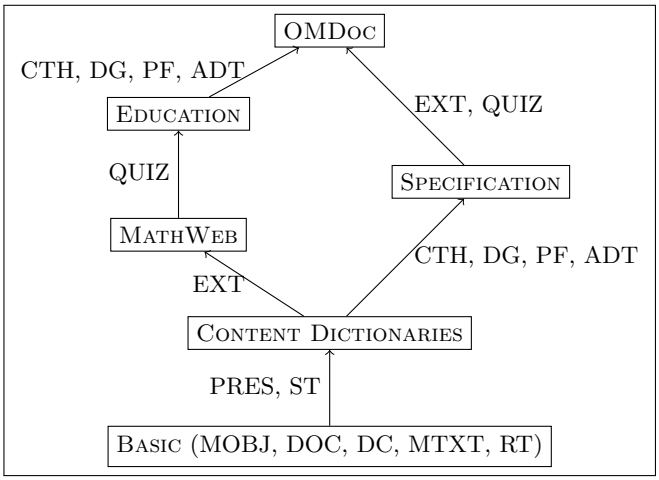


Fig. 22.2. OMDoc Sub-Languages and Modules

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 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 [BCC+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 [CoF04]) 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 [AHMS00, 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 [WM99] or XHTML [Gro00] 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

OMDoc Applications, Tools, and Projects

oIn this part we will address current applications, tools and projects using the OMDoc format. We will first discuss the possibilities and tools of processing documents in the OMDoc format via style sheets with the purpose of generating documents specialized for consumption by other mathematical software systems, and by humans. Then we will present three projects descriptions that use OMDoc at the core.

23

OMDoc resources

In this chapter we will describe various public resources for working with the OMDoc format.

23.1 The OMDoc Web Site, Wiki, and Mailing List

The main web site for the OMDoc format is `http://omdoc.org`. It hosts news about developments, applications, collaborators, and events, provides access to an list of "frequently asked questions" (FAQ), and current and old OMDoc specifications and provides pre-generated examples from the OMDoc distribution.

There are two mailing lists for discussion of the OMDoc format:

`omdoc@omdoc.org` is for announcements and discussions of the OMDoc format on the user level. Direct your questions to this list.

`omdoc-dev@omdoc.org` is for developer discussions.

For subscription and archiving details see the OMDoc resources page for mailing lists [Koh08].

Finally, the OMDoc web site hosts a Wiki [OMDb] for user-driven documentation and discussion.

23.2 The OMDoc distribution

All resources on the OMDoc web site are available from the MathWeb Subversion repository [OMDa] for anonymous download. Subversion (svn) is a collaborative version control system – to support a distributed community of developers in accessing and developing the OMDoc format, software, and documentation, see [Mat] for a general introduction to the setup. The head revision of the OMDoc repository are accessible on the web at `https://svn.omdoc.org/repos/omdoc/trunk` via a regular web browser. The svn server

allows anonymous read access to the general public. To check out the OMDoc distribution, use

```
svn co https://svn.omdoc.org/repos/omdoc/trunk
```

This will create a directory `omdoc`, with the sub-directories

directories	content
`bin`, `lib`, `oz`, `thirdParty`	programs and third-party software used in the administration and examples
`css`, `xsl`	style sheets for displaying OMDoc documents on the web, see Chapter 25 for a discussion.
`doc`	The OMDoc documentation, including the specification, papers about a the OMDoc format and tools.
`dtd`, `rnc`	The OMDoc document type definition and the RelaxNG schemata for OMDoc
`examples`	Various example documents in OMDoc format.
`projects`	various contributed developments for OMDoc. Documentation is usually in their `doc` sub-directory

After the initial check out, the OMDoc distribution can be kept up to date by the command `svn -q update` in the top-level directory from time to time. To obtain write access contact `svnadmin@omdoc.org`.

23.3 The OMDoc bug tracker

`MathWeb.org` supplies a BugZilla bug-tracker [Bug05] at `http://bugzilla.mathweb.org:8000` to aid the development of the OMDoc format. BugZilla is a server-based discussion forum and bug tracking system. We use it to track, archive and discuss tasks, software bugs, and enhancements in our project. Discussions are centered about threads called "bugs" (which need not be software bugs at all), which are numbered, can be searched, and can be referred to by their URL. To use BugZilla, just open an account and visit the OMDoc content by querying for the "product" OMDoc. For offline use of the bug-tracker we recommend the excellent Deskzilla application [Des05], which is free for open-source projects like OMDoc.

Further development of the OMDoc format will be public and driven by the discussions on BugZilla, the OMDoc mailing list, and the OMDoc Wiki (see Section 23.1).

23.4 An XML catalog for OMDoc

Many XML processes use system IDs (in practice URLs) to locate supporting files like DTDs, schemata, style sheets. To make them more portable, OMDoc documents will often reference the files on the `omdoc.org` web server, even in

situations, where they are accessible locally e.g. from the OMDoc distribution. This practice not only puts considerable load on this server, but also slows down or even blocks document processing, since the XML processors have to retrieve these files over the Internet.

Many processors can nowadays use XML catalogs to remap public identifiers and URLs as an alternative to explicit system identifiers. A catalog can convert public IDs like the one for the OMDoc DTD (`-//OMDoc//DTD OMDoc V1.2//EN`) into absolute URLs like `http://omdoc.org/dtd/omdoc.dtd`. Moreover, it can replace remote URLs like this one with local URLs like `file:///home/kohlhase/omdoc/dtd/omdoc.dtd`. This offers fast, reliable access to the DTDs and schemata without making the documents less portable across systems and networks.

To facilitate the use of catalogs, the OMDoc distribution provides a catalog file `lib/omdoc.cat`. This catalog file can either be imported into the system's catalog[1] using a `nextCatalog` element of the form

```
<nextCatalog xml:id="omdoc.cat" catalog="file:///home/kohlhase/omdoc/lib/omdoc.cat"/>
```

or by making it known directly to the XML processor by an application-specific method. For instance for `libxml2` based tools like `xsltproc` or `xmllint`, it is sufficient to include the path to `omdoc.cat` in the value of the `XML_CATALOG_FILES` environment variable (it contains a whitespace-separated list of FILES).

23.5 External Resources

The OMDoc format has been used on a variety of projects. Chapter 26 gives an overview over some of the projects (use the project home pages given there for details), a up to date list of links to OMDoc projects can be found at `http://omdoc.org/projects/`. These projects have contributed tools, code, and documentation to the OMDoc format, often stressing their special vantage points and applications of the format.

[1] This catalog is usually at `file:///etc/xml/catalog` on Unix systems; unfortunately there is no default location for Windows machines.

24

Validating OMDoc Documents

In Chapter 1 we have briefly discussed the basics of validating XML documents by document type definitions (DTDs) and schemata. In this chapter, we will instantiate this discussion with the particulars of validating OMDoc documents.

Generally, DTDs and schemata are context-free grammars for trees[1], that can be used by a **validating parser** to reject XML documents that do not conform to the constraints expressed in the OMDoc DTD or schemata discussed here.

Note that none of these grammars can enforce all constraints that the OMDoc specification in Part III of this book imposes on documents. Therefore grammar-based validation is only a necessary condition for OMDoc-**validity**. Still, OMDoc documents should be validated to ensure proper function of OMDoc tools, such as the ones discussed in Chapters 25 and 26. Validation against multiple grammars gives the best results. With the current state of validation technology, there is no clear recommendation, which of the validation approaches to prefer for OMDoc. DTD validation is currently best supported by standard XML applications and supports default values for attributes. This allows the author who writes OMDoc documents by hand to elide implicit attributes and make the marked-up text more readable. XML- and RelaxNG schema validation have the advantage that they are namespace-aware and support more syntactic constraints. Neither of these support mnemonic XML entities, such as the ones used for Unicode characters in PresentationMathML, so that these have to be encoded as Unicode code points. Finally RelaxNG schemata do not fully support default values for attributes, so that OMDoc documents have to be normalized[2] to be RelaxNG-valid.

[1] Actually, a recent extension of the XML standard (XLink) also allows to express graph structures, but the admissibility of graphs is not covered by DTD or current schema formalisms.

[2] An OMDoc document is called **normalized**, iff all required attributes are present. Given a DTD or XML schema that specifies default values, there are

We will now discuss the particulars of the respective validation formats. As the RelaxNG schema is the most expressive and readable for humans we consider it as the normative grammar formalism for OMDoc, and have included it in Appendix D for reference.

24.1 Validation with Document Type Definitions

The OMDoc document type definition [Kohc] can be referenced by the public identifier `"-//OMDoc//DTD OMDoc V1.2//EN"` (see Section 23.4). The DTD driver file is `omdoc.dtd`, which calls various DTD modules.

DTD-validating XML parsers are included in almost all XML processors. The author uses the open-source RXP [Tob] and xmllint [Veia] as standalone tools. If required, one may validate OMDoc documents using an SGML parser such as nsgmls, rather than a validating XML parser. In this case an SGML declaration defining the constraints of XML applicable to an SGML parser must be used (see [Cla97] for details).

To allow DTD-validation, OMDoc documents should contain a document typedeclaration of the following form:

```
<!DOCTYPE omdoc PUBLIC "-//OMDoc//DTD OMDoc V1.2//EN"
                       "http://omdoc.org/dtd/omdoc.dtd">
```

The URI may be changed to that of a local copy of the DTD if required, or it can be dropped altogether if the processing application has access to an XML catalog (see Section 23.4). Whether it is useful to include document type declarations in documents in a production environment depends on the application. If a document is known to be DTD- or even OMDoc-valid, then the validation overhead a `DOCTYPE` declaration would incur from a validating parser[3] may be conserved by dropping it.

24.1.1 Parametrizing the DTD

The OMDoc DTD makes heavy use of parameter entities, so we will briefly discuss them to make the discussion self-contained. Parameter entity declarations are declarations of the form

```
<!ENTITY % assertiontype "theorem|proposition|lemma|%otherassertiontype;">
```

standard XML tools for XML-normalization that can be pipelined to allow RelaxNG validation, so this is not a grave restriction.

[3] The XML specification requires a validating parser to perform validation if a `DOCTYPE` declaration is present

in the DTD. This one makes the abbreviation `%assertiontype;` available for the string "`theorem|proposition|lemma|observation`" (in the DTD of the document in Listing 24.1). Note that parameter entities must be fully defined before they can be referenced, so recursion is not possible. If there are multiple parameter entity declarations, the first one is relevant for the computation of the replacement text; all later ones are discarded. The internal subset of document type declaration is pre-pended to the external DTD, so that parameter entity declarations in the internal subset overwrite the ones in the external subset.

The (external) DTD specified in the `DOCTYPE` declaration can be enhanced or modified by adding declarations in square brackets after the DTD URI. This part of the DTD is called the internal subset of the `DOCTYPE` declaration, see Listing 24.1 for an example, which modifies the parameter entity `%other-assertiontype;` supplied by the OMDoc DTD to extend the possible values of the `type` attribute in the `assertion` element for this document. As a consequence, the `assertion` element with the non-standard value for the `type` attribute is DTD-valid with the modified internal DTD subset.

Listing 24.1. A Document Type Declaration with Internal Subset

```
<!DOCTYPE omdoc PUBLIC "-//OMDoc//DTD OMDoc V1.2//EN"
                       "http://omdoc.org/omdoc.dtd"
  [<!ENTITY % otherassertiontype "observation">]>
...
<assertion type="observation">...</assertion>
...
```

24.1.2 DTD-based Normalization

Note that if a OMDoc fragment is parsed without a DTD, i.e. as a well-formed XML fragment, then the default attribute values will not be added to the XML information set. So simply dropping the DOCTYPE declaration may change the semantics of the document, and OMDoc documents should be normalized[4] first. Normalized OMDoc documents should carry the `standalone` attribute in the XML processing instruction, so that a normalized OMDoc document has the form given in Listing 24.2.

Listing 24.2. A normalized OMDoc document without DTD

```
<?xml version="1.0" standalone="yes"?>
<omdoc xml:id="something" version="1.2" xmlns="http://www.mathweb.org/omdoc">
 ...
</omdoc>
```

The attribute `version` and the namespace declaration `xmlns` are fixed by the DTD, and need not be explicitly provided if the document has a `DOCTYPE` declaration.

[4] The process of DTD-normalization expands all parsed XML entities, and adds all default attribute values

24.1.3 Modularization

In OMDoc 1.2 the DTD has been modularized according to the W3C conventions for DTD modularization [Alt01]. This partitions the DTD into **DTD module**s that correspond to the OMDoc modules discussed in Part III of this book.

These DTD modules can be deselected from the OMDoc DTD by changing the **module inclusion entities** in the local subset of the document type declaration. In the following declaration, the module PF (see Chapter 17) has been deselected, presumably, as the document does not contain proofs.

```
1 <!DOCTYPE omdoc PUBLIC "-//OMDoc//DTD OMDoc V1.2//EN"
                         "http://omdoc.org/dtd/omdoc.dtd"
  [<!ENTITY % omdoc.pf.module "IGNORE">]>
```

Module inclusion entities have the form `%omdoc.`⟪*ModId*⟫`.module;`, where ⟪*ModId*⟫ stands for the lower-cased module identifier. The OMDoc DTD repository contains DTD driver files for all the sub-languages discussed in Section 22.3, which contain the relevant module inclusion entity settings. These are contained in the files `omdoc-`⟪*SlId*⟫`.dtd`, where ⟪*SlId*⟫ stands for the sub-language identifier.

Except for their use in making the OMDoc DTD more manageable, DTD modules also allow to include OMDoc functionality into other document types, extending OMDoc with new functionality encapsulated into modules or upgrading selected OMDoc modules individually. To aid this process, we will briefly describe the module structure. Following [Alt01], DTD modules come in two parts, since we have inter-module recursion. The problem is for instance that the `omlet` element can occur in mathematical texts (`mtext`), but also contains `mtext`, which is also needed in other modules. Thus the modules cannot trivially be linearized. Therefore the DTD driver includes an entity file ⟪*ModId*⟫`.ent` for each module ⟪*ModId*⟫, before it includes the grammar rules in the **element module**s ⟪*ModId*⟫`.mod` themselves. The entity files set up parameter entities for the qualified names and content models that are needed in the grammar rules of other modules.

24.1.4 Namespace Prefixes for OMDoc elements

Document type definitions do not natively support XML namespaces. However, clever coding tricks allow them to simulate namespaces to a certain extent. The OMDoc DTD follows the approach of [Alt01] that parametrizes namespace prefixes in element names to deal gracefully with syntactic effects of namespaced documents like we have in OMDoc.

Recall that element names are **qualified name**s, i.e. pairs consisting of a namespace URI and a local name. To save typing effort, XML allows to abbreviate qualified names by namespace declarations via `xmlns` pseudo-attribute: the element and all its descendants are in this namespace, unless they have a namespace attribute of their own or there is a namespace declaration in

a closer ancestor that overwrites it. Similarly, a namespace abbreviation can be declared on any element by an attribute of the form `xmlns:nsa="nsURI"`, where `nsa` is a name space abbreviation, i.e. a simple name, and `nsURI` is the URI of the namespace. In the scope of this declaration (in all descendants, where it is not overwritten) a qualified name `nsa:n` denotes the qualified name `nsURI:n`.

The mechanisms described in [Alt01] provide a way to allow for namespace declarations even in the (namespace-agnostic) DTD setting simply by setting a parameter entity. If `NS.prefixed` is declared to be `INCLUDE`, using a declaration such as `<!ENTITY % NS.prefixed "INCLUDE">` either in the local subset of the `DOCTYPE` declaration, or in the DTD file that is including the OMDoc DTD, or the DTD modules presented in this appendix, then all OMDoc elements should be used with a prefix, for example `<omdoc:definition>`, `<omdoc:CMP>`, etc. The prefix defaults to `omdoc:` but another prefix may be declared by declaring in addition the parameter entity `omdoc.prefix`. For example, `<!ENTITY % omdoc.prefix "o">` would set the prefix for the OMDoc namespace to `o:`.

Note that while the Namespaces Recommendation [Bra99] provides mechanisms to change the prefix at arbitrary points in the document, this flexibility is not provided in this DTD (and is probably not possible to specify in any DTD). Thus, if a namespace prefix is being used for OMDoc elements, so that for example the root element is:

```
<omdoc:omdoc xmlns:omdoc="http://www.mathweb.org/omdoc">
...
</omdoc:omdoc>
```

then the prefix must be declared in the local subset of the DTD, as follows:

```
<!DOCTYPE omdoc:omdoc PUBLIC "-//OMDoc//DTD OMDoc V1.2//EN"
                             "http://omdoc.org/dtd/omdoc.dtd"
  [<!ENTITY % NS.prefixed "INCLUDE"><!ENTITY % omdoc.prefix "omdoc">]>
```

The OMDoc DTD references six namespaces:

language	namespace	prefix
MathML	`http://www.w3.org/1998/Math/MathML`	`m:`
OpenMath	`http://www.openmath.org/OpenMath`	`om:`
XSLT	`http://www.w3.org/1999/XSL/Transform`	`xsl:`
Dublin Core	`http://purl.org/dc/elements/1.1/`	`dc:`
Creative Commons	`http://creativecommons.org/ns`	`cc:`
OMDoc	`http://www.mathweb.org/omdoc`	`omdoc:`

These prefixes can be changed just like the OMDoc prefix above.

24.2 Validation with RelaxNG Schemata

RelaxNG [Vli03] is a relatively young approach to validation developed outside the W3C, whose XML schema paradigm was deemed overburdened. As

a consequence, RelaxNG only concerns itself with validation, and leaves out typing, normalization, and entities. RelaxNG schemata come in two forms, in XML syntax (file name extension `.rng`) and in compact syntax (file name extension `.rnc`). We provide the RelaxNG schema [Kohd] as the normative validation schema for OMDoc. As compact syntax is more readily understandable by humans, we have reprinted it as the normative grammar for OMDoc documents in Appendix D. Just as in the case for the OMDoc DTD, we provide schemata for the OMDoc sub-languages discussed in Section 22.3. These are contained in the driver files `omdoc-`⟪*SlId*⟫`.rnc`, where ⟪*SlId*⟫ stands for the sub-language identifier.

There is currently no standard way to associate a RelaxNG schema with an XML document[5]; thus validation tools (see `http://relaxng.org` for an overview) have to be given a grammar as an explicit argument. One consequence of this is that the information that an OMDoc document is intended for an OMDoc sub-languages must be managed outside the document separately from the document.

There are various validators for RelaxNG schemata, the author uses the open-source xmllint [Veia] as a stand-alone tool, and the nXML mode [Cla05] for the Emacs editor [Sta02] for editing XML files, as it provides powerful RelaxNG-based editing support (validation, completion, etc.).

24.3 Validation with XML Schema

For validation[6] with respect to XML schemata (see [XML]) we provide an XML schema for OMDoc [Kohe], which is generated from the RelaxNG schema in Appendix D via the trang system described above. Again, schemata for the sub-languages discussed in Section 22.3 are provided as `omdoc-`⟪*SlId*⟫`.rnc`, where ⟪*SlId*⟫ stands for the sub-language identifier.

To associate an XML schema with an element, we need to decorate it with an `xsi:schemaLocation` attribute and the namespace declaration for XML schema instances. In Listing 24.3 we have done this for the top-level `omdoc` element, and thus for the whole document. Note that this mechanism makes mixing XML vocabularies much simpler than with DTDs, that can only be associated with whole documents.

Listing 24.3. An XML document with an XML Schema.

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

[5] In fact this is not an omission, but a conscious design decision on the part of the RelaxNG developers.

[6] There are many schema-validating XML parsers, the author uses the open-source xmllint [Veia].

```
...
</omdoc>
```

25

Transforming OMDoc by XSLT Style Sheets

In the introduction we have stated that one of the design intentions behind OMDoc is to separate content from presentation, and leave the latter to the user. In this section, we will briefly touch upon presentation issues. The technical side of this is simple: OMDoc documents are regular XML documents that can be processed by XSLT style sheets [XSL99] to produce the desired output formats. There are several high-quality XSLT transformers freely available including `saxon` [Kay], `xalan` [The], and `xsltproc`[Veib]. Moreover, XSLT is natively supported by the newest versions of the browsers MS Internet Explorer [Cor] and Mozilla [Org].

XSLT style sheets can be used for several tasks in maintaining OMDoc, such as for instance converting other XML-based input formats into OMDoc (e.g. `cd2omdoc.xsl` for converting OpenMath content dictionaries into OMDoc format), or migrating between different versions of OMDoc e.g. the style sheet `omdoc1.1adapt1.2.xsl` that operationalizes all the syntax changes from Version 1.1 of OMDoc to version 1.2 (see Appendix A for a tabulation). We will now review a set of XSLT style sheets for OMDoc, they can be found in the OMDoc distribution (see Section 23.2) or on the web at [Kohf].

25.1 Extracting and Linking XSLT Templates

One of the main goals of content markup for mathematical documents is to be independent of the output format. In Chapter 19, we have specified the conceptual infrastructure provided by the OMDoc language, in this section we will discuss the software infrastructure needed to transform OMDoc documents into various formats.

The `presentation` elements for symbols in OpenMath or ContentMathML formulae allow a declarative specification of the result of transforming expressions involving these symbols into various formats. To use this information in XSLT style sheets, the content of `presentation` elements must be transformed into XSLT templates, and these must be linked into the

generic transformation style sheet. The OMDoc distribution provides two meta-style-sheets for these tasks.

The first one — `expres.xsl` — compiles the content of the `presentation` and `omstyle` elements in the source file into XSLT templates. The style sheet takes the parameter `report-errors`, which is set to '`no`' by default; setting it to '`yes`' will enable more verbose error reports. The OMDoc distribution provides Unix `Makefiles` that specify the target ⟪*base*⟫`-tmpl.xsl` for each OMDoc file ⟪*base*⟫`.omdoc`, so that the templates file can be generated by typing `make` ⟪*base*⟫`-tmpl.xsl`. Note that `expres.xsl` follows the references in the `ref` elements (it `ref`-normalizes the document see Section 11.5) before it generates the templates[1].

The second style sheet — `exincl.xsl` — generates link table for a specific OMDoc document. This style sheet `ref`-normalizes the document and outputs an XSLT style sheet that includes all the necessary template files. `expres.xsl` takes two parameters: `self` is the name of the source file name itself[2]. The `Makefiles` in the OMDoc distribution specify the target ⟪*base*⟫`-incl.xsl`, so that the link table can be generated by typing `make` ⟪*base*⟫`-incl.xsl`.

Let us now consider the example scenario in Figure 25.1: Given an OMDoc document ⟪*document*⟫`.omdoc` that uses symbols from theories `a`, `b`, `c`, `d`, and `e` which are provided by the OMDoc documents ⟪*background*⟫`.omdoc`, ⟪*special*⟫`.omdoc` and ⟪*local*⟫`.omdoc`, we need to generate the template files ⟪*background*⟫`-tmpl.xsl`, ⟪*special*⟫`-tmpl.xsl`, and ⟪*local*⟫`-tmpl.xsl` (via `expres.xsl`) as well as ⟪*document*⟫`-incl.xsl` (via `exincl.xsl`). Now it is only necessary to include the link table ⟪*document*⟫`-incl.xsl` into a generic transformation style sheet to specialize it with the notation information specified in the `presentation` elements in theories `a`, `b`, `c`, `d`, and `e`.

The transformation architecture based on the `Makefiles` provided with the OMDoc distribution does the linking by creating a specialized style sheet ⟪*document*⟫`2html.xsl` that simply includes the generic OMDoc transformation style sheet `omdoc2html.xsl` (see Section 25.3), and the style sheet ⟪*document*⟫`-incl.xsl`. Changing this simple control style sheet allows to add site- or language-specific templates (by adding them directly or including respective style sheets). An analogous processing path leads ⟪*document*⟫`.tex` using `omdoc2tex.xsl` and from there to PDF using tools like `pdflatex`.

[1] In the current implementation, `expres.xsl` generates one large template that combines the XSLT code for all target formats. This simplifies the treatment of the default presentations as requested by the specification in Section 19.3, but hampers mixing presentation information from multiple sources. An implementation based on modes would probably have advantages in this direction in the long run.

[2] For some reason XSLT processors do not provide access to this information portably.

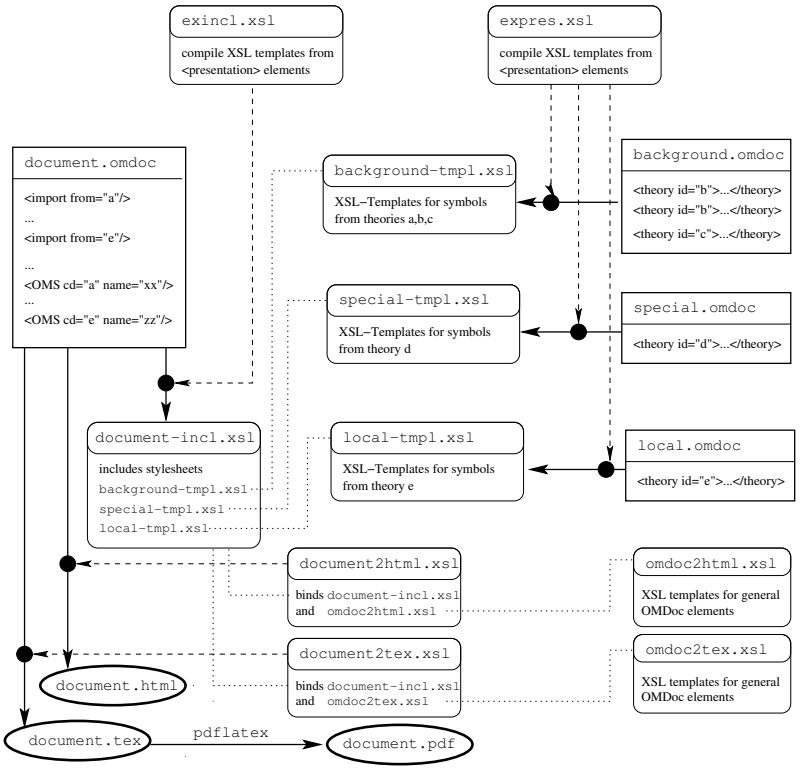


Fig. 25.1. The OMDoc presentation Process

Other processing architectures may be built up using on-demand technologies, e.g. servlets, mediators, or web services, but will be able to follow the same general pattern as our simpleminded implementation in `Makefiles`.

We will make use of this general architecture based on extraction and linking via XSLT style sheets in the transformation of OMDoc documents below.

25.2 OMDoc Interfaces for Mathematical Software Systems

One of the original goals of the OpenMath, Content-MathML and OMDoc languages is to provide a communication language for mathematical software systems. The main idea behind this is to supply systems with interfaces to a universally accepted communication language standard (an interlingua), and so achieve interoperability for n systems with only $2n$ translations instead of n^2. As we have seen in Section 2.1, OpenMath and Content-MathML pro-

vide a good solution at the level of mathematical objects, which is sufficient for systems like computer algebra systems. OMDoc adds the level of mathematical statements and theories to add support for automated reasoning systems and formal specification systems.

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

Usually, the task of writing an XSLT style sheet for such a conversion is a relatively simple task, since the input language of most mathematical software system is isomorphic to a subset of OMDoc. This suggests the general strategy of applying the necessary syntax transformations (this has to be supplied by the style sheet author) on those OMDoc elements that carry system-relevant information and transforming those that are not (e.g. Metadata and `CMP` elements for most systems) into comments. Much of the functionality is already supplied by the style sheet `omdoc2sys.xsl`, which need only be adapted to know about the comment syntax.

The task of translating an OMDoc document into system-specific input has two sub-tasks. We will discuss them using the concrete example of the `omdoc2pvs.xsl` style sheet that transforms OMDoc documents to the input language of the Pvs theorem prover [ORS92]: The first task is to translate elements at the statement- and theory level to the input language this is hand-coded by supplying suitable templates for the OMDoc statement and theory elements in an extension of the `omdoc2sys.xsl` style sheet. The second task is to translate the formulae to the input language. Here, the system usually has a particular way of expressing complex formulae like function applications and binding expressions; in the concrete case of Pvs, function application uses a prefix function argument syntax, and n-ary binding expressions, where the scope is separated by a colon from the variable list. This information must also be encoded in respective templates for the `om:OMA`, `om:OMBIND`, `om:OMV` elements from OpenMath and the `m:apply` and `m:ci` from Content-MathML. For the symbol elements, we have to distinguish two cases: the predefined symbols of the system language and the object symbols that are introduced by the user to formalize a certain problem. In both cases, the transformation procedure needs input on how these symbols are to be represented in the system language. For the object symbols we assume that there are suitable `theory` structures available, which declare them in `symbol` elements, thus we can assume that these `theory` structures also contain `use` elements with appropriate `format` attribute in the `presentation` elements for those symbols that need special representations in the system language. For the predefined symbols of the system language, we assume the same. To be able to transform an OMDoc document into system input, we need a language definition theory, i.e. an OMDoc document that contains a `theory` which provides `symbol`s for all the predefined words of the system language. This theory must also contain

`presentation` elements with `use` children specialized the input formats of all systems targeted for communication.

Listing 25.1. A `symbol` in a Language Definition Theory

```
<symbol name="sigmatype">
 <metadata>
   <dc:description>
     The dependent function type constructor is a binding operator. The source type is
     the type of the bound variable X, the target type is represented in the body.
   </dc:description>
 </metadata>
</symbol>

<presentation xml:id="pr-sigmatype" for="#sigmatype" role="binding">
  <style format="pvs">
    <text>[</text>
    <recurse select="*[2]/*"/><text> -&amp;gt; </text><recurse select="*[3]"/>
    <text>]</text>
  </style>
  <style format="nuprl">
    <recurse select="*[2]/*"/><text> -&amp;gt; </text><recurse select="*[3]"/>
  </style>
</presentation>
```

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

One of the practical problems that remains to be solved for interfaces between mathematical software systems is that of semantic standardization of input languages. For mathematical objects, this has been solved in principle by supplying a theory level in the form of OpenMath or OMDoc content dictionaries that define the necessary mathematical concepts. For systems like theorem provers or theory development environments we need to do the same with the logics underlying these systems. For an effort to systematize logics into a hierarchy that fosters reuse and communication of systems, based on a series of experiments of interfacing with the theorem proving systems Ωmega [BCF+97], InKa [HS96], Pvs [ORS92], λ*Clam* [RSG98b], TPS [ABI+96] and CoQ [Tea] see Section 26.18

25.3 Presenting OMDoc to Humans

We will now discuss the software infrastructure needed to transform OMDoc documents into human-readable form in various formats. We speak of of OMDoc **presentation** for this task.

Due to the complex nature of OMDoc presentation, only part of it can actually be performed by XSLT style sheets. For instance, sub-tasks like reasoning about the prior knowledge of the user, or her experience with certain proof techniques is clearly better left to specialized applications. Our processing model is the following: presenting an OMDoc is a two-phase process.

The first phase is independent of the final output format (e.g. HTML, MathML, or LaTeX) and produces another OMDoc representation specialized to the respective user or audience, taking into account prior knowledge, structural preferences, bandwidth and time constraints, etc. This phase usually generates a narrative-structured document from a knowledge-centered one.

The second phase is a formatting process that can be extracted by XSLT style sheets that transforms the resulting specialized document into the respective output format with notational- and layout preferences of the audience. We will only discuss the second one and refer the reader for ideas about the first process to systems like P.rex [Fie01a, FH01].

The presentation of the OMDoc document elements and statements is carried out by the style sheets `omdoc2html.xsl` for XHTML, `omdoc2html.xsl` for XHTML+MathML and `omdoc2tex.xsl` for LaTeX. These style sheets are divided into files according to the OMDoc modules and share a large common code base `omdoc2share.xsl`, basically the first two include the latter and only redefine some format-specific options. For instance, `omdoc2share.xsl` supplies an infrastructure for internationalization introduced in Section 14.1. This allows to generate localized presentations of the OMDoc documents, if enough information is present in the multilingual groups of `CMP` elements. `omdoc2share.xsl` takes a parameter `TargetLanguage`, whose value can be a whitespace-separated preference list of ISO 639 norm two-letter country codes. If `TargetLanguage` consists of a single entry, then the result will only contain this language with gaps where the source document contains no suitable `CMP`. Longer `TargetLanguage` preference lists will generally result in more complete, but multilingual documents. Apart from the language-specific elements in the source document, localization also needs to know about the presentation of certain keywords used in OMDoc markup, e.g. the German "Lemma" and the French "Lemme" for `<assertion type="lemma">`. This information is kept in the keyword table `lib/locale.xml` in the OMDoc distribution, which contains all the keywords necessary for presenting the OMDoc elements discussed so far. An alternative keyword table can be specified by the parameter `locale`.

26

OMDoc Applications and Projects

This chapter presents a variety of applications and projects that use the OMDoc format or are related to it in a substantive way.

Apart from the projects directly reported here, the OMDoc format is used by the new research field of Mathematical Knowledge Management (MKM; cf. `http://www.mkm-ig.org/`), which combines researchers in mathematics, computer science, and library science. We refer the reader to the proceedings of the annual MKM conference [BC01b, ABD03, ABT04, Koh05a, BF06].

26.1 Introduction

The text in the project descriptions has been contributed[1] by the authors marked in the section headings, for questions about the projects or systems, please visit the web-sites given or contact the authors directly. Note that the material discussed in this chapter is under continuous development, and the account here only reflects the state of mid-2006, see `http://omdoc.org/projects/` for more and current information.

26.1.1 Overview

The OMDoc format as a whole and the applications mentioned above are supported by a variety of tools for creating, manipulating, and communicating OMDoc documents. We can distinguish four kinds of tools:

Interfaces for Mathematical Software Systems like automated theorem provers. These system are usually add-ins that interpret the internal representation of formalized mathematical objects in their host systems and recursively generate formal OMDoc documents as output and communication

[1] If your OMDoc project is not represented here, please contact `m.kohlhase@jacobs-university.de` to arrange for inclusion in later editions of this book.

streams. Some of these systems also have input filters for OMDoc like the ✓eriFun described in Section 26.20, but most rely on the OMDoc transformation to their native input syntax described in Section 25.2.

Invasive Editors i.e. are add-ins or modes that "invade" common general-purpose editing systems and specialize them to deal with the OMDoc format. The OMDoc mode for the Emacs editor presented in Section 26.16, the **C**Point add-in for MS PowerPoint (Section 26.14), the Mathematica® notebook converter (Section 26.17), the Sentido plugin for Mozilla-based browsers, and the plugin for TEX$_{\text{MACS}}$ (Section 26.19) are examples for this kind of editor. They differ from simple output filter in providing editing functionality for OMDoc specific information.

Human-Oriented Frontend Formats for instance the QMath project described in Section 26.2 defines an interface language for a fragment of OMDoc, that is simpler to type by hand, and less verbose than the OMDoc that can be generated by the `qmath` parser. STEX defines a human-oriented format for OMDoc by extending the TEX/LATEX with content markup primitives, so that it can be transformed to OMDoc. See Section 26.15 for details.

Mathematical Knowledge Bases The MBase and Maya systems described in Sections 26.4 and 26.12 are web-based mathematical knowledge bases that offer the infrastructure for a universal, distributed repository of formalized mathematics represented in the OMDoc format.

26.1.2 Application Roles of the OMDoc Format

The applications above support the utilization of the OMDoc format in several roles. Generally, OMDoc can used of as a

Communication Standard between mechanized reasoning systems.

Data Format for Controlled Refinement from informal presentation to formal specification of mathematical objects and theories. Basically, an informal textual presentation can first be marked up, by making its structure explicit (classifying text fragments as definitions, theorems, proofs, linking text, and their relations), and then formalizing the textually given mathematical knowledge in logical formulae (by adding `FMP` elements; see Chapter 14).

Document Preparation Language. The OMDoc format makes the large-scale document- and conceptual structures explicit and facilitates maintenance on this level. Individual documents can be encoded as lightweight narrative structures, which can directly be transformed to e.g. XHTML+MathML or LATEX, which can in turn be published on the Internet.

Basis for Individualized (Interactive) Documents. Personalized narrative structures can be generated from MBase content making use of the conceptual structure encoded in MBase together with a user model. For instance,

the MMiSS, MathDox, and ActiveMath projects described in Sections 26.6 to 26.8 use the OMDoc infrastructure in an educational setting. They make use of the content-orientation and the explicit structural markup of the mathematical knowledge to generate on the fly specialized learning materials that are adapted to the students prior knowledge, learning goals, and notational tastes.

Interface for Proof Presentation. As the proof part of OMDoc allows small-grained interleaving of formal (`FMP`) and textual (`CMP`) presentations in multiple languages (see e.g. [HF97, Fie99]).

26.2 QMath: A Human-Oriented Language and Batch Formatter for OMDoc

Project Home	`http://www.matracas.org/qmath/index.en.html`
Authors	Alberto González Palomo Toledo, Spain[2]

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

The "Q" was intended to mean "quick", since QMath began in 1998 as an abbreviated notation for MathML. The first version (0.1) just expanded the formulas found enclosed by "$" signs, which were abbreviated forms of the MathML element names, and added the extra markup such as `<mrow>` and the like. The second (0.2) did the same thing, but this time allowing an algebraic notation that was fixed in the source code. Finally, version 0.3 allowed the redefinition of symbols while parsing, but it was not capable of expanding formulas embedded in XML documents like the previous ones did until version 0.3.8.[3] For a more detailed history see [GPb].

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

The grouping of symbols in the context files is arbitrary. However, the ones included with QMath follow the OpenMath Content Dictionaries hierarchy so that, for instance, the English language syntax for the symbols in the "arith1" CD is defined in the context "Mathematics/OpenMath/arith1".

Figure 26.1 shows a minimal QMath document, and the OMDoc document generated from it. The first line ("`QMATH 0.3.8`") in the QMath document is required for the parser to recognize the file. The lines beginning with "`:`" are metadata items, the first of which, `:en`, declares the primary language for the document, in this case English. Specifying the language is required, as it sets the basic keywords accordingly, and there is no default (privileged) language in QMath. For example, the English keyword "`Context`" is written "`Contexto`" if the language is Spanish. (Similarly, the arithmetic context is "`Matemáticas/Aritmética`"). Then, the "OMDoc" context is loaded, defining the XML elements to be produced by the metadata items and the different kinds of paragraphs: plain text, theorem, definition, proof, example, etc.

[2] The author is currently employed part-time in the ActiveMath project, developed by Saarland University and the DFKI, but this work was done on his own, without their supervision or support.

[3] This offers an alternative to the OQMath wrapper mentioned in Section 26.8.

```
QMATH 0.3.8
:en
Context: "Mathematics/OMDoc"

:"Diary"
:W. Smith
:1984-04-04 18:43:00+00:00

Context: "Mathematics/Arithmetic"

Theory:[<-thoughtcrime]

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

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

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

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

Symbol: = OP_EQ "relation1:eq"
Symbol: neq OP_EQ "relation1:neq"
Symbol: ¬= OP_EQ "relation1:neq"
Symbol: ≠ OP_EQ "relation1:neq"

...
```

```
<?xml version='1.0' encoding='UTF-8' standalone='no'?>
<!DOCTYPE omdoc PUBLIC "-//OMDoc//DTD OMDoc V1.2//EN"
                       " ../../../../ dtd/omdoc.dtd">
<omdoc xmlns='http://www.mathweb.org/omdoc' version='1.2'
       xmlns:dc='http://purl.org/dc/elements/1.1/'>
 <metadata>
  <dc:language>en</dc:language>
  <dc:title>Diary</dc:title>
  <dc:creator role='aut'>W. Smith</dc:creator>
  <dc:date>1984-04-04T18:43:00+00:00</dc:date>
 </metadata>
 <theory xml:id='thoughtcrime'>
  <imports from="arith1"/>
  <imports from="relation1"/>
  <omtext>
   <metadata><dc:title>Down with Big Brother</dc:title></metadata>
   <CMP>
    Freedom is the freedom to say
    <OMOBJ xmlns='http://www.openmath.org/OpenMath'>
     <OMA>
      <OMS cd='relation1' name='eq'/>
      <OMA>
       <OMS cd='arith1' name='plus'/>
       <OMI>2</OMI><OMI>2</OMI>
      </OMA>
      <OMI>4</OMI>
     </OMA>
    </OMOBJ>.
    If that is granted, all else follows.
   </CMP>
  </omtext>
 </theory>
</omdoc>
```

Fig. 26.1. A minimal QMath document (top left) and its OMDoc result (bottom). Some symbol definitions are displayed in the top right.

After that setup come the document title, author name (one line for each author), and date, which form the content of the OMDoc metadata element.

The document is composed of paragraphs (which can be nested) separated by empty lines, and formulas are written enclosed by "$" signs.

There is an Emacs mode included in the source distribution, that provides syntax highlighting and basic navigation based on element identifiers.

It is also possible to use it on an XML document for expanding only the mathematical expressions. QMath will detect automatically the input format, either QMath text or XML, and in the later case output everything verbatim except for the QMath language fragments found inside the XML processing instructions of the form `<?QMath ... ?>` and the mathematical expressions between "$".

```
<?xml version='1.0' encoding='UTF-8' standalone='no'?>
<!DOCTYPE omdoc PUBLIC "-//OMDoc//DTD OMDoc V1.2//EN"
                       " ../../../../ dtd/omdoc.dtd">
<?QMath
:en
Context: "Mathematics/Arithmetic"
?>
<omdoc xmlns='http://www.mathweb.org/omdoc' version='1.2'
       xmlns:dc='http://purl.org/dc/elements/1.1/'>
 <metadata>
  <dc:language>en</dc:language>
  <dc:title>Diary</dc:title>
  <dc:creator role='aut'>W. Smith</dc:creator>
  <dc:date>1984-04-04T18:43:00</dc:date>
 </metadata>
 <theory xml:id='thoughtcrime'>
  <omtext>
   <metadata><dc:title>Down with Big Brother</dc:title></metadata>
   <CMP>
    Freedom is the freedom to say $2+2=4$.
    If that is granted, all else follows.
   </CMP>
  </omtext>
 </theory>
</omdoc>
```

Fig. 26.2. The same example document, using QMath only for the formulas.

While QMath was a good improvement over manual typing of the OMDoc XML, it does not scale well: in real documents, with more than a couple of nesting levels, it is difficult to keep track of where the current paragraph belongs.

One solution is to use it only for the mathematical expressions, and rely on some XML editor for the document navigation and organization, such as the OMDoc mode for Emacs described in Section 26.16 or the OQmath mode for jEdit in Section 26.9. Another is to use the Sentido browser/editor in Section 26.3, which reimplements and extends QMath's functionality.

QMath is Free Software distributed under the GNU General Public License (GPL [Fre91]).

26.3 Sentido: an Integrated Environment for OMDoc

Project Home	`http://www.matracas.org/sentido/index.en.html`
Authors	Alberto González Palomo Toledo, Spain[4]

SENTIDO is an integrated environment for browsing, searching, and editing collections of OMDOC documents. It is implemented as an extension for the MOZILLA/FIREFOX browsers to avoid the biggest problems found when using QMATH: the need to compile the program for installing, the batch mode of interaction that made small corrections consume much of the author's time, and the lack of any support for document navigation and search.

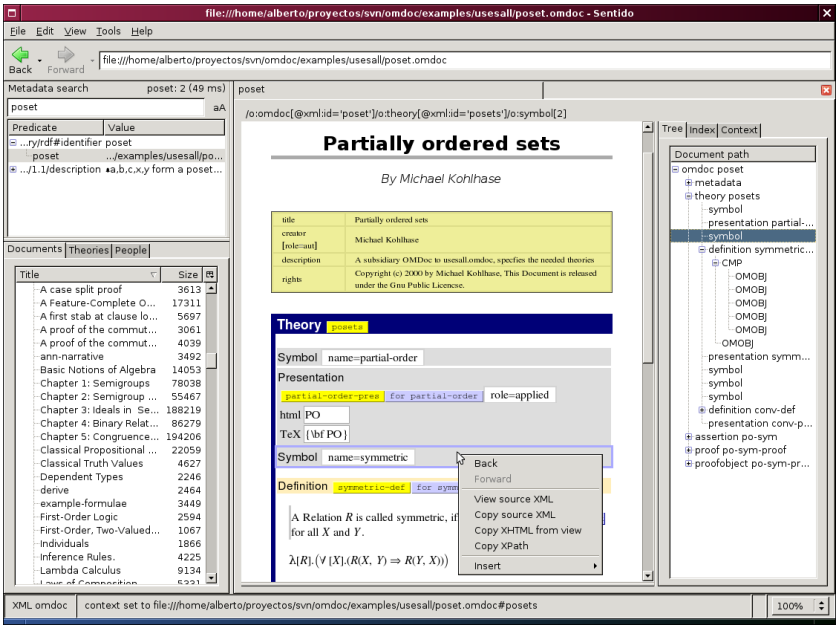


Fig. 26.3. SENTIDO after indexing the OMDoc repository in the library (left) and loading a document from it (center and right).

Figure 26.3 shows a typical session initiated by searching in the document library (described below in more detail) and opening one of the results. The context menu displays the options for browsing back and forward, viewing the XML (OMDOC) source of the selected element, copying it to the system clipboard, copying its MATHML rendering or an XPATH expression that identifies it, and inserting new elements.

[4] The author is currently employed part-time in the ACTIVEMATH project, developed by Saarland University and the DFKI, but this work was done on his own, without their supervision or support.

26.3.1 The User Interface

The window is made to resemble the web browser, and consists of two main panes: the smaller one on the left contains the interface for the "document library", and the right one the "document view" and associated information like the document tree, element identifiers index, and context at the current cursor position.

The document library is a knowledge base about documents, the theories defined in them, and people mentioned in their metadata as authors, editors, translators, etc. It is implemented as an RDF store with the documents organized in collections called "volumes" with references to documents, so that different volumes can have documents in common. The tabs labelled "Documents", "Theories" and "People" display different views of the library content.

The bibliographic data for each document is stored using the Bibliographic Record Schema [Len04], which includes FOAF[5] entries for people.

The documents in the library are indexed by the search engine, which stores their metadata entries and theory identifiers in an abridged inverted index to speed up the searches to the point where "search as you type" becomes possible[6]. The search pattern accepts regular expression syntax, as shown in Figure 26.4.

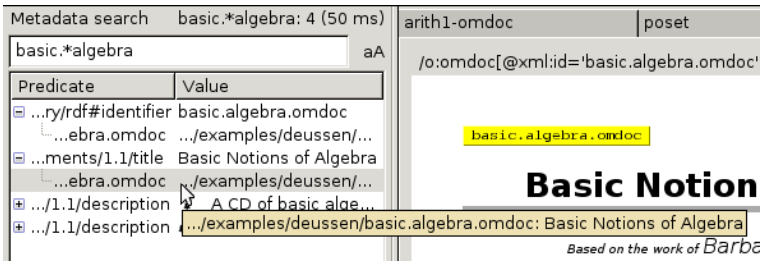


Fig. 26.4. Metadata Search in SENTIDO: tooltips show the content of the cropped entries.

The document view is built using XHTML + MATHML that can be edited normally, with the changes being propagated to the internal OMDOC.

The view is built on demand (using XSLT) as the subparts of the document are unfolded in the document navigation tree found in the right part of the window. This has been found important in practice since many real

[5] "Friend of a friend", described in their web page `http://www.foaf-project.org` as being "about creating a Web of machine-readable homepages describing people, the links between them and the things they create and do."

[6] On the author's 1 GHz laptop computer, the search times in a library of around two thousand documents are usually between 100 and 200 milliseconds.

uses of OMDoc involve documents that contain large lists of elements, like exercises, that are largely independent of each other and thus do not usually require being viewed at the same time, and the biggest delay in opening a medium to large sized document was by far the display of the XHTML view. Another motivation for this approach is to progress towards handling the source document more like a database, and customize its presentation for the task at hand.

Sentido adds some options to the context menu in the browser, to allow the user to open links to OMDoc files from web pages (see Figure 26.5).

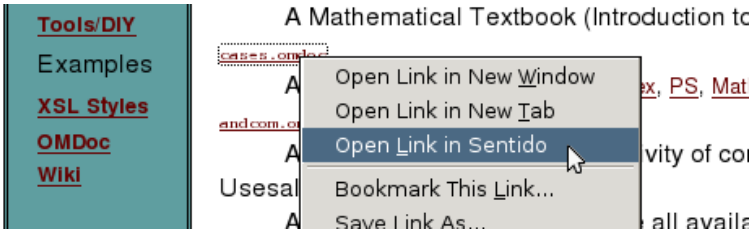


Fig. 26.5. Mozilla's Context Menu after Installing Sentido.

26.3.2 Formula Editing

Mathematical expressions are entered using a selectable linear syntax, translated by a new version of the QMath parser described in Section 26.2. This is a much more capable implementation based on finite-state cascades [Abn96].

There are five grammars included in the install package, that are used for translating back and forth between OpenMath and the linear syntax of QMath and the Computer Algebra Systems Maxima, Yacas, Maple™ and Mathematica®. More syntaxes can be added by writing new grammars, with a format similar to QMath "context" files.

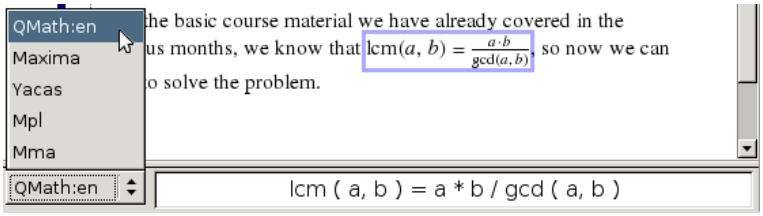


Fig. 26.6. The formula editor under the document view, with the input syntax menu and the text field where the formula is typed, which updates continuously the internal OpenMath representation and the MathML view.

When the cursor enters a formula, the linear input field appears at the bottom of the document view, as seen in Figure 26.6. It contains a text field for editing, and a menu button for selecting the syntax, which can be done at any moment: the linear expression is regenerated instantaneously from its OpenMath form, so it is possible to enter a formula using, for instance, Mathematica® syntax, then select another syntax such as Maple™, and get the expression immediately translated, going through its OpenMath representation (Figure 26.7).

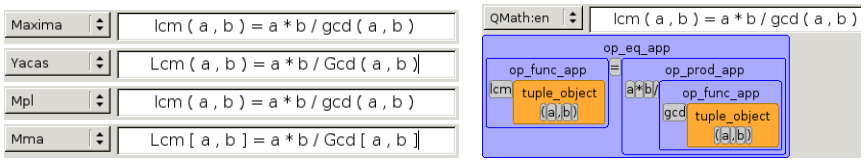


Fig. 26.7. The formula is translated by Sentido each time the user selects another syntax (left, the vertical line is the blinking caret), and it is possible to view the parse tree (right), updated as the input is modified.

Insertion of formulae is achieved by typing the dollar symbol "$", which produces an empty formula readily editable so that the sequence of keystrokes is similar to typing TEX or QMath text: one can type `$e^(pi*i)+1=0$` and get $e^{\pi i}+1=0$ without having to look at the formula editor or use the mouse. The changes as the formula is being modified are stored, and the display updated from the OpenMath form, at each point when there is a complete parse of the formula. This gives immediate feedback on how the program understands the input.

An important difference is that there is no need to care about "context files" any more. In QMath, specifying a "context" had a double function: putting symbols in scope for disambiguation, and selecting a notation style for them. Those aspects are separated in Sentido: the in-scope symbols are automatically determined from the enclosing theory and those imported from it (recursively), and the notation is selectable by the user.

Note that the parser allows any characters supported by the browser rendering engine of Mozilla/FireFox (a big subset of Unicode), not just ASCII. For example, the number 3.14159265... can be entered either as π or with an ASCII form depending on the selected syntax: "pi" for QMath, "%pi" for Maxima, or "Pi" for Yacas, Maple™ and Mathematica®.

26.3.3 Future Work and Availability

Sentido is a long term personal project that has been in development for several years (since 2004), entirely in the author's spare time and using his

own computing resources, based on experiments[7] and notes collected during the development of QMath. Therefore, we expect it to continue developing during the foreseeable future unless a better application appears that makes it redundant.

Its components are designed to be reusable, which is tested from time to time by producing spin-off applications that use subsets of its functionality in a self-contained way. One example is the small Computer Algebra System called Algebra [GPa], that contains parts of Sentido such as the new parser combined with specific ones like the function plotter and the term rewriting engine.

Future developments will focus on what we consider the two main tasks for a development environment for semantic encoding of mathematical content:

- Ease the tedium of writing all the details needed for an unambiguous encoding of the content. This is where the flexible input parser comes into play: having a syntax redefinable at any point in the content simplifies the expression input, as the syntax can be adapted to the context in which an expression occurs.
- Provide some benefit once we have the semantic encoding which would not be present with an ambiguous encoding such as TEX. Here we need to implement detailed checking and strong search capabilities. A next step would be to assist the writing process by inferring new content and informing the input interface about the context as mentioned above.

Some planned improvements in Sentido are:

- Make the browser open OMDoc documents linked from normal pages directly in Sentido, by implementing a stream handler for the `MIME` type `application/omdoc+xml`.
- Integrate Algebra into Sentido, to add automated symbolic manipulation to the document editing process.
- Extend the checking being done on the theories: at the time of writing these lines, only the theory import relations are checked for loops and unknown theory references, which was already enough to locate several mistyped theory identifiers in the OMDoc repository.
- Implement useful features found in other projects such as Theorema [PB04]. This is strongly related to the two points above since Theorema implements many features needed for the task of content checking which are still missing in Sentido, and some of them are available in proof-of-concept form in Algebra.

Sentido is Free Software distributed under the GNU General Public License (GPL [Fre91]).

[7] Some of those early experiments with Mozilla inspired work done on adapting OpenOffice and TEXMACS for OMDoc in collaboration with George Goguadze [GGP03]

26.4 MBase, an Open Mathematical Knowledge Base

Project Home	`http://www.mathweb.org/mbase`
Authors	Andreas Franke[1], Michael Kohlhase[2] [1] Computer Science, Saarland University [2] School of Engineering and Science, International University Bremen

We describe the MBase system, a web-based mathematical knowledge base. It offers the infrastructure for a universal, distributed repository of formalized mathematics. Since it is independent of a particular deduction system and particular logic, the MBase system can be seen as an attempt to revive the Qed initiative [[QED96]]$^{84}_{a}$ from an infrastructure viewpoint. See [KF01] for the logical issues related to supporting multiple logical languages while keeping a consistent overall semantics. The system is realized as a mathematical service in the MathWeb system [FK99, Zim04], an agent-based implementation of a mathematical software bus for distributed mathematical computation and knowledge sharing. The content language of MBase is OMDoc.

Err(84)

We will start with a description of the system from the implementation point of view (we have described the data model and logical issues in [KF01]).

The MBase system is realized as a distributed set of MBase servers (see figure 26.8). Each MBase server consists of a Relational Data Base Management System (RDBMS) connected to a mozart process (yielding a MathWeb service) via a standard data base interface. For browsing the MBase content, any MBase server provides an `http` server (see [MBa] for an example) that dynamically generates presentations based on HTML or XML forms.

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

[84] Erratum! reference to QED (added text)

for specialized searching (see [Duc98] for a related system and the origin of this idea).

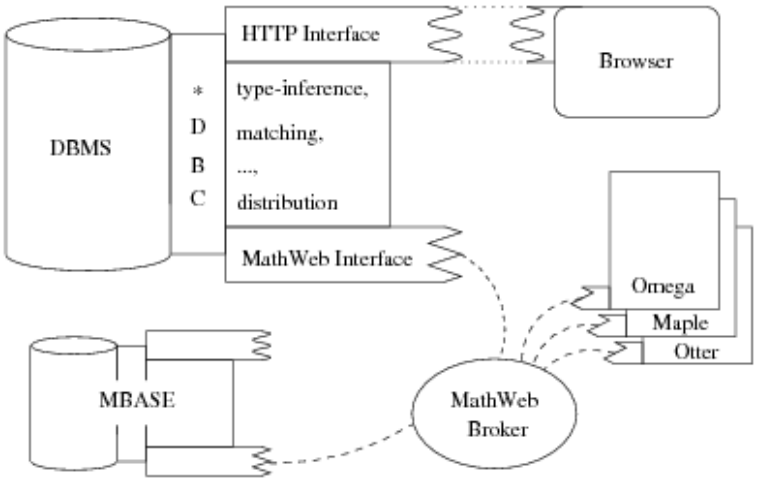


Fig. 26.8. System Architecture

MBase supports transparent distribution of data among several MBase servers (see [KF01] for details). In particular, an object O residing on an MBase server S can refer to (or depend on) an object O' residing on a server S'; a query to O that needs information about O' will be delegated to a suitable query to the server S'. We distinguish two kinds of MBase servers depending on the data they contain: *archive servers* contain data that is referred to by other MBases, and *scratch-pad* MBases that are not referred to. To facilitate caching protocols, MBase forces archive servers to be *conservative*, i.e. only such changes to the data are allowed, that the induced change on the corresponding logical theory is a conservative extension. This requirement is not a grave restriction: in this model errors are corrected by creating new theories (with similar presentations) shadowing the erroneous ones. Note that this restriction does not apply to the non-logical data, such as presentation or description information, or to scratchpad MBases making them ideal repositories for private development of mathematical theories, which can be submitted and moved to archive MBases once they have stabilized.

26.5 A Search Engine for Mathematical Formulae

Project Home	`http://search.mathweb.org/`
Authors	Ioan Sucan, Michael Kohlhase Computer Science, International University Bremen

As the world of information technology grows, being able to quickly search data of interest becomes one of the most important tasks in any kind of environment, be it academic or not. We present a search engine for mathematical formulae. The MathWebSearch system harvests the web for content representations of formulae (currently MathML and OpenMath) and indexes them with substitution tree indexing, a technique originally developed for accessing intermediate results in automated theorem provers. For querying, we present a generic language extension approach that allows to construct queries by minimally annotating existing representations.

Generally, searching for mathematical formulae is a non-trivial problem — especially if we want to be able to search occurrences of the query term as sub-formulae — for the following reasons:

1. *Mathematical notation is context-dependent*. For instance, binomial coefficients can come in a variety of notations depending on the context: $\binom{n}{k}$, ${}_nC^k$, C^n_k, and C^k_n all mean the same thing: $\frac{n!}{k!(n-k)!}$. In a formula search we would like to retrieve all forms irrespective of the notations.
2. *Identical presentations can stand for multiple distinct mathematical objects*, e.g. an integral expression of the form $\int f(x)dx$ can mean a Riemann Integral, a Lebesgue Integral, or any other of the 10 to 15 known anti-derivative operators. We would like to be able to restrict the search to the particular integral type we are interested in at the moment.
3. *Certain variations of notations are widely considered irrelevant*, for instance $\int f(x)dx$ means the same as $\int f(y)dy$ (modulo α-equivalence), so we would like to find both, even if we only query for one of them.

To solve this formula search problem, we concentrate on *content representations of mathematical formulae*, since they are presentation-independent and disambiguate mathematical notions.

A Running Example: The Power of a Signal

A standard use case for MathWebSearch is that of an engineer trying to solve a mathematical problem such as finding the power of a given signal $s(t)$. Of course our engineer is well-versed in signal processing and remembers that a signal's power has something to do with integrating its square, but has forgotten the details of how to compute the necessary integrals. He will therefore call up MathWebSearch to search for something that looks like $\int_?^? s^2(t)dt$ (for the concrete syntax of the query see Listing 26.1 in Section 26.5). MathWebSearch finds a document about Parseval's Theorem, more specifically

$\frac{1}{T}\int_0^T s^2(t)dt = \Sigma_{k=-\infty}^{\infty} \mid c_k \mid^2$ where c_k are the Fourier coefficients of the signal. In short, our engineer found the exact formula he was looking for (he had missed the factor in front and the integration limits) and moreover a theorem he may be able to use.

Indexing Mathematical Formulae

For indexing mathematical formulae on the web, we will interpret them as first-order terms. This allows us to use a technique from automated reasoning called *term indexing* [Gra96]. This is the process by which a set of terms is stored in a special purpose data structure (the **index**) where common parts of the terms are potentially shared, so as to minimize access time and storage. The indexing technique we work with is a form of tree-based indexing called *substitution-tree indexing*. A substitution tree, as the name suggests, is simply a tree where substitutions are the nodes. A term is constructed by successively applying substitutions along a path in the tree, the leaves represent the terms stored in the index. Internal nodes of the tree are **generic terms** and represent similarities between terms.

The main advantage of substitution tree indexing is that we only store substitutions, not the actual terms, and this leads to a small memory footprint. Adding data to an existing index is simple and fast, querying the data structure is reduced to performing a walk down the tree. Index building is done in similar fashion to [Gra96]. Once the index is built, we keep the actual term instead of the substitution at each node, so we do not have to recompute it with every search. At first glance this may seem to be against the idea of indexing, as we would store all the terms again, not only the substitutions. However, due to the tree-like structure of the terms, we can in fact store only a pool of (sub)terms and define the terms in our index using pointers to elements of the pool (which are simply other terms). To each of the indexed terms, a data string is attached — a string that represents the exact location of the term. We use XPointer [GMMW03] to specify this.

Unfortunately, substitution tree indexing does not support subterm search in an elegant fashion, so when adding a term to the index, we add all its subterms as well. This simple trick works well: the increase in index size remains manageable and it greatly simplifies the implementation.

A Query Language for Content Mathematics

When designing a query language for mathematical formulae, we have to satisfy a couple of conflicting constraints. The language should be content-oriented and familiar, but it should not be specialized to a given content representation format. Our approach to this problem is to use a simple, generic extension mechanism for XML-based representation formats rather than a genuine query language itself. The query extension is very simple, it adds two new attributes to the respective languages: `mq:generic` and `mq:anyorder`,

where the prefix `mq:` abbreviates the namespace URI `http://mathweb.org/MathQuery/` for MathWebSearch.

In this way, the user need not master a new representation language, and we can generate queries by copy and paste and then make parts of the formulae generic by simply adding suitable attributes. We will use Content MathML [ABC+03a] in the example, but MathWebSearch also supports OpenMath and a shorthand notation that resembles the internal representation we are using for terms (prefix notation). The `mq:generic` attribute takes string values and can be specified for any element in the query, making it into a (named) query variable: its contents are ignored and it matches any term in the search process.

While of searching expressions of the form $A = B$, we might like to find occurrences of $B = A$ as well. At this point the `mq:anyorder` attribute comes in. Inside an `apply` tag, the first child defines the function to be applied; if this child has the attribute `mq:anyorder` defined with the value "yes", the order of the subsequent children is ignored. If we do not want to specify the function name, we can use the `mq:generic` attribute again, but this time for the first child of an `apply` tag. Given the above, the query of our running example has the form presented in Listing 26.1. Note that we do not know the integration limits or whether the formula is complete or not.

Listing 26.1. Query for Signal Power

```
<math xmlns="http://www.w3.org/1998/Math/MathML"
      xmlns:mq="http://mathweb.org/MathQuery">
  <apply><int/>
    <domainofapplication mq:generic="domain"/>
    <bvar> <ci mq:generic="time"/> </bvar>
    <apply><power/>
      <apply><ci mq:generic="fun"></ci><ci mq:generic="time"/></apply>
      <cn>2</cn>
    </apply>
  </apply>
</math>
```

Input Processing

MathWebSearch can process any kind of XML representation for content mathematics. The system is modular and provides an interface which allows to add XML-to-index-term transformers. . We will discuss input processing for Content-MathML.

Given an XML document, we create an index term for each of its `math` elements. Consider the example on the right: We have the standard mathematical notation of an equation (1), its

1) Mathematical expression: $f(x) = y$

2) Content MathML:

```
<apply><eq/>
  <apply>
    <ci>f</ci>
    <ci>x</ci>
  </apply>
  <ci>y</ci>
</apply>
```

3) Term representation: $eq(f(x), y)$

Content MathML representation (2), and the term we extract for indexing (3). As previously stated, any mathematical construct can be represented in a similar fashion.

Search modulo α-renaming becomes available via a very simple input processing trick: during input processing, we add a `mq:generic` attribute to every bound variable (but with distinct strings for different variables). Therefore in our running example the query variable t (`@time` in Listing 26.1) in the query $\int_?^? s^2(t)dt$ is made generic, therefore the query would also find the variant $\frac{1}{T}\int_0^T s^2(x)dx = \Sigma_{k=-\infty}^{\infty} \mid c_k \mid^2$.

Result reporting

For a search engine for mathematical formulae we need to augment the set of result items (usually page title, description, and page link) reported to the user for each hit. As typical pages contain multiple formulae, we need to report the exact occurrence of the hit in the page. We do this by supplying an XPointer reference where possible. Concretely, we group all occurrences into one page item that can be expanded on demand and within this we order the groups by number of contained references. For any given result, a detailed view is available. This view shows the exact term that was matched (using Presentation MathML) and the used substitution (a mapping from the query variables specified by the `mq:generic` attributes to certain subterms) to match that specific term.

Case Studies and Results

We have tested our implementation on the content repository of the ConneXions Project, available via the OAI protocol [OAI02]. This gives us a set of over 3,200 articles with mathematical expressions to work on. The number of terms represented in these documents is approximately 53,000 (77,000 including subterms). The average term depth is 3.6 and the maximal one is 14. Typical query execution times on this index are in the range of milliseconds. The search in our running example takes 14 ms for instance. There are, however, complex searches (e.g. using the `mq:anyorder` attribute) that internally call the searching routine multiple times and take up to 200 ms but for realistic examples execution time is below 50 ms. We are currently building an index of the 86,000 Content MathML formulae from `http://functions.wolfram.com`. Here, term depths are much larger (average term depth 5.7, maximally around 50) resulting in a much larger index: it is just short of 2 million formulae. First experiments indicate that search times are largely unchanged by the increase in index size.

In the long run, it would be interesting to interface MathWebSearch with a regular web search engine and create a powerful, specialized, full-feature

application. This would resolve the main disadvantage our implementation has – it cannot search for simple text. A simple socket-based search API allows to integrate MathWebSearch into other content-based mathematical software systems.

26.6 Semantic Interrelation and Change Management

Project Home	`http://www.mmiss.de`
Authors	Bernd Krieg-Brückner, Achim Mahnke Computer Science, University of Bremen, Germany

The corpus of electronically available mathematical knowledge increases rapidly. Usually, mathematical objects are embedded in and related to different kinds of documents like articles, books, or lecture material, the domain of which can be different from mathematics, e.g., engineering or computer science. Therefore, maintaining high-quality mathematical knowledge becomes a non-trivial engineering task for teams of authors.

In this scenario, sharing and reuse is the key to efficient development. Unfortunately, while there has been a large body of research concerning the sharing and reuse of program developments, sharing and reuse of documents has until now been mainly done by little more than cut and paste. However, to ensure sustainable development, i.e. continuous long-term usability of the contents, sharing and reuse needs to be supported by tools and methods taking into account the semantic structure of the document. In developing these methods and tools we can benefit from the experience in and associated support tools.

We address this problem by providing a methodology to specify coherence and consistency of documents by interrelation of semantic terms and structural entities, supported by a tool for fine-grained version control and configuration management including change management. Semantic interrelation explicates the meaning lying behind the textual content, and relates the semantic concepts within and across documents by means of an ontology. To allow change management, each document is structured in-the-small. Each document corresponds to a package, and packages may be structured in-the-large using folders and import relations. The ideas and methods explained here have been developed in the MMiSS project which aimed at the construction of a multi-media Internet-based adaptive educational system (see [**?**, KBLL+04, KBKB+04]).

26.6.1 Semantic Interrelation Via Ontologies

Ontologies provide the means for establishing a semantic structure. An ontology is a formal explicit description of concepts in a domain of discourse. The MMiSSLaTeX package for ontologies provides a set of easy-to-use macros for the declaration of ontologies in LaTeX documents. They are used to *declare* the ontology of semantic terms used in a document, in a prelude up front. This *specification* of the document contains at least a rigorous hierarchical structure of the terminology (a taxonomy, the *signature* of the document), and may be seen as an elaborate index structure. Moreover, relations between terms may be defined for more semantic interrelation.

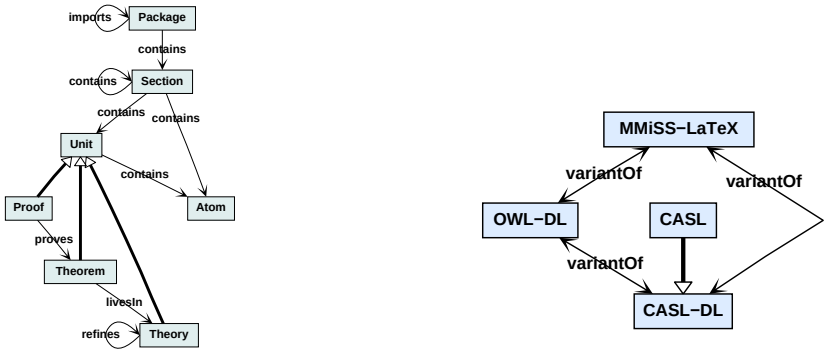


Fig. 26.9. (a) Parts of the System's Ontology (b) Formalism variants

The ontology serves a dual purpose — just as the specification of an abstract data type in program development: it specifies the content to be expected in the body of the document in an abstract, yet precise, manner — the content developers requirement specification; and it specifies the content for reference from the outside — the user's perspective, who may then view the body of the document as a black box. The content developer will use the MMiSSLaTeX `Def` command to specify the *defining* occurrence of a promised term, as for an index. Using the structuring in-the-large facilities via packages, the external user may then refer between documents using various kinds of *reference* commands, as the content developer may within a document.

The next section will show, how we can explore this domain ontology — supplied by the author — in order to capture semantic relations between document parts and use these relations for supporting a management of change for mathematical documents.

26.6.2 Change Management

The notion of *change management* is used for the maintenance and preservation of consistency and completeness of a development during its evolution. More precisely, we want to have a consistent configuration in which all versions are compatible and there are no cyclic definitions or proofs. At the same time, it should be a complete configuration: there should be no dangling forward references.

Such notions are well-known for formal languages. In contrast, natural language used for writing teaching material does not usually possess a well-defined semantics, and the notion of consistency is arguable. Different authors may postulate different requirements on the material in order to regard it as being consistent. The existence of a user-defined ontology helps a great deal to check references. However, we can make even better use of the information contained in the ontology.

The System's Ontology

The aim is to allow change management with regard to consistency and completeness requirements defined by the user in terms of an ontology. In order to unify this approach with the structural consistency and completeness properties introduced above, we express the document structure, originally defined by a document type definition, as an ontology, the so-called *System's Ontology* (see Fig. 26.9a). It defines the following relations between structural elements of documents:

`comprises` An obvious structuring mechanism is nesting of individual parts of a document, leading to the contains relation. The contains relation is part of a family of `comprises` relations that share common properties.

`reliesOn` A family of `reliesOn` relations reflects the various dependencies between different parts of a document. For example, a theorem *lives in* a theory, or proof *proves* a theorem.

`pointsTo` The family of `pointsTo` relations is very similar, and relates references with the defining occurrence of a semantic term.

`variantOf` Another structuring relation is introduced by variants. Parts of a document may e.g. be written in various languages which gives rise to a `variantOf` relation between these document parts and their constituents; it is an equivalence relation.

It is now rather straightforward to formulate consistency and completeness rules in terms of invariants of these relations. Formulating these invariants as formal rules will enable us to implement a generic and flexible change management that keeps track of the invariants and informs the user about violations when a previously consistent document has been revised, leading to various kinds of error (e.g. for `reliesOn` relations) or warning messages (e.g. for `pointsTo` relations).

Properties of Interactions between Structuring Mechanisms.

This approach also allows us to lift relations to structuring mechanisms allowing more modular and localized change management. For example, relating the `comprises` and `reliesOn` relations allows us to formalize invariants regarding the closure of document parts with respect to the `reliesOn` relation: We can require that there is a proof for each theorem in a package. Furthermore, if two structural entities are related by `reliesOn`, their relation is propagated along the `comprises` relation towards the root of the hierarchy of nested structural entities, such that (for a theorem T a proof P, and sections A, B):

B contains P & A contains T & P proves $T \Rightarrow B$ `reliesOn` A.

If the user changes section A, the repository will only need to check all sections that A relies on (such as B here) for invariants, and not the whole

document. However, in contrast to formal developments as in e.g. the MAYA system [AH05], there is no rigorous requirement that a document should obey all the rules. There may be good reasons, for instance, to present first a "lightweight" introduction to all notions introduced in a section before giving the detailed definitions. In this particular case, one would want to introduce forward pointers to the definitions rather than making the definitions rely on the introduction; thus the rules are covered.

In any case, the more structure there is, the better the chances are for preserving consistency and completeness; any investment in introducing more `reliesOn` relations, for example, will pay off eventually. The change management will observe whether revisions by the user will affect these relations and, depending on the user's preferences, emit corresponding warnings.

The aim is to allow users to specify individual notions of consistency by formulating the rules that the relations should obey. This should be possible for the relations between the particular (predefined) structuring mechanisms, but also in general between semantic terms of the user's own ontology. Our work in this direction will rely on the methods and tools provided by the HETS system (see Section 26.13).

26.6.3 Variants

The concept of variants adds a new dimension to hierarchically structured documents. The idea is to maintain and manage different variants of structural entities (document subtrees) which represent the same information in different ways — variants are meant to glue them together.

Managing different natural language variants in parallel is an obvious example. Another one is the formalism variant which denotes the particular formalism in which a formal content part like a theorem or a definition is expressed. Considering ontology development itself, for example, we propose to use variants to maintain different formal representations for the same semantic concept together with its documentation. Figure 26.9b shows the possible variants for declaring ontology components (see [MTea04] for details).

The MMiSS repository provides functions to store and retrieve these structural variants by means of specifications for selecting particular variants for editing or presentation.

26.6.4 Relations to OMDoc

OMDoc provides modules for marking up the knowledge structure and the narrative structure of mathematical documents. MMiSS combines these two viewpoints by giving means for structuring the document contents (which constitutes the narrative structure) and for specifying the incorporated knowledge by use of ontologies. Therefore, we have implemented an export of MMiSS documents to (content and narrative) OMDoc documents and vice versa.

26.7 MathDox: Mathematical Documents on the Web

Project Home	`http://www.mathdox.org`
Authors	A.M. Cohen, H. Cuypers, E. Reinaldo Barreiro Department of Mathematics and Computer Science, Eindhoven University of Technology

Abstract

The MathDox system provides an infrastructure for interactive mathematical documents that make use of the World Wide Web. These documents take input from various sources, users, and mathematical services. Communication between these different entities can be realized using OpenMath. But, such communication and the interactivity inside the mathematical document take place in a specific, dynamic context. In this paper we discuss our approach to such a dynamic mathematical context: MathDox. It consists of both an XML-based markup language for interactive mathematical contents and a set of software tools realizing the interactivity. cl

26.7.1 Introduction

Although the notion of an interactive mathematical document has been around for several years, cf. [CM98], its realization is nowhere near the final stage. For instance, recent progress in web technologies has enabled a much smoother communication of mathematics than ever before. The use of an interactive mathematical document (IMD) can provide a window to the world of mathematical services on the Internet, and a mathematical service on the Internet can be created by the building of an interactive mathematical document. MathDox is an ensemble of software tools for creating IMDs, it includes

1. an XML based language that offers markup support for the source texts of IMDs;
2. a document server, rendering interactive mathematical documents from source text and interactively obtained information;
3. mathematical services, providing connections with CASs like Mathematica® and GAP via OpenMath phrasebooks (cf. [OM]).

The creation of MathDox is a project at the Technische Universiteit Eindhoven (the RIACA institute). Several people at RIACA have helped creating it; here we mention Manfred Riem, Olga Caprotti, Hans Sterk, Henny Wilbrink, Mark Spanbroek, Dorina Jibetean. The system is mainly built with Java and related technology. The products are available via the project web site and will be licensed under the Lesser Gnu Public License [Fre99].

26.7.2 The Language

The MathDox source is an XML document. We have derived our own document type definitions (DTD) for these source texts. We have been influenced by both DocBook [WM99] and OMDoc. The former is a fairly general standard for electronic books, the latter is a very rich, and strongly logic-oriented standard for mathematical documents—the main subject of this book. Both OMDoc and MathDox use OpenMath [BCC+04], the difference being that OMDoc focuses on representing mathematical knowledge whereas MathDox focuses on interactivity. The connections with both DocBook and OMDoc are of importance to us because we expect several authoring tools for it to emerge in the coming few years, and we want to profit from their presence.

The mathematics in the MathDox source is given by means of OpenMath objects. This feature has clear advantages in terms of portability. The DocBook type grammar sees to it that there are natural scopes, where mathematical objects 'live'. For instance, when a chapter begins with "Let $\mathbb{F}$ be a field", the scope of the variable $\mathbb{F}$ is assumed to be the whole chapter (although, somewhere further down the hierarchy, say in a section of the chapter, this assignment can be overridden).

Interactivity in MathDox is taken care of by XML tags representing various programming constructs as well as queries to external mathematical services. These actions take place within part of the context, which fixes the precise semantics of the objects involved. Further constructs are available for handling context and user input. Our notion of context is based on [FHJ+99b]. Context is divided into static and dynamic context. The static context may be defined as the set of all XML sources from which a interactive document can be prepared for use. Two extreme forms are OpenMath Content Dictionaries and a chapter of an ordinary book. The dynamic context behaves more like the state of a CAS. It keeps track of the variables introduced, their properties, their values, and their scopes. The MathDox language has constructs for storing and changing this information. For example, the field $\mathbb{F}$ introduced at the beginning of a chapter may be specified to be a finite field of five elements in the context of a particular section of the chapter.

Although semantics is the primary target, some features for presentation have been built into the language. In order to have a flexible presentation, we use presentation-annotated OpenMath. In MathDox we allow style attributes inside OpenMath objects. By discarding these style attributes, regular OpenMath is obtained. For instance, by providing the appropriate value for the style attribute, the author has a choice between a slash and a fraction display. In $\frac{3/4+2/3}{5}$ we have used them both.

Another way of solving presentation issues is illustrated by the statement: $3, 4 \in \mathbb{Z}$. The corresponding OpenMath expression would be the equivalent of $3 \in \mathbb{Z} \wedge 4 \in \mathbb{Z}$, but our OpenMath statement reads that the sequence $3, 4$ belongs to $\mathbb{Z}$. So here, the semantics of the element-of symbol has been stretched so as to help out presentation.

26.7.3 The MathDox System

An essential component of the MathDox software is its document server. It provides a view to the client of the content and manages both the static and the dynamic context. The usage of the MathDox document server is shown in Figure 26.10. We explain in some detail the main components shown in this picture.

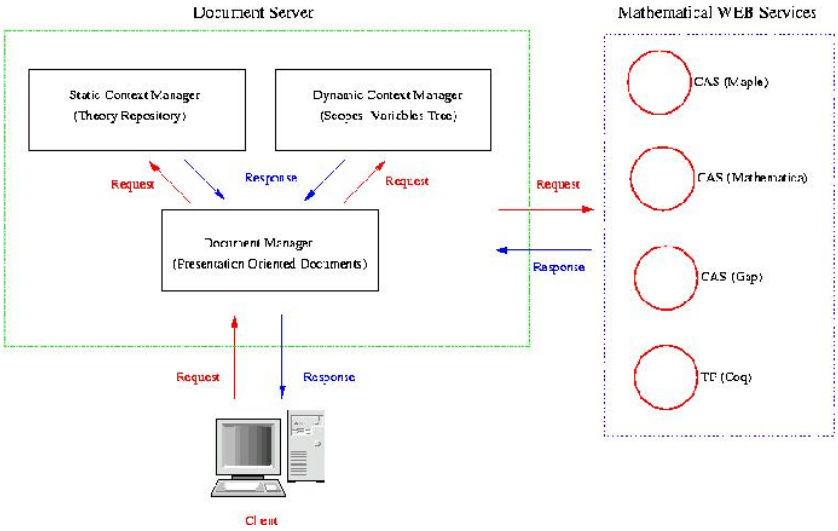


Fig. 26.10. The MathDox software

1. The *client*. The client is realized by a math-enabled web browser. It will present views of the documents served to the user, interact with the user, and communicate user input to the document server.
   The communication between client and server takes place via the HTTP request/response mechanism. The responsibility for interaction is mostly on the server side.
2. The *document server*. This server caters for presentation, communication, and context. It supports a wide range of actions ranging from handling queries to searching within documents for mathematical content and from placing (and retrieving) objects into the context, to rendering documents in different views.
   The document server is realized as a Java enhanced web application [JSP] inside a web server. It is not a monolithic entity. As shown in Figure 26.10, it is formed by the system parts. The *document manager* serves views to the client. IMDs can be thought of as programs (scripts) encoding the production of a response. In generating the response, they can make use of the

information contained in the static context, and in the dynamic context (scopes and variables), the user input communicated along with request, and the results of computations carried on by one or more mathematical services.
Another part is the *static context manager* which is responsible for managing a repository of MathDox mathematical theories.
The final (third) part is the *dynamic context manager* which is responsible for the dynamic information.

3. *mathematical services.* Mathematical services can be very diverse: some may serve as general interfaces to CAS or to Theorem Provers. The MathDox software provides ways to access these services via standard protocols, among which those developed under the MONET project [Mon]. The mechanism extends the phrasebook set-up for OpenMath [CCC+00, CCR00]. For constructing specific OpenMath services, we employ our Java OpenMath library ROML [ROM].

26.7.4 Conclusion

Now that MathDox is close to a complete working version, trial applications are in the make. We mention

- a server for providing designs of experiments on command to statisticians,
- an exercise repository for the EU funded LeActiveMath project,
- a mathematics course on calculus, with automated natural language production from a formal-mathematical source for the EU funded project WebALT,
- interactive lecture notes (the successor of [CCS99]) for an Abstract Algebra course within a mathematically oriented Bachelor curriculum,
- educational material for highschool mathematics in the Netherlands.

26.8 OMDoc in ActiveMath

Project Home	`http://www.activemath.org/`
Authors	The ActiveMath group: Erica Melis, Giorgi Goguadse, Alberto Gonzales-Palomo, Adrian Frischauf, Martin Homik, Paul Libbrecht, Carsten Ullrich DFKI GmbH and Universität des Saarlandes

ActiveMath is a mature web-based intelligent learning environment for mathematics that has been developed since 2000 at the University of Saarland and at the German Research Institute of Artificial Intelligence (Intelligent Learning Environments Group headed by Erica Melis). Its learning objects are encoded in an extension of OMDoc.

26.8.1 The ActiveMath System

In addition to presenting pre-defined interactive materials, it adaptively generates courses according to the learner's goals, learning scenarios, competencies, and preferences. For this, **Tutorial Component** requests **learning**objects [8], related to the learning goal to be retrieved from several repositories. The retrieval of object-IDs is realized by a mediator taking into account structures and meta data of learning objects, and then the Tutorial Component assembles them to a course skeleton depending on a **Learner Model**. For details see [Ull05, Ull04].

In several stages a **Presentation Component** fills and transforms this skeleton to a material in the requested output format. In the interactive browser formats dummies can represent Learning Objects that can be instantiated dynamically — depending on the learning progress or on requests by the user.

This Learner Model stores the learning history, the user's profile and preferences, and a set of beliefs that the systems holds about the cognitive and meta-cognitive competencies and the motivational state of the learner. The domain model that underlies the structure of the learner model is inferred from the content for that domain and its meta data represented in the OMDoc source.

ActiveMath is internationalized and 'speaks' German, English, French, Spanish, Russian, and Chinese by now. Its mathematical notation rendering can as well be adapted to national' standards.

To realize a smooth and efficient cooperation of all components and in order to integrate further internal and external services, ActiveMath has adopted a modular service-oriented architecture displayed in Figure 26.11. It

[8] Following the classical definitions, learning objects are any resources that are used the learning activity. When in OMDoc, learning objects considered are such as a `definition`, an `omtext` or an interactive exercise.

includes the XML-RPC web communication protocol for its simplicity and support. In addition, an event framework enables the asynchronous messaging for any changes.

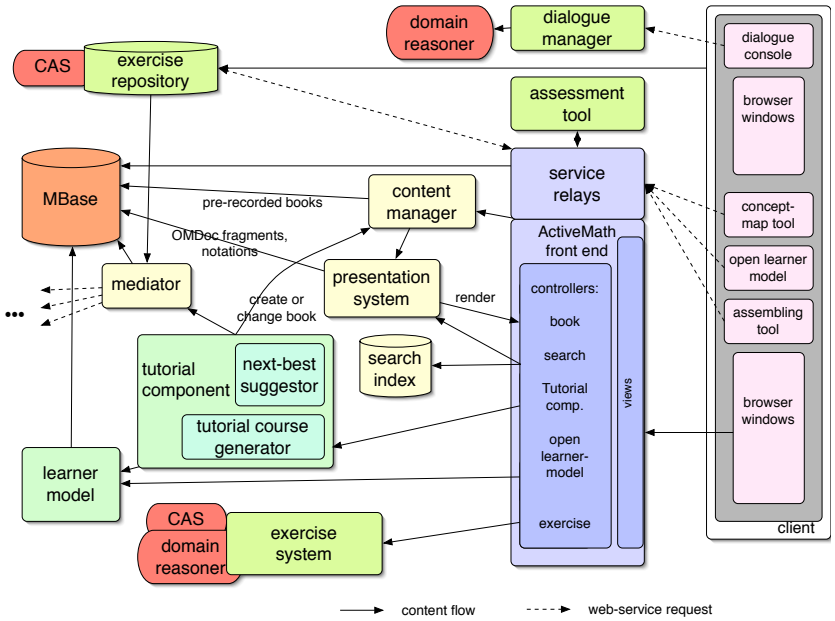


Fig. 26.11. The Components, Services and Information Flow in ActiveMath

A complex subsystem in its own right is ActiveMath's exercise subsystem [GGPM05] that plays interactive exercises, computes diagnoses and provides feedback to the learner in a highly personalized way. It reports events to inform the other components about the user's actions.

In 2005, large educational contents exist in ActiveMath's repositories for Fractions (German), Differential Calculus (German, English, Spanish) at high school and first year university level, operations research (Russian, English), Methods of Optimization (Russian), Statistics and Probability Calculus (German), Matheführerschein (German), and a Calculus course from University of Westminster in London.

ActiveMath's Service-Approach

The encoding of content in OMDoc is an advantage for ActiveMath's Web-service approach. If available, the services – including Web repositories – can communicate more semantic information than just meta data. However, the interoperability of the content encoding is only one side of the Semantic Web

coin. Hence, the developments for ActiveMath also include the reuse and interoperability of components and tools [MGH+05].

External services that are being connected currently are the Siette assessment tool [CGM+04] and one or more repository of interactive exercises and interactive content.

26.8.2 OMDoc Extensions for ActiveMath

The ActiveMath DTD extends the OMDoc DTD version 1.1 in several directions:

- new types of items such as `misconception`s, additional types of items such as types of exercises (`MCQ`, `FIB`, `map`, `problem`),
- additional several relations with types such as `for` or `prerequisite-of`,
- other additional meta data such as `difficulty`, `competency`, or `field`,
- additional infrastructure as, e.g., in exercises, additional structure such as content packages [GUM+04].

The metadata and relation extensions are compliant with the Learning Metadata Standards IEEE and IMS LOM [IEE02, Con01]. Most of the extensions are pedagogically/educationally motivated. Some details follow.

The educational metadata include `competency` and `competencylevel` that are used for assessment, evaluation, and for adaptive suggestions of examples and exercises in course generation. As for competencies, ActiveMath supports Bloom's taxonomy of learning goal levels [Blo56] and the more recent taxonomy from the Program for International Student Assessment (PISA) [KAB+04] and National Council of Teachers of Mathematics (NCTM).

ActiveMath educational metadata include `learning_context` which was in first versions of LOM. Metadata values, such as `difficulty`, `abstractness`, and "typical learning time" have been annotated with the corresponding learning context (allowing to say that an example is hard for an undergraduate but not for a higher class). The ActiveMath DTD introduced some educational relation types which facilitate adaptive course generation and concept map exercises, among others.

The OMDoc format has been refactored in ActiveMath in order to represent metadata in a form that is separable from the representation of the knowledge item. For example, some metadata represented in form of attributes of an item is moved inside the metadata element. The purpose of such a separation is to facilitate the management of learning materials in ActiveMath. Components such as Tutorial Component and Learner Model do not deal with the content of the knowledge items but rather with their metadata only and hence it is convenient to have a way to extract metadata records from the content.

For the internationalization each OMDoc item may have sub-elements in several languages since ActiveMath does not translate learning objects on the fly.

ActiveMath extends the OMDoc `example` element. A detailed explanation can be found in [MG04]. In case of a worked-out example, the micro-structure of this element is enriched with a solution that has a structure similar to a proof in OMDoc. It differs from the proof element since the solution might not only prove a statement, but also calculate the value of some expression or explore the properties of a particular structure (e.g. curve discussion). This representation allows for different presentations, and serves as a basis for the automatic generation of exercises by fading some parts of the structure of a worked-out example (see [MG04]).

The new exercise representation of ActiveMath was the basis for extending the Math QTI standard [MGP04]. Even though its origin can be traced to OMDoc originally not much is left from the QUIZ representation of OMDoc which supports only very limited types of exercises and did not have enough infrastructure. The micro-structure of an interactive exercise has to allow for different kinds of interactivity, checking the correctness of the answer, providing feedback, etc. This interaction graph can be automatically filled with information by the exercise subsystem components that can communicate with external systems in order to generate feedback to the user.

A description of ActiveMath language for exercises can be found in [GGPM05].

26.8.3 Usage of Semantic Representation in ActiveMath

The fact that the Tutorial Component employs metadata to search for appropriate learning objects and assemble them has been sketched above. In addition, other tools and components of ActiveMath make use of the semantics of OpenMath, the ActiveMath metadata and OMDoc more generally.

Computer Algebra Services

Computer algebra system (CAS) — currently Yacas [Yac], Maxima [Max], and Wiris [Wir] — are integrated as external services. Via a broker, a CAS receives queries (partially Monet queries) to evaluate OpenMath expressions. This enables the exercise system to evaluate user input, e.g., for numerical or semantic equivalence with a particular expression. The service CAS has to translate in- and output via phrasebooks.

Presentation Component

The naive approach to rendering OMDoc documents would be to fetch the items from a data base, assemble them (or parts of them) and then run several style-sheets on the resulting sequence; Those style-sheets would depend on the requested output format (HTML+ Unicode, XHTML+ MathML, PDF via LaTeX, svg, or slides), the target browser (we support Mozilla, FireFox, Internet Explorer) and the personalization.

This approach turned out to be infeasible for complex, real-world applications. Therefore ActiveMath includes a multi-stage presentation process as described in [ULWM04]. It has many advantages, among them a much better performance and even better perceived performance through multiple caching, a clear separation of different concerns which provides more flexibility for the various adaptivity dimensions that ActiveMath supports, including selection of learning objects, link annotations language, specific presentations of pages, exercises etc, and of mathematical expressions, target output format, browser.

The final rendering maintains the references to mathematical symbols but renders them invisible. This information can then be used by copy-and-paste and for tool tips that indicate the name of a symbol on the page.

For an even more specialized presentation of mathematical notation which is often requested by authors and users we developed a complex presentation tag representation and an authoring facility for it [**?**]. These special presentations are integrated into the presentation process upon request.

Copy and Paste

The rendering includes an invisible reference to the unique identifier of mathematical symbols and expressions. This provides a basis for copying the reference to an OpenMath expression, i.e., the semantics of the expression to a computer algebra system, to the input editor (in dictionary and exercises), and into exercise blanks. The actual transfer mechanism is, because of security limitations and because of resource management, a drag-and-drop operation which allows immediate negotiation between the recipient and source. This allows to transform to the appropriate encoding on demand. Alternate encodings include OpenMath with a restricted set of content-dictionaries, HTML with embedded presentation and content MathML. Reference to OMDoc items and to pages of a book in ActiveMath are exchangeable using the same paradigm.

Interactive Concept Map Tool icmap

icmap utilizes the OMDoc encoding and relations for generating feedback to users' inputs [MKH05]. The tool visualizes (parts of) a domain and relations between concepts and between concepts and satellites.

Semantic Search

ActiveMath's search facility has been upgraded to enable not only approximate search results but also to search semantically for (OpenMath) mathematical expressions, for certain types of learning objects and objects with particular metadata. The implementation of the search uses Jakarta Lucene with its high-performance and easy deployment.

OMDoc-Related Components and Tools of ActiveMath

Many of the tools described above have not been sufficient for the purposes of a complex and mature educational application such as ActiveMath. Therefore, we had to improve them or implement some from scratch. In particular, these include authoring tools (for which improvement is still ongoing), transformation tools, validation tools, and style sheets. Moreover, new tools have been developed or integrated into ActiveMath, e.g., an input editor that returns OpenMath.

The conversion of OMDoc source to presentation code is done using XSLT style sheets. We started with the style sheets available in OMDoc repository and added to them the ActiveMath linking schemata. These style sheets needed more polishing since they were too big and the management of notations was not feasible. Moreover, the TEX oriented style sheets had to be refurbished in order to work well with big documents.

Further tools have been realized within the authoring tools which are covered in Section 26.9.

26.9 Authoring Tools for ActiveMath

Project Home	http://www.activemath.org/projects/jEditOQMath
Authors	Paul Libbrecht DFKI GmbH and Universität des Saarlandes

The OMDoc content to be delivered by ActiveMath are OMDoc documents with OpenMath formulae. Experience has shown that writing the XML-source by hand is feasible and even preferred if the author wants to follow the evolution of content's structure. It is similar to HTML editing. However, the complexity of XML makes it hard to keep an overview when writing mathematical expressions. Therefore, the OQMath processor has been implemented: it uses QMath for formulae and leaves the rest of the OMDoc written as usual XML.

OQMath has been integrated in a supporting XML-editor, jEdit. This editor provides structural support at writing XML-documents. Authors, even with no XML-knowledge, can easily write valid document jEditOQMath. This package includes, in a one-click installer, QMath, OQMath, jEdit, and Ant-scripts for publication of the content in ActiveMath knowledge bases. These scripts validate the references in the content. These scripts also provide authors with short cycles edit-in-jEditOQMath-and-test-in-ActiveMath. More about jEditOQMath can be seen from `http://www.activemath.org/projects/jEditOQMath` at [Lib04]

jEditOQMath provides search facilities as well as contextual drops from items presented in an ActiveMath window. This way the testing of content in the target environment and the authoring experience are bound tighter together, thus making jEditOQMath closer to the WYSIWYG paradigm without being limited to its simple visual incarnation.

To date, more than $10'000$ *items* of OMDoc content has been written using these authoring tools in Algebra and Calculus. This experience with authors considerably improved our understanding of what today's authors need and what different classes of authors can cope with.

Among the greatest difficulties of authoring content for ActiveMath was the art of properly choosing mathematical semantic encoding: the mathematical discourse is made of very fine notation rules along with subtle digressions to these rules... formalizing them, as is needed when writing OpenMath or the QMath formulae for them, turns out to often be overwhelming. The usage of the ellipsis in such a formula as $1, \ldots, k, \ldots, n$ is a simple example of semantic encoding challenge. The knowledge organization of OMDoc that makes it possible to define one's own OpenMath symbols has been a key ingredient to facing this challenge.

Among the features most requested by authors, which we have tried to answer as much as possible, are a short edit-and-test cycle and validation facilities taking in account the overall content.

Validation Tools

Automated validation of OMDoc content has many facets. XML-validation with a DTD and Schema is a first step. However there are still many structure rules mentioned only as human readable forms in the OMDoc specifications. References between OMDoc items is another important facet which has been answered by ActiveMath knowledge bases and publishing scripts. Experience has proved that ignoring such errors has lead repeatedly to authors complaining about the weirdest behaviours of the overall learning environment. Many other simple validations could be done in order to support the author, for example the validation of a picture embedding, or of fine grained typing of relations (for example, that a definition should only be *for* a symbol).

Further validation tools are being investigated, for example, those tuned to particular pedagogical scenarios.

Further Authoring Tools for ActiveMath

jEditOQMath clearly remains for users who feel comfortable with source editing. Experience has shown that authors having written HTML or TEX earlier did not find this paradigm problematic. It is, however, a steep learning slope for beginner authors. A more visual component is being worked upon, able to display and edit visually the children of a `CMP`, including formulae.[9] This component, along with forms and summaries for metadata, should provide a visual environment to edit OMDoc content for ActiveMath in a relatively accessible way.

Another area where source editing has shown difficulties is in the process of authoring exercises with many steps... the rich structure of the exercises, along with the non-neglect able space taken by the display of XML-source has challenged several authors, having difficulties to overview such sources as 600 Kb of OQMath source for a single exercise. A web-based visual authoring environment is under work within the ActiveMath group.

[9] More about the component for OMDoc micro-structure can be read from `http://www.activemath.org/projects/OmdocJdomAuthoring/`.

26.10 SWiM – An OMDoc-based Semantic Wiki

Project Home	`http://kwarc.eecs.iu-bremen.de/projects/swim`
Authors	Christoph Lange, Michael Kohlhase Computer Science, International University Bremen

SWiM is a semantic wiki for collaboratively building, editing and browsing a mathematical knowledge base of OMDoc theories. Our long-term objective is to develop a software that facilitates the creation of a shared, public collection of mathematical knowledge and serves work groups of mathematicians as a tool for collaborative development of new theories. Even though the work reported here was initially motivated by solving the MKM author's dilemma [KK04], we contend that the new application area MKM can also contribute to the development of semantic wikis.

Technically, SWiM is based on the semantic wiki engine IkeWiki [Sch06], which was chosen because of its modular design, its rich semantic web infrastructure, its user assistance for annotations, and its orientation towards learning [SBB+06].

26.10.1 Semantic Wikis

A wiki [LC01] is a web server application that allows users to browse, create, and edit hyperlinked pages in a web browser, usually using a simple text syntax. In contrast to most content management systems, wiki pages are accessible via an URL containing their title. A new page can be created by linking from an existent page to the page to be created. This link will then lead to an edit form. Usually, anyone is allowed to edit pages on a wiki, but access can be restricted. Other characteristics of wikis include permanent storage of old page versions (with facilities to display differences between two versions and to restore a certain version), notification about recent changes, and full-text search.

Semantic wikis [VKS+07, TS06] enhance wikis by Semantic Web technologies, such as RDF [LS99] or ontologies. Usually one page represents one concept from a real-world domain, which has a type, possibly some metadata, and typed links to other concepts. For example, a link from a wiki page about "Life, the Universe and Everything" to another page about Douglas Adams could be typed as "is author of". In terms of RDF, this can be expressed by the following subject–predicate–object triple,

("Douglas Adams", isAuthorOf, "Life, the Universe and Everything")

where the *isAuthorOf* relation would be defined in an ontology. These links are usually displayed in a navigation box next to the page contents. Semantic wikis only deal with wiki text, not with mathematics, though some allow to embed mathematical formulae as presentational-only TeX.

SWiM encourages users to collaborate: Non-mathematicians can collaborate in creating a "Wikipedia of mathematics" by compiling the knowledge available so far, while scientists can collaboratively develop new theories. Users get an immediate reward for many of their contributions: Once they specify the type of a page or relations of one page to another, this information will be displayed in a box of navigation links. We intend to make the data created in SWiM usable for external services by offering an export facility for OMDoc documents and by integrating them into SWiM. Mathematicians developing theories will be assisted to retain an overview of theory dependencies in order not to break them. Social software services will further utilize the semantic information available from the theories and from tracking the user interaction log ("Who did what on which page when?"). User feedback to pages can be extended to social bookmarking, which is "the practice of saving bookmarks [of Internet resources] to a public web site and 'tagging' them with keywords." [Lom05] The more users tag a certain resource, the higher a social bookmarking service will rank it.

The enhancements of the data model semantic wikis bring along — compared to traditional wikis — are already present in the OMDoc format, so that an OMDoc-based wiki only needs to operationalize their underlying meaning. For example, typed links, which are implemented via an extension to the wiki syntax in Semantic MediaWiki [VKV+06] or editable through a separate editor in IkeWiki [Sch06], are implemented by means of the `for` attribute to OMDoc's elements (e.g. `<example for="#id-of-assertion">`). SWiM makes them editable easily and visualizes them adequately. A semantic wiki targeted at mathematics must ensure that dependencies between concepts are preserved. Results in this area will be interesting for non-mathematical semantic wikis as well, especially when they support higher levels of formalization such as ontologies.

26.10.2 Design of SWiM

Concepts and Relations

The smallest unit that can be displayed, edited, linked to, or archived in a wiki is a page. In a semantic wiki, it usually describes one *concept*, including its properties and its relations to other concepts. While standalone OMDoc documents can contain more than one theory, is is important to keep pages small in a wiki to improve the effectivity of usage. Furthermore, usual semantic wikis only store and display metadata and typed links per page; SWiM does too.[10] Users are strongly encouraged to define at most one theory per wiki page and to roll out non-constitutive statements (see Section 15.1) to separate pages, referencing their context theory. As constitutive

[10] Semantic information will only be considered on the theory and statement levels of OMDoc — directly or through reasoning in the case of transitive closures —, not on the object level.

statements cannot exist without an enclosing theory, but as, on the other hand, we want each wiki page to form a valid document, we introduced a new element `swim:page`, which can be a child of an `omdoc` element and which has the same content model as a `theory` element — in particular, it can hold several theory-constitutive statements and connect them to their context theory.

OMDoc's system ontology has been partly coded in OWL-DL and imported to the wiki's RDF store, which is implemented using the Jena Semantic Web Framework for Java [JEN08]. Theories as well as statements of any type form concepts, and the most important relations between those concepts are extracted from the OMDoc pages on saving and then stored as RDF triples. These relations include:

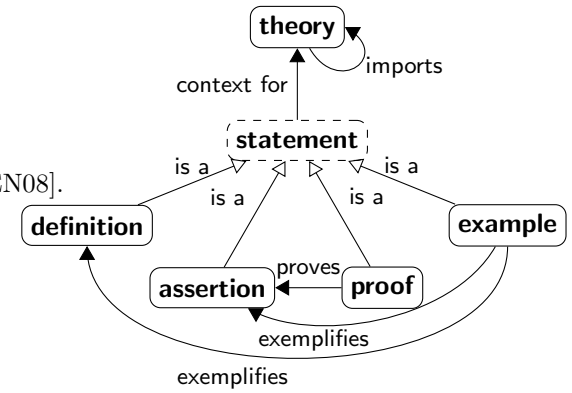


Fig. 26.12. Subset of OMDoc's system ontology

- The import relation between theories
- The relation of a statement to its context theory
- The relation of an example to the statement it exemplifies
- The relation of a proof to the assertion it proves

It is planned to also take relations given by user interaction into consideration, such as "Who edited which page when?", and to combine ontology-defined relations and user relations. For example, a metric estimating the *degree of difficulty* of a page, calculated by counting the questions on the discussion page, could be implemented. Furthermore, the user can specify taxonomic relations, which cannot be stated explicitly in OMDoc, such as ("all differentiable functions are continuous"), as annotations in an ontology language like RDF Schema or Owl.

User Interface and Interaction Model

Pages can be rendered to XHTML plus presentational MathML using the transformations described in Chapter 25. There is also a browsable source code view, which is useful for documents that are not written in textbook style.

Not only will the user be able to navigate along the dependency graph, she will also be able to *interact* with the system: she will be asked whether she wants to explore the theories required as dependencies in further detail.

Suppose that the user is currently reading the page containing the theory `ring` from the elementary algebra example from Chapter 7. In this case the wiki will not only display navigation links to the direct dependencies `group` and `monoid`, but it will also provide unobtrusive buttons that allow the user to give one of the commands in Figure 26.13. Not only the last case will be recorded — the others are interesting as well for *social bookmarking*. For example, if many users requested a theory t to be explained, the system could default to display not only the direct dependencies but also the level-two dependencies, for it seems that t is too difficult for only being explained shallowly.

No, thanks! "*I already know group and monoid.*"
Explain "*Please show me group and monoid, I want to learn about ring's prerequisites.*" — group and monoid will be displayed.
Explore "*Please show me* all *prerequisites for ring.*" — group, monoid, and semigroup, are opened in separate windows or serialized into one page.
Suspend "*I want to know about group and monoid, but only later.*" — SWiM keeps a notice in the user's profile that she wants to read group and monoid sometime. Reminder links to suspended theories are shown on a separate navigation bar.

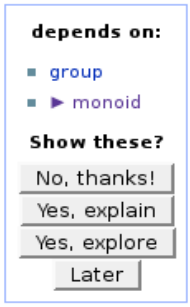


Fig. 26.13. The command buttons to navigate along the dependencies

Further work

Further work on SWiM will concentrate on integrating a lightweight management of change process. Second, while the wiki is yet a user-friendly *browser*, there is still a demand for assisting users to *edit* OMDoc. To this end, the QMath preprocessor (see Section 26.2) will be integrated into SWiM. Mathematical objects entered as QMath will be kept in this syntax for display in the edit form, but they will be converted to OMDoc for rendering for presentation and when pages are exported to another application.

26.11 Induction Challenge OMDoc Manager (ICOM)

Project Home	`http://www.cs.nott.ac.uk/~lad/research/challenges/challenge_manager.html`
Authors	Thomas D. Attfield, Monica C. Duarte, Lin Li, Ho-Ying Mak, Adam M. Neal, Lewis M. Toft, Zixuan Wang, Louise A. Dennis School of Computer Science and Information Technology, University of Nottingham

We describe work in progress to create a system for organising and presenting a set of challenge problems collected by the Induction Theorem Proving community. These challenge problems come from a number of sources and are presented in different logics using different presentation conventions.

The intention is to provide a system which will allow these problems to be stored in a unified format and will support the collection, browsing and extraction of the problems.

OMDoc is an obvious choice for representing such problems and the system is able to take advantage of much existing work on the manipulation of XML documents.

26.11.1 The Induction Challenge Problems

Inductive Theorem proving is a small field. The main theorem provers within this field are NqThm [BM79] (now re-engineered as ACL2 [KM96]), InKa [AHMS99], the Clam series [BvHHS90, RSG98a] and rrl [KZ95]. Twelf [PS99] also looks at the automation of inductive proof in the context of logical frameworks. Within the field it is hard to assess claims for the superiority of any given system since there is naturally a tendency to report "successes" – difficult or challenging problems automatically proved. There is also a desire within the community to develop a store of shared knowledge about the challenges that face the automation of proof by mathematical induction.

TPTP (Thousands of Problems for Theorem Proverss) [SS98] is a library of test problems for first-order ATP systems. They provide the ATP community with a comprehensive library complete with unambiguous names and references. All the problems are stated in a standardised formulation of first-order logic and are widely used to benchmark first-order systems. They are also used as the test set for the CASC competition [Sut01] which compares such systems. One of the benefits of the TPTP library to the ATP community is the existence of a common set of problems by which comparisons can be made.

It is not practical for inductive theorem provers to follow the pattern of the TPTP library. Various attempts have been made to build a similar corpus of problems requiring inductive reasoning. The most mature of these

was based on the Boyer-Moore [BM79] corpus[11]. This corpus was unpopular partly because there was repetition within the problem set and partly because many problems depended on a few particular function definitions. But the major objection was that induction theorem provers use a number of different logics, some of which are typed and some of which are not, which made it difficult to agree on a standard format. The use of other logics also raised translation issues and a fully automated process for converting the theorems, even into an agreed typed language was never produced.

A group of researchers within the community[12] agreed that instead of a large set of benchmarks in a standard logic they would each put forward a number of "Challenge Problems". These should present interesting challenges to the automation of inductive proof or illustrate important features which an inductive prover should be able to handle. A set of these problems would be collected which would remain sufficiently small that an individual could represent them within their own theorem proving system as they saw fit[13]. These challenge problems are currently described in a high-level way and written up in an ad hoc fashion. The descriptions contain both mathematical notation and commentary. They are difficult to read, navigate or use in any particular system.

OMDoc seems ideally suited as a format for representing these challenge problems: it can represent both text and formulae; it is not tied to any particular logic and it supports the extraction of data into a number of different formats. As an added benefit its hyper-text features would potentially allow definitions to be stored separately and shared between problems. Individual theorem provers can then concentrate on translations between OpenMath content dictionariesnd their own logics and individuals submitting problems can specify the appropriate content dictionary for the problem.

26.11.2 System Description

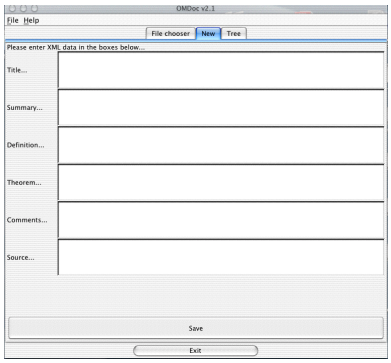
The Induction Challenge OMDoc Manager (ICOM) is designed to be a system which will ease the submission and extraction process for the problems. Our intention is to provide a submission interface that will create a simple OMDoc markup for the problems which can subsequently be edited by a user and to provide browsing and extraction capabilities.

Each challenge problem description contains six distinct sections (e.g. Summary, Definitions, Comments). Currently a user who wishes to enter a problem into our system is presented with the form shown on the right with a field for each section.

[11] This has become known as the Dmac corpus after David McAllester who translated a fragment of the NqThm corpus into a simpler language.

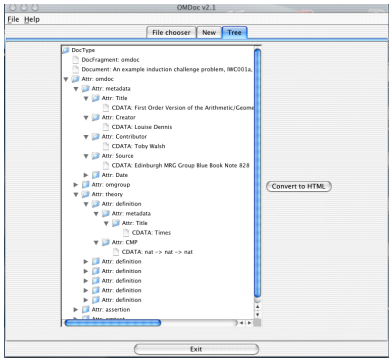
[12] At the 2000 CADE Workshop on the Automation of Proof by Mathematical Induction.

[13] The current set can be found at `http://www.cs.nott.ac.uk/~lad/research/challenges`.



Each section, once entered by a user, is placed in a `CMP` tag. These tagged fragments are wrapped in standard OMDoc headers and footers to produce a valid OMDoc. This completed document is then written to disk and stored. We are currently working on a simple parser to translate equations into `om:OMOBJ` structures which a user will then be able to edit (for instance to specify the appropriate content dictionaries). We hope this will be easier than adding all the OpenMath tags by hand.

An existing document can be displayed as a tree and from this tree the document can be directly manipulated. This tree display also allows the user to see the structure of the document more clearly. It is also possible to extract an HTML view of the contents of the document so it can be displayed in a web browser and read by a human.



Our implementation language is JAVA and we use its JAXP DOM API. DOM [DOM] is a W3C standard which uses a tree-based model (storing data in hierarchies of nodes). This means that once an OMDoc has been created or opened all the document's data is in memory and so data can be accessed rapidly. DOM also enables simple modification of documents by adding or deleting nodes. Although Sax (an alternative model) achieves better performance and less memory overhead than DOM, it is easier to traverse and modify XML documents using a DOM tree structure. Since we anticipate that users may wish to modify the initial OMDoc produced by our system we adopted the DOM model instead.

26.11.3 Further Work

ICOM is still in the early stages of development. Currently our most pressing aim is to provide improved support for entering equations. Once this is in place we hope to add searching facilities and provide better mechanisms for links to be created between challenge problems. We would also like to experiment with the automatic extraction of problems into a theorem prover via an MBase [KF00] and a MathWeb [FHJ+99a].

26.12 Maya: Maintaining Structured Developments

Project Home	`www.dfki.de/~inka/maya.html`
Authors	Serge Autexier[1], Dieter Hutter[1], Till Mossakowski[2], Axel Schairer[1] [1] DKFI GmbH, Stuhlsatzenhausweg 3, D 66123 Saarbrücken [2] Computer Science, University of Bremen, Germany

26.12.1 Overview

The MAYA-system was originally designed to maintain and utilize the structuring mechanisms incorporated in various specification languages when evolving and verifying formal software developments. In this setting, a software system as well as their requirement specifications are formalised in a textual manner in some specification language like CASL [CoF04] or VSE-SL [AHL$^+$00]. All these specification languages provide constructs similar to those of OMDOC to structure the textual specifications and thus ease the reuse of components. Exploiting this structure, e.g. by identifying shared components in the system specification and the requirement specification, can result in a drastic reduction of the proof obligations, and hence of the development time which again reduces the overall project costs.

However, the logical formalisation of software systems is error-prone. Since even the verification of small-sized industrial developments requires several person months, specification errors revealed in late verification phases pose an incalculable risk for the overall project costs. An *evolutionary formal development* approach is absolutely indispensable. In all applications so far development steps turned out to be flawed and errors had to be corrected. The search for formally correct software and the corresponding proofs is more like a *formal reflection* of partial developments rather than just a way to assure and prove more or less evident facts.

The MAYA-system supports an evolutionary formal development since it allows users to specify and verify developments in a structured manner, incorporates a uniform mechanism for verification *in-the-large* to exploit the structure of the specification, and maintains the verification work already done when changing the specification. MAYA relies on *development graphs* [AH05, Hut00] as a uniform (and institution independent[14]) representation of structured specifications, and which provide the logical basis for the *Complex theories* and *Development graphs* of OMDOC[15]. Relying on development graphs enables MAYA to support the use of various (structured) specification languages like OMDOC, CASL [CoF04], and VSE-SL [AHL$^+$00]

[14] This includes, for instance, that it does not require a particular logic (see e.g. [MAH06] for more details).
[15] These are the modules CTH and DG, respectively.

to formalise mathematical theories or formal software developments. To this end MAYA provides a generic interface to plug in additional parsers for the support of other specification languages. Moreover, MAYA allows the integration of different theorem provers to deal with the actual proof obligations arising from the specification, i.e. to perform verification *in-the-small*.

Textual specifications are translated into a structured logical representation called a development graph, which is based on the notions of consequence relations and morphisms and makes arising proof obligations explicit. The user can tackle these proof obligations with the help of theorem provers connected to MAYA like Isabelle [Pau94] or INKA [AHMS99].

A failure to prove one of these obligations usually gives rise to modifications of the underlying specification. MAYA supports this evolutionary process as it calculates minimal changes to the logical representation readjusting it to a modified specification while preserving as much verification work as possible. If necessary it also adjusts the database of the interconnected theorem prover. Furthermore, MAYA communicates explicit information how the axiomatization has changed and also makes available proofs of the same problem (invalidated by the changes) to allow for a reuse of proofs inside the theorem provers. In turn, information about a proof provided by the theorem provers is used to optimise the maintenance of the proof during the evolutionary development process.

26.12.2 From Textual to Logical Representation

The specification of a formal development in MAYA is always done in a textual way using specification languages like CASL , OMDOC or VSE-SL. MAYA incorporates parsers to translate such specifications into the MAYA-internal specification language DGRL ("Development Graph Representation Language"). DGRL provides a simply-typed λ-calculus to specify the local axiomatization of a theory in a higher-order logic. While unstructured specifications are solely represented as a signature together with a set of logical formulas, the structuring operations of the specification languages are translated into the structure of a development graph. Each node of this graph corresponds to a theory. The axiomatization of this theory is split into a local part which is attached to the node as a set of higher-order formulas and into global parts, denoted by ingoing definition links, which import the axiomatization of other nodes via some consequence morphisms (such as the `imports` element in OMDOC). While a *local* link imports only the local part of the axiomatization of the source node of a link, *global* links are used to import the entire axiomatization of a source node (including all the imported axiomatization of other nodes). In the same way local and global *theorem link*s are used to postulate relations between nodes (see [AH05] for details) which correspond to OMDOC's `theory-inclusion` and `axiom-inclusion` elements.

On the left hand side, Figure 26.14 shows the graphical user interface of MAYA. The right hand side shows the development graph in MAYA for a

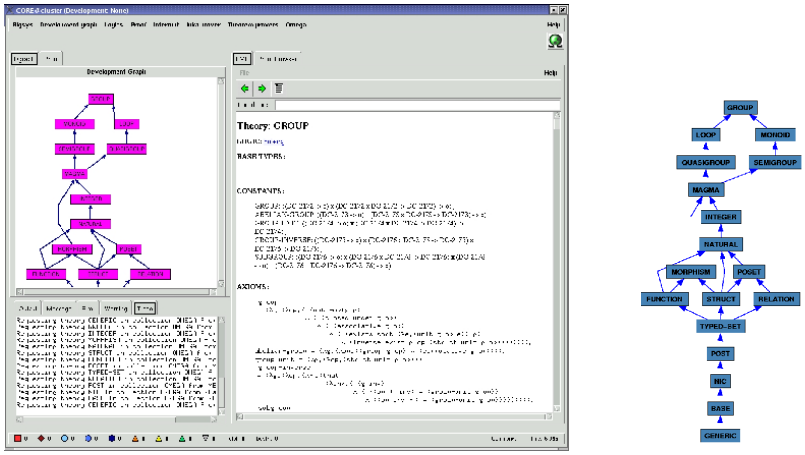


Fig. 26.14. The graphical user interface of MAYA & the development graph for the OMDOC Representation of Groups in MBASE

formalisation of groups. The formalisation was given in OMDOC and imported into MAYA from the OMDOC database MBASE (see [KF01] and Section 26.4).

26.12.3 Verification In-the-large

The development graph is the central data-structure to store and maintain the formal (structured) specification, the arising proof obligations and the status of the corresponding verification effort (proofs) during a formal development.

MAYA distinguishes between proof obligations postulating properties between different theories (like the `theory-inclusion` and `axiom-inclusion` elements in OMDOC) and lemmata postulated within a single theory (like `assertion` in OMDOC). As theories correspond to subgraphs within the development graph, a relation between different theories, represented by a global theorem link, corresponds to a relation between two subgraphs. Each change of these subgraphs can affect this relation and would invalidate previous proofs of this relation. Therefore, MAYA decomposes relations between different theories into individual relations between the local axiomatization of a node and a theory (denoted by a local theorem link). Each of these relations decomposes again into a set of proof obligations postulating that each local axiom of the node is a theorem in the target theory with respect to the morphism attached to the link.

While definition links establish relations between theories, theorem links denote lemmata postulated about such relations. Thus, the reachability between two nodes establishes a formal relation between the connected nodes (i.e. the theory of the source node is part of the theory of the target node wrt. the morphisms attached to the connecting links). MAYA uses this property

to prove relations between theories by searching for paths between the corresponding nodes (instead of decomposing the corresponding proof obligation in the first place).

26.12.4 Verification In-the-small

When verifying a local theorem link or proving speculated lemmata, the conjectures have to be tackled by some interconnected theorem prover. In both cases the proofs are done *within* a specific theory. Thus, conceptually each theory may include its own theorem prover. In principle, there is a large variety of integration types. The tightest integration consists of having a theorem prover for each node wrt. which theory conjectures must be proven, and the theorem prover returns a proof object generated during the proof of a conjecture. Those are stored together with the conjecture and can be used by Maya to establish the validity of the conjecture if the specification is changed. The loosest integration consists in having a single generic theorem prover, which is requested to prove a conjecture within some theory and is provided with the axiomatization of this theory. The theorem prover only returns whether it could prove a conjecture or not, without any information about axioms used during the proof. For a detailed discussion of the advantages and drawbacks of the different integration scenarios see [AM02].

Currently, Maya supports two integration types: One where information about used axioms is provided by the theorem prover, and one where no such information is provided. In the first case, Maya stores the proof information and the axioms used during the proof. In the second case, Maya assumes there is a proof for the proof obligation, as there is no information about the proof. In both scenarios, Maya makes use of generic theorem provers which are provided with the axiomatization of the current theory. Currently Maya provides all axioms and lemmata located at theories that are imported from the actual theory by definition links to the prover. Switching between different proof obligations may cause a change of the current underlying theory and thus a change of the underlying axiomatization. Maya provides a generic interface to plug in theorem provers (based on an XML-RPC protocol) that allows for an incremental update of the database of the prover.

26.12.5 Evolution of Developments

The user executes changes to specifications in their textual representation. Parsing a modified specification results in a modified DGRL-specification. In order to support a management of change, Maya computes the differences of both DGRL-specifications and compiles them into a sequence of basic operations in order to transform the development graph corresponding to the original DGRL-specification to a new one corresponding to the modified DGRL-specification. Examples of such basic operations are the insertion or deletion of a node or a link, the change of the annotated morphism of a link, or the

change of the local axiomatization of a node. As there is currently no optimal solution to the problem of computing differences between two specifications, MAYA uses heuristics based on names and types of individual objects to guide the process of mapping corresponding parts of old and new specification. Since the differences of two specifications are computed on the basis of the internal DGRL-representation, new specification languages can easily be incorporated into MAYA by providing a parser for this language and a translator into DGRL.

The development graph is always synthesised or manipulated with the help of the previously mentioned basic operations (insertion/deletion/change of nodes/links/axiomatization) and MAYA incorporates sophisticated techniques to analyse how these operations will affect proof obligations or proofs stored within the development graph. They incorporate a notion of monotonicity of theories and morphisms, and take into account the sequence in which objects are inserted into the development graph. Furthermore, the information about the decomposition and subsumption of global theorem links obtained during the verification *in-the-large* is explicitly maintained and exploited to adjust them once the development graph is altered. Finally, the knowledge about proofs, e.g. the used axioms, provided by the interconnected theorem provers during the verification *in-the-small* is used to preserve or invalidate the proofs.

26.12.6 Conclusion and System Availability

The MAYA-system is mostly implemented in Common Lisp while parts of the GUI, shared with the ΩMEGA-system [SHB+99], are written in Mozart. The CASL-parser is provided by the CoFI-group in Bremen (see Section 26.13). The MAYA-system is available from the MAYA-web-page at `http://www.dfki.de/~inka/maya.html`.

The Heterogeneous Tool Set (HETS, see Section 26.13) extends MAYA with a treatment of hiding [MAH06], and a uniform treatment of different logics based on the notion of heterogeneous development graphs [Mos02]. Furthermore, it is planned to extend this with the maintenance of theory-specific control information for theorem provers. The latter comprises a management for building up the database of theorem provers by demand rather than providing all available axioms and lemmata at once and it comprise the management of meta-level information, like tactics or proof plans, inside MAYA.

26.13 Hets: The Heterogeneous Tool Set

Project Home	`www.tzi.de/cofi/hets`
Authors	Till Mossakowski, Christian Maeder, Klaus Lüttich Computer Science, University of Bremen, Germany

26.13.1 Motivation

> *"There is a population explosion among the logical systems used in computer science. Examples include first order logic, equational logic, Horn clause logic, higher order logic, infinitary logic, dynamic logic, intuitionistic logic, order-sorted logic, and temporal logic; moreover, there is a tendency for each theorem prover to have its own idiosyncratic logical system. We introduce the concept of* institution *to formalize the informal notion of 'logical system'." [GB92]*

In the area of formal specification and logics used in computer science, numerous logics are in use:

- logics for specification of data types,
- process calculi and logics for the description of concurrent and reactive behaviour,
- logics for specifying security requirements and policies,
- logics for reasoning about space and time,
- description logics for knowledge bases in artificial intelligence and for the Semantic Web,
- logics capturing the control of name spaces and administrative domains (e.g. the ambient calculus), etc.

Indeed, at present, it is not imaginable that a combination of all these (and other) logics would be feasible or even desirable — even if it existed, the combined formalism would lack manageability, if not become inconsistent. Often, even if a combined logic exists, for efficiency reasons, it is desirable to single out sublogics and study translations between these (cf. e.g. [Sch04]). Moreover, the occasional use of a more complex formalism should not destroy the benefits of *mainly* using a simpler formalism.

This means that for the specification of large systems, heterogeneous multi-logic specifications are needed, since complex problems have different aspects that are best specified in different logics. Moreover, heterogeneous specifications additionally have the benefit that different approaches being developed at different sites can be related, i.e. there is a formal interoperability among languages and tools. In many cases, specialized languages and tools often have their strengths in particular aspects. Using heterogeneous specification, these strengths can be combined with comparably small effort.

OMDoc deliberately refrains from a full formalization of mathematical knowledge: it gains its flexibility through avoiding the specification of a formal semantics of the logic(s) involved. By contrast, the Heterogeneous Tool

Set (HETS, [MMLW]) is based on a rigorous formal semantics. HETS gains its flexibility by providing *formal interoperability*, i.e. integration of different formalisms on a clear semantic basis. Hence, HETS is a both flexible, multilateral *and* formal (i.e. based on a mathematical semantics) integration tool. Unlike other tools, it treats logic translations (e.g. codings between logics) as first-class citizens.

26.13.2 Institutions, Entailment Systems and Logics

Heterogeneous specification is based on individual (homogeneous) logics and logic translations [Mos05]. To be definite, the terms 'logic' and 'logic translation' need to be formalized in a precise mathematical sense. We here use the notions of *institution* [GB92] and *entailment system* [Mes89], and of *comorphism* [GR02] between these.

Logical theories are usually formulated over some (user-defined) vocabulary, hence it is assumed that an institution provides a notion of *signature*. Especially for modular specification, it is important to be able to relate signatures, which is done by *signature morphisms*. These can be composed, and hence form a *category of signatures and signature morphisms*.

Furthermore, an institution provides notions of *sentences* and *models* (over a given signature Σ). Models and sentences are related by a *satisfaction relation*, which determines when a given sentence holds in a model. An entailment system also provides an *entailment (provability) relation*, which allows to infer sentences (conclusions) from given sets of sentences (premises, or axioms).

Finally, it is assumed that each signature morphism leads to translations of sentences and models that preserve satisfaction and entailment.

A *institution comorphism* is a translation between two institutions. It maps signatures to signatures, sentences to sentences and models to models, such that satisfaction is preserved (where models are mapped contravariantly, i.e. against the direction of the comorphism).

We refer the reader to the literature [GB92, Mes89, MGDT05] for formal details of institutions and comorphisms. Subsequently, we use the terms "institution" and "logic" interchangeably, as well as the terms "institution comorphism" and "logic translation".

26.13.3 The Architecture of the Hets System

HETS is a tool for parsing, static analysis and proof management combining various such tools for individual specification languages, thus providing a tool for heterogeneous multi-logic specification (see Fig. 26.15). The graph of currently supported logics and logic translations is shown in Fig. 26.16. However, syntax and semantics of heterogeneous specifications as well as their implementation in HETS is parametrized over an arbitrary such logic graph. Indeed, the HETS modules implementing the logic graph can be compiled independently of the HETS modules implementing heterogeneous specification,

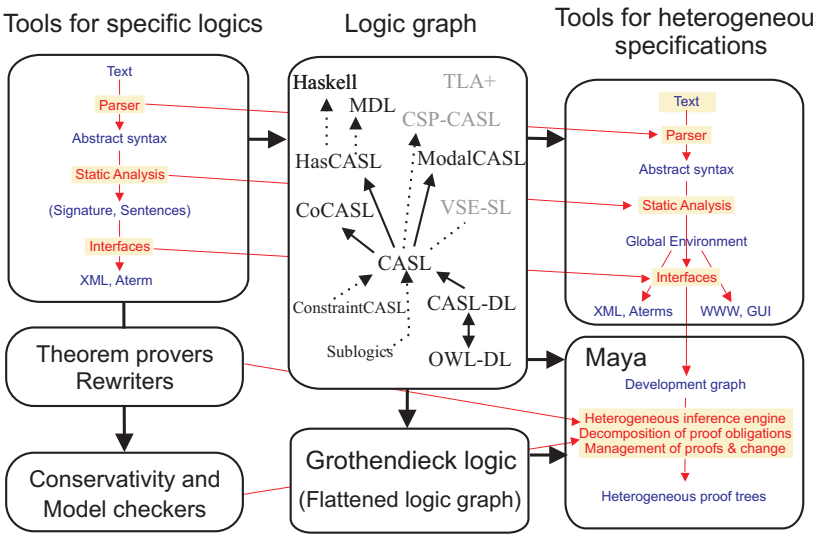


Fig. 26.15. Architecture of the heterogeneous tool set

and this separation of concerns is essential to keep the tool manageable from a software engineering point of view.

Heterogeneous CASL (HETCASL; see [Mos04]) includes the structuring constructs of CASL, such as union and translation. A key feature of CASL is that syntax and semantics of these constructs are formulated over an arbitrary institution (i.e. also for institutions that are possibly completely different from first-order logic resp. the CASL institution). HETCASL extends this with constructs for the translation of specifications along logic translations.

Like MAYA (see Section 26.12), HETS provides a representation of structured specifications which are the logical basis for the *Complex theories* and *Development graphs* of OMDOC[16].

For proof management, MAYA's calculus of development graphs has been extended with hiding and adapted to heterogeneous specification. Development graphs provide an overview of the (heterogeneous) specification module hierarchy and the current proof state, and thus may be used for monitoring the overall correctness of a heterogeneous development.

HETS also provides a translation of CASL to and from a subset of OMDOC (namely some formal first-order subset). Future work aims at a deeper integration of HETS and OMDOC that provides a translation to and from OMDOC for each of the logics integrated in HETS. Moreover, OMDOC itself will become a "logic" (but only with syntax, without model theory) within

[16] These are the modules CTH and DG, respectively.

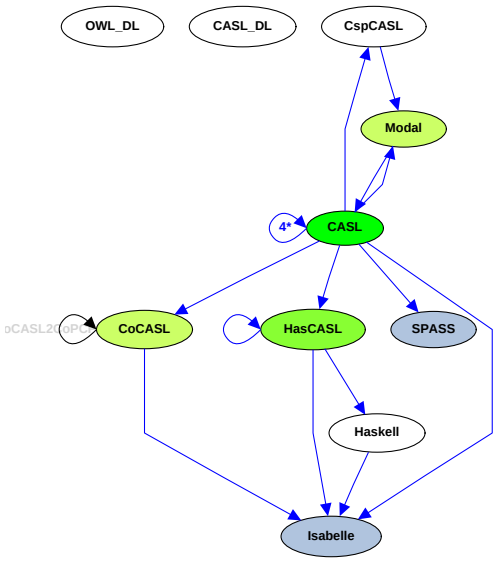


Fig. 26.16. Graph of logics currently supported by HETS. The more an ellipse is filled, the more stable is the implementation of the logic.

HETS, such that also informal OMDOC documents (or formal OMDOC documents written in a logic currently not available in HETS) will be manageable for HETS. In this way, the data formats of OMDOC and HETS will converge, such that tools e.g. for searching, versioning or management of change can be implemented uniformly for both.

26.14 CPoint: An OMDoc Editor in MS PowerPoint

Project Home	`http://kwarc.eecs.iu-bremen.de/software/CPoint/`
Authors	Andrea Kohlhase Digital Media in Education (DiMeB), Dept. of Mathematics and Computer Science, University Bremen

CPoint is an invasive, semantic OMDoc editor in MS PowerPoint (with an OMDoc outlet) that enables a user to distinguish between form and content in a document. As such it can be viewed as an authoring tool for OMDoc documents with a focus on their presentational potential. It enables a user to make implicit knowledge explicit. Moreover, it provides several added-value services to a content author in order to alleviate the short term costs of semantic mark up in contrast to its long term gains.

26.14.1 The CPoint Approach

CPoint started out as a part of the Course Capsules Project (CCaps) at Carnegie Mellon University (2001 — 2004), has been subsequently supported by the International University Bremen (2004) and is now developed further at the 'Digital Media in Education' group at Bremen University. **C**Point is distributed under the Gnu Lesser General Public License (LGPL) [Fre99]. The newest version can be downloaded from the project homepage.

PowerPoint (PPT) slides address exclusively the issue of presentation — the placement of text, symbols, and images on the screen, carefully sequenced and possibly animated or embellished by sound. This directly leads to the question: *What exactly is the content in a PPT presentation?*

Obviously, the text and the pictures carry content as does the textual, presentational, and placeholder structure. For instance the ordering of information by writing it in list form, grouping information bubbles in one slide, or marking text as title by putting it into a 'title' placeholder can be mapped directly onto the OMDoc `omgroup` and `metadata` elements. Unfortunately though, this content exploits neither OMDoc's theory level nor the statement or formula level in more than a very superficial way.

The 'real' content is hidden beneath the presentation form: the authors, lecturers, and audience know or learn this real content by **categorizing** what they see, and **combining** it with what they already know and presently hear. **C**Point stands for 'Content in PowerPoint'. It models this by providing the author with a tool to explicitly store the additional implicit knowledge with the PPT show itself and from within the PPT environment without destroying the presentational aspects of the PPT document. Moreover, **C**Point **converts** the additional content to the appropriate OMDoc levels, so that the resulting OMDoc document captures all content. For an author the semantic markup process is a long-term investment. In order to alleviate the author's costs, **C**Point has implemented several **added-value services**.

26.14.2 The CPoint Application

CPoint extends PPT's presentational functionalities by semantic ones to get a handle on its visible and invisible content. As an invasive editor (see [**?**]) **C**Point makes these semantic authoring tools available through a toolbar in the PPT menu (see Figure 26.17) where they are available whenever PPT is running. **C**Point is written in Visual Basic for Applications and can be distributed as a PPT add-in.

Fig. 26.17. The **C**Point Menu Bar

The top-level structure of a PPT presentation is given by slides. Each slide contains **PPT objects**, e.g. text boxes, shapes, images, or tables. These objects carry certain properties like text structure (e.g. ordered lists), document structure (e.g. being a title in the text hierarchy), or presentational structure (e.g. color, bold font, italic font, or symbol font). **C**Point enables the author to attach additional information to each PPT object. In particular, the author is empowered to transform implicit into explicit knowledge by categorizing, combining and enhancing these objects semantically.

Categorizing

The semantic annotation process typically starts with understanding an object's role in the to be transmitted knowledge and a subsequent categorization. The author selects the respective PPT object and assigns a suitable (didactic) role and category from a pre-defined list ranging from hard core mathematical categories like "Theory", "Definition", or "Assertion" to didactic elements like "Question" or "Comment". If a PPT object is part of a multi-part presentation (e.g. ranging over multiple slides) of a semantic entity, it can be marked as a sequel and inherits all information from previous parts. This way the PPT dependant linearity of the objects can be overcome.

Combining

For categorized PPT objects the author can input category specific content via the respective details form (see Figure 26.18 as an example for a PPT group categorized as "Axiom"). In particular, PPT objects can be assigned a relation via **C**Point's reference system. For instance, the axiom in Figure 26.18 sits in the theory called 'taxonomy of shapes'. A more sophisticated example would be a proof *for* an assertion that is constructed out of several, individual proof steps succeeding one another. Frequently, an author wants to reference implicit knowledge (e.g. theories can comprise entire concepts and as such are typically not explicitly presented in a lecture). Here, she can use **C**Point to

create abstract PPT objects called **abstract objects** that are invisible in the actual PPT show but can be dealt with like all other PPT objects.

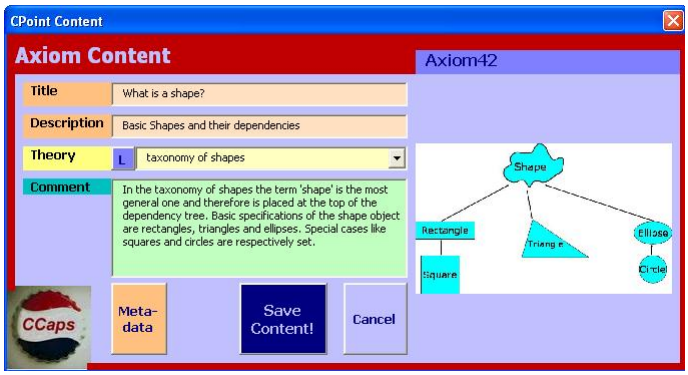


Fig. 26.18. The **C**Point Content Form for an Axiom Object

The information annotated in these processes can be exploited for added-value services.

OMDoc *Conversion*

The heart of **C**Point is the functionality for converting a fully (**C**Point-)edited presentation into a valid OMDoc document. This generated OMDoc document can for instance be read into computer-supported education systems like ActiveMath (see [MBA+01] and Section 26.8).

Added-Value Services

As author support is essential for the motivation doing the semantic markup process, **C**Point offers the following added-value services:

Content Search and Navigation **C**Point's GoTo facility makes use of the additional semantic quality of PPT objects by offering content search. For instance if an author remembers the existence of a definition of "equivalence" in some (older) PPT presentation, she might look up all PPT objects in a collection of several PPT presentations that are categorized as "Definition" and whose title contain the word "equivalence". The author is offered a list of all these objects and by selecting one she is directed to the specific PPT object.

Dependency Graphs CPointGraphs enables the user to view graph based presentations of the annotated knowledge on distinct detail levels.

Semantics-Induced Presentation The module CPointAuthor offers the presentation of the underlying semantics. Whenever the author selects a

PPT object basic semantic information (like category, title, and main references) is presented to her. With **C**Point's Visualize Mode semantic labels for annotated PPT objects are generated.

Creation of Pre-Categorized PPT Objects Based on an individually designed CSS style sheet categorized, styled PPT objects can be *created* with CPointAuthor. The layout is determined in the CSS file by the respective category (e.g. proposition) or superordinate classification (e.g. assertion, content, general).

Math Glyphs in PPT Based on the PPT add-in TexPoint, the CMath functionalities empower an author to define individual symbol presentations. **C**Point introduces a mathematical user interface, which fully integrates mathematical symbols into PowerPoint presentations based on the semantics of the underlying objects rather than simply generating appropriate ink marks. For instance, the author might categorize a PPT object as a symbol with the name 'reals' for the real numbers. The specific Unicode character to represent the real numbers can be declared with **C**Point. Subsequently, whenever the author writes the text '`\reals`' and activates the math mode, then this sequence of characters is replaced by the previously declared presentation. The symbol presentation may also be given in LaTeX form so that TexPoint can transform the LaTeX code into PPT glyphs. Note that this feature is not limited to math glyphs but can be used for handy abbreviations (macros) as well.

Editorial Notes Treating PPT presentations as content documents requires more editing, therefore CPointNotes add editorial functionalities like grouped editorial notes and navigation within these.

OMDoc To PPT The CPointImport module enables the import of OMDoc documents into the PPT application. According to an individual underlying CSS style sheet PPT objects in a newly created PPT presentation are generated.

ActiveMath Integrated development environment for ActiveMath content and specific ActiveMath book creation for a selected PPT object.

26.14.3 Future Work

In the future the addition of other added-value services for users is planned. We want to shift the focus from the authoring role to the recipient role of a PPT presentation, e.g. in form of a CPointStudent module in accordance with the CPointAuthor module. Furthermore, a new, more basic and therefore more user-friendly interface for **C**Point novices will be implemented. This CPointBasic module will try to overcome the heavily form-oriented format of **C**Point. In a next step the growing of a **C**Point user will be supported by offering advanced **C**Point utilities that will extend CPointBasic. Additionally, the success of "social software" under the Web 2.0 paradigm like "social bookmarking" gives rise to the idea of a new personal and sharable PPT objects management where the predefined categories in **C**Point are replaced by

"social tags". Another **C**Point project is its extension for usage by teachers in school, which usefulness has already been established in [Koh06]. The newest project at the International University of Bremen is the implementation of a **C**Point-like editor for MS Word.

26.15 sTeX: A LaTeX-Based Workflow for OMDoc

BErr(85)

Project Home	http://kwarc.iu-bremen.de/projects/stex/
Authors	Michael Kohlhase Computer Science, International University Bremen

EErr(85)

One of the reasons why OMDoc has not been widely employed for representing mathematics on the web and in scientific publications, may be that the technical communities that need high-quality methods for publishing mathematics already have an established method which yields excellent results — the TeX/LaTeX system. A large part of mathematical knowledge is prepared in the form of TeX/LaTeX documents.

We present sTeX (Semantic TeX) a collection of macro packages for TeX/LaTeX together with a transformation engine that transforms sTeX documents to the OMDoc format. sTeX extends the familiar and time-tried LaTeX workflow until the last step of Internet publication of the material: documents can be authored and maintained in sTeX using a simple text editor, a process most technical authors are well familiar with. Only the last (publishing) step (which is fully automatic) transforms the document into the unfamiliar XML world. Thus, sTeX can serve as a conceptual interface between the document author and OMDoc-based systems: Technically, sTeX documents are transformed into OMDoc, but conceptually, the ability to semantically annotate the source document is sufficient.

26.15.1 Recap of the TeX/LaTeX System

TeX [Knu84] is a document presentation format that combines complex page-description primitives with a powerful macro-expansion facility, which is utilized in LaTeX (essentially a set of TeX macro packages, see [Lam94]) to achieve more content-oriented markup that can be adapted to particular tastes via specialized document styles. It is safe to say that LaTeX largely restricts content markup to the document structure[17], and graphics, leaving the user with the presentational TeX primitives for mathematical formulae. Therefore, even though LaTeX goes a great step into the direction of a content/context markup format, it lacks infrastructure for marking up the functional structure of formulae and mathematical statements, and their dependence on and contribution to the mathematical context.

But the adaptable syntax of TeX/LaTeX and their tightly integrated programming features have distinct advantages on the authoring side:

- The TeX/LaTeX syntax is much more compact than OMDoc, and if needed, the community develops LaTeX packages that supply new functionality with a succinct and intuitive syntax.

[85] Erratum: The domain is kwarc.eecs.iu-bremen.de

[17] supplying macros e.g. for sections, paragraphs, theorems, definitions, etc.

- The user can define ad-hoc abbreviations and bind them to new control sequences to structure the source code.
- The TEX/LATEX community has a vast collection of language extensions and best practice examples for every conceivable publication purpose. Additionally, there is an established and very active developer community that maintains these.
- A host of software systems are centered around the TEX/LATEX language that make authoring content easier: many editors have special modes for LATEX, there are spelling/style/grammar checkers, transformers to other markup formats, etc.

In other words, the technical community is heavily invested in the whole workflow, and technical know-how about the format permeates the community. Since all of this would need to be re-established for an OMDoc-based workflow, the community is slow to take up OMDoc over TEX/LATEX, even in light of the advantages detailed in this book.

26.15.2 A LATEX-based Workflow for XML-based Mathematical Documents

An elegant way of sidestepping most of the problems inherent in transitioning from a LATEX-based to an XML-based workflow is to combine both and take advantage of the respective values.

The key ingredient in this approach is a system that can transform TEX/LATEX documents to their corresponding XML-based counterparts. That way, XML-documents can be authored and prototyped in the LATEX workflow, and transformed to XML for publication and added-value services.

There are various attempts to solve the TEX/LATEX to XML transformation problem; the most mature is probably Bruce Miller's LaTeXML system [Mil07]. It consists of two parts: a re-implementation of the TEX analyzer with all of its intricacies, and an extensible XML emitter (the component that assembles the output of the parser). Since LATEX style files are (ultimately) programmed in TEX, the TEX analyzer can handle all TEX extensions[18], including all of LATEX. Thus the LaTeXML parser can handle all of TEX/LATEX, if the emitter is extensible, which is guaranteed by the LaTeXML binding language: To transform a TEX/LATEX document to a given XML format, all TEX extensions must have "LaTeXML bindings", i.e. directives to the LaTeXML emitter that specify the target representation in XML.

The sTEX system that we present here supplies a set of TEX/LATEX packages and the respective LaTeXML bindings that allow to add enough structural information in the TEX/LATEX sources, so that the LaTeXML system can transform them into documents in OMDoc format.

[18] i.e. all macros, environments, and syntax extensions used int the source document

26.15.3 Content Markup of Mathematical Formulae in TEX/LATEX

The main problem here is that run-of-the-mill TEX/LATEX only specifies the presentation (i.e. what formulae look like) and not their content (their functional structure). Unfortunately, there are no universal methods (yet) to infer the latter from the former. Consider for instance the following "standard notations"[19] for binomial coefficients: $\binom{n}{k}$, ${}_nC^k$, $\mathcal{C}^n_k$, and $\mathcal{C}^k_n$ all mean the same thing: $\frac{n!}{k!(n-k)!}$. This shows that we cannot hope to reliably recover the functional structure (in our case the fact that the expression is constructed by applying the binomial function to the arguments n and k) from the presentation alone short of understanding the underlying mathematics.

The apparent solution to this problem is to dump the extra work on the author (after all she knows what she is talking about) and give her the chance to specify the intended structure. The markup infrastructure supplied by the STEX collection lets the author do this without changing the visual appearance, so that the LATEX workflow is not disrupted. We speak of **semantic preloading** for this process. For instance, we can now write

```
\CSum{k}1\infty{\Cexp{x}k}  instead of  \sum_{k=1}^\infty x^k    (26.1)
```

for the mathematical expression $\sum_{k=1}^{\infty} x^k$. In the first form, we specify that we are applying a function (`CSumLimits` $\hat{=}$ sum with limits) to four arguments: (*i*) the bound variable k (*ii*) the number 1 (*iii*) ∞ (*iv*) `\Cexp{x}k` (i.e. x to the power k). In the second form, we merely specify hat LATEX should draw a capital sigma character (Σ) whose subscript is the equation $k = 1$ and whose superscript is ∞. Then it should place next to it an x with an upper index k.

Of course human readers (who understand the math) can infer the content structure from the expression $\sum_{k=1}^{\infty} x^k$ of the right-hand representation in (26.1), but a computer program (who does not understand the math or know the context in which it was encountered) cannot. However, a converter like LaTeXML can infer this from the left-hand LATEX structure with the help of the curly braces that indicate the argument structure. This technique is nothing new in the TEX/LATEX world, we use the term "**semantic macro**" for a macro whose expansion stands for a mathematical object. The STEX collection provides semantic macros for all Content-MathML elements together with LaTeXML bindings that allow to convert STEX formulae into MathML.

26.15.4 Theories and Inheritance of Semantic Macros

Semantic macros are traditionally used to make TEX/LATEX code more portable. However, the TEX/LATEX scoping model (macro definitions are scoped either

[19] The first one is standard e.g. in Germany and the US, the third one in France, and the last one in Russia

in the local group or to the end of the document), does not mirror mathematical practice, where notations are scoped by mathematical environments like statements, theories, or such (see [Koh05b] for a discussion and examples). Therefore the sTeX collection provides an infrastructure to define, scope, and inherit semantic macros.

In a nutshell, the sTeX `symdef` macro is a variant of the usual `newcommand`, only that it is scoped differently: The visibility of the defined macros is explicitly specified by the `module` environment that corresponds to the OMDoc `theory` element. For this the `module` environment takes the optional `KeyVal` arguments `id` for specifying the theory name and `uses` for the semantic inheritance relation. For instance a `module` that begins with

```
\begin{module}[id=foo,uses={bar,baz}]
```

restricts the scope of the semantic macros defined by the `\symdef` form to the end of this module given by the corresponding `\end{module}`, and to any other `module` environment that has `[uses={...,foo,...}]` in its declaration. In our example the semantic macros from the modules `bar` and `baz` are inherited as well as the ones that are inherited by these modules.

We will use a simple module for natural number arithmetics as an example. It declares a new semantic macro for summation while drawing on the basic operations like + and − from LaTeX. `\Sumfromto` allows us to express an expression like $[\sum_{i=1}^{n} 2i-1]_r^{86}$ as `\Sumfromto{i}1n{2i-1}`. In this example we have also made use of a local semantic symbol for n, which is treated as an arbitrary (but fixed) symbol (compare with the use of `\arbitraryn` below, which is a new — semantically different — symbol). Err(86)

```
\begin{module}[id=arith]
  \symdef{Sumfromto}[4]{\sum_{#1=#2}^{#3}{#4}}
  \symdef[local]{arbitraryn}{n}
4 What is the sum of the first $\arbitraryn$ odd numbers, i.e.
  $\Sumfromto{i}1\arbitraryn{2i-1}?$
\end{module}
```

is formatted by sTeX to

What is the sum of the first n odd numbers, i.e. $\sum_{i=1}^{n} 2i-1$?

Moreover, the semantic macro `Sumfromto` can be used in all `module` environments that import it via its `uses` keyword. Thus sTeX provides sufficient functionality to mark up OMDoc theories with their scoping rules in a very direct and natural manner. The rest of the OMDoc elements can be modeled by LaTeX environments and macros in a straightforward manner.

The sTeX macro packages have been validated together with a case study [Koh05b], where we semantically preloaded the course materials for a two-semester course "General Computer Science I&II" at International University Bremen and transform them to the OMDoc, so that they can be used in the ActiveMath system (see Section 26.8).

[86] Erratum! correct example given (original text was: "$\sum_{i=1}^{n} x^i$")

26.16 An Emacs mode for editing OMDoc Documents

Project Home	`http://www.cs.cmu.edu/~ccaps`
Authors	Peter Jansen School of Computer Science, Carnegie Mellon University

We describe an EMACS major mode for editing OMDOC documents, developed by the COURSE CAPSULES project group at the CMU School of Computer Science. This mode extends the EMACS editor [Sta02] with functionality intended to help visualize, edit, and create documents written in OMDOC format.

The mode is part of the OMDOC distribution (see Section 23.2), it is provided under the conditions specified in the Library Gnu Public License [Fre99].

26.16.1 Introduction

The CCaps project has developed tools to convert legacy materials written in a variety of formats (POWERPOINT, MATHEMATICA®, etc.) into the OMDOC format (see Sections 26.14 and 26.17). In many cases the output generated by such tools needs to be post-processed or otherwise modified.

To this end, a user must open the file, read and understand its contents and perform the appropriate modifications. Though an OMDOC document is a regular text file, most of its content consists of markup, which is hard to read and tedious to type. It is therefore important to support the user with tools that make a document easier to read and modify, either in the form of a separate editor, or as an extension of an existing editor.

One approach to this is to build a *visual* OMDOC *editor*, which presents the document in a form resembling conventional mathematical documents (i.e., without showing the markup explicitly, and with appropriate formatting for mathematical formulae), and offers the user functionality to modify or annotate its content.

While this is ideal for user understanding of document content, it presupposes consistent syntactic correctness, makes it more difficult to inspect or change markup directly, and may present challenges as to resolving user action ambiguities.

We have taken this approach in the **C**Point and MATHEMATICA® addins (see Sections 26.14 and 26.17). But we also wanted a tool that would maintain full control of all the textual information, while offering support for readability and editing functionality. We chose for this tool to extend the EMACS editor [Sta02], which lends itself very well to this task (as well as being the editor for general use for several of our group).

26.16.2 OMDoc mode functionality

We now look at the different categories of functionality in slightly more detail.

Visualization

is currently provided by the use of the EMACS *font-lock* mechanism to give different categories of tags and content different fonts and colors to make them easily recognizable. Element categories currently recognized correspond to the OMDOC 1.2 modules: Document structure, Math, Theories, Auxiliaries, Presentation, OPENMATH, and the Dublin Core elements.

A customizable *indentation function* allows for intelligible layout, which is helpful both in hand-coding and the editing of the output of a legacy transformation process. There are key bindings for line, region, and enclosing element indentation.

Editing

Functionality consists mainly of automated insertion of *templates* for each of the OMDOC elements, both via mode-dependent menu options and key bindings, grouped by element category (the same categories as given above).

The template insertion mechanism is based on `tempo.el`, which allows for the maintenance of a *list of insertion points* the user can navigate in between to supply or change the values of certain attributes.

The main function currently available for completing incomplete elements is the equivalent of the standard `electric-/` function. We are planning to add several other completion functions in the near future (for tags, tag sets, attribute names, and symbol and theory names).

The mode also provides for *validation*: either internally (as a simple local syntax check to check well-formedness) or externally (via an external xml validation validation engine). Internal validation builds an abbreviated parse tree, and highlights discrepancies, suggesting possible modifications of insertions of element opening or closing tags. External validation runs an external xml validation engine (RXP or nsgmls, depending on the configuration variables), and shows the output in a separate buffer.

Document creation

is supported by automatic insertion of a basic OMDOC *skeleton* in new buffers as well as a *time-stamp updating* mechanism and some smaller functions that extract information from the user's environment variables to supply information for some of the metadata slots (see the example in Figure 26.19 below).

26.16.3 Examples

We illustrate some of the above by means of a few screen shots. The example in Figure 26.20 is taken while editing a document that was semi-automatically generated from part of a MATHEMATICA® notebook ([Sut06]). Here, the user has already run an automated indentation function, for example by activating

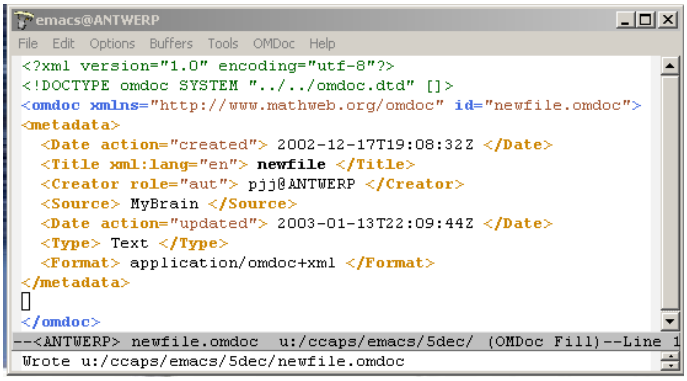


Fig. 26.19. Opening a new Buffer in OMDoc Mode

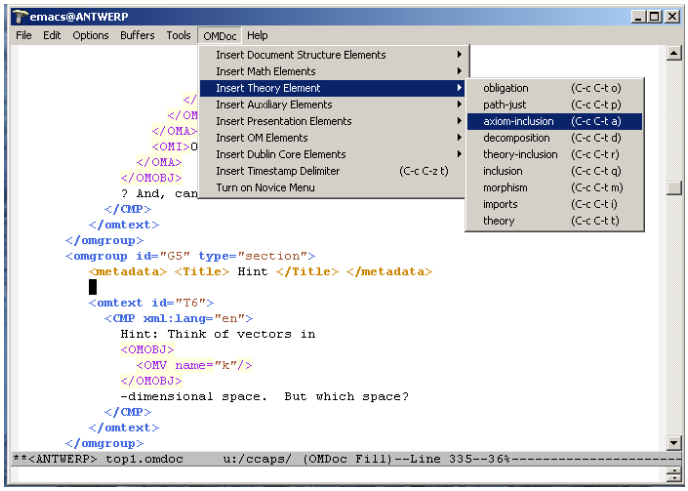


Fig. 26.20. Editing an OMDoc Document

`omdoc-indent-enclosing-main` by typing `C-c C-q`, and is now about to use the OMDoc menu to enter a new construct.

After this operation, which could also have been performed by typing the key sequence `C-c C-t a`), Emacs inserts the following text at the point (i.e. cursor position).

```
<axiom-inclusion xml:id="" to=""> </axiom-inclusion>
```

The second example shows the skeleton template that is automatically inserted when the user opens a new file: Figure 26.19. Note that the file name has been used as id and title automatically, and the user's address appears in the Author field. Timestamps are inserted in Date fields for both creation

and update, and the latter is adjusted automatically every time changes are saved to the file.

26.17 Converting Mathematica Notebooks to OMDoc

Project Home	`http://www.cs.cmu.edu/~ccaps`
Authors	Klaus Sutner School of Computer Science, Carnegie Mellon University

We describe a tool that converts MATHEMATICA® notebooks to OMDOC. The program is implemented entirely in MATHEMATICA® and easily extensible.

Creating an editor for general mathematical documents is notoriously difficult, in particular when input methods are required that mimic the traditional two-dimensional layout of many formulae. Thus, it seems natural to use an existing high-quality system such as the MATHEMATICA® notebook front end as an authoring tool for mathematical documents. A considerable amount of effort has gone into the design of this front end, see for example [Wol00], resulting in a surprisingly versatile system. The notebook front end provides a rich set of palettes that allow inexperienced users to construct complicated expressions almost instantaneously. For more advanced users there is a well-thought-out set of keyboard operations that make it possible to create, navigate and edit two-dimensional expressions with relative ease and without recourse to time-consuming mouse-based operations. Unlike with TEX, the results are immediately visible and corrections are easy to make. Nonetheless, the quality of the typeset expression approaches that of TEX. Last, but not least, the MATHEMATICA® kernel can be used to generate complicated expressions and even whole notebooks automatically.

MATHEMATICA® provides significant support for import, export and manipulation of XML documents and expressions, see [Wol02]. Thus, one can export a notebook in MATHML format, or in a special NOTEBOOKML format. Unfortunately, these export mechanisms cannot be modified directly to produce highly marked-up documents in OMDOC format.

The NB2OMDOC converter uses a recursive descent parser, that scans the given notebook document and generates corresponding OMDOC. As far as structured text is concerned this is a fairly straightforward operation. However, special care needs to be taken to deal with mathematical text elements, such as definitions, theorems, proofs and such like, and mathematical expressions, in inline format as textual elements as well as in evaluatable format (as input for the MATHEMATICA® kernel). We comment on both issues in turn.

MATHEMATICA® notebooks provide reasonable support for the creation of well-structured documents, but enforce no particular discipline. A fragment of a typical notebook, showing some section headers and a bit of text with inline mathematical formulae is shown in Figure 26.21.

In order to facilitate the translation process it is advisable to front-load the process: the author of the notebook is encouraged to use a special notebook stylesheet, `OMDocStyle.nb`, that defines a number of syntactic categories

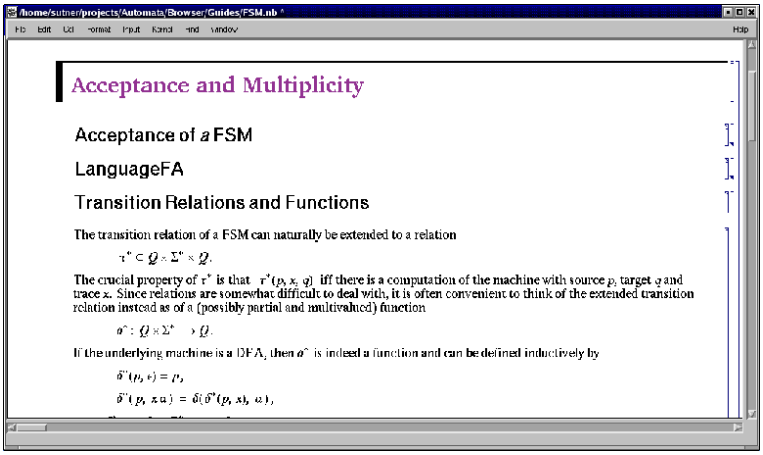


Fig. 26.21. A MATHEMATICA® Notebook

normally absent in a notebook. These categories are implemented as a combination of the cell types and cell labels. As a typical example, consider a proof of some assertion such as a theorem. Ordinarily, a sequence of plain text cells would be used to express a proof since none of the standard MATHEMATICA® stylesheets provide a special proof style–though some have a theorem style. The elements defined in `OMDocStyle.nb` are easily accessible via pulldown menus or via keyboard shortcuts in the notebook front end. Moreover, the special styles are color-coded in the notebook, so that it is easy for the author to see which elements are present and which might be missing.

The conversion of mathematical expressions in the notebook is accomplished in a two-step procedure. First, we use the built-in MATHEMATICA® operation `ExpressionToSymbolicMathML` that produces a symbolic expression representing a MathML term that corresponds to the original notebook expression. In a second, post-processing step this expression is then transformed into an OPENMATH expression. The post-processing relies heavily on the sophisticated pattern matching mechanism in MATHEMATICA® and uses a special collection of rewrite rules. The rules are based on fairly simple-minded heuristics but do produce adequate results so long as the starting expression is not too complicated. As an example, consider the simple polynomial expression $ax^2 + bx + c$ whose internal representation in MATHEMATICA® looks like so (we assume here the expression appears inline within a block of text, the situation for an input expression is entirely similar):

```
Cell[ BoxData[FormBox[RowBox[{
      RowBox[{"a", " ", SuperscriptBox["x", "2"]}], "+", " ",
      RowBox[{"b", " ", "x"}], " ", "+", " ", "c"}],
4       TraditionalForm]]]
```

The first conversion step produces the following MATHEMATICA® expression, shortened here to save space:

```
XMLElement["math",
  {"xmlns" ->"http://www.w3.org/1998/Math/MathML"},
  {XMLElement[ "apply", {}, {XMLElement["plus", {}, {}],
   XMLElement[ "apply", {}, {XMLElement["times", {}, {}],
     XMLElement["ci", {}, {"a"}],
     XMLElement[ "apply", {}, {XMLElement["power", {}, {}],
       XMLElement["ci", {}, {"x"}],
       XMLElement["cn", {"type"->"integer"}, {"2"}]}]}], ... }]}]
```

The post-processing finally yields the following expression, again shown only in part.

```
XMLElement["OMOBJ", {}, {XMLElement["OMA", {},
    {XMLElement["OMS",
          {"cd"->"arith1", "name"->"plus"}, {}],
       XMLElement["OMA", {}, {XMLElement[ "OMS",
          {"cd"->"arith1", "name"->"times"}, {}],
       XMLElement["OMV", {"name"->"a"}, {}],
       XMLElement["OMA", {}, {XMLElement["OMS",
          {"cd"->"arith1", "name"->"power"}, {}], ...}]}]}]}]
```

The content dictionary was properly guessed in this instance. Judging from the limited experiments we have undertaken so far, it seems reasonable to expect that a fair amount of the translation can be automated given that the field of discourse is limited, and that the author is willing to customize the rewrite rules that control the post-processing step. Fortuitously, very little knowledge of MATHEMATICA® programming beyond some basic syntax is necessary for the creation of these rules; mathematicians are likely to find these rules fairly intuitive and natural.

At present, the conversion program is somewhat limited in its ability to deal with arbitrarily structured notebooks. It works well with a suite of notebooks developed specifically for the `OMDocStyle.nb`, but requires modification for other types of notebooks. While it is not our goal to provide a truly general conversion tool with a scope comparable to, say, the built-in conversion to MathML, some generalizations are still needed at this point.

Another crucial issue is the extension of the rewrite rules used in the post-processing step leading from MathML to OPENMATH. No effort has been made so far to systematically generate a set of rules suitable for a large class of documents. At the very least, an extension mechanism is needed that makes it easy for non-expert users to create the necessary rule tables.

Lastly, it is desirable to create a MATHEMATICA® palette-based tool that focuses more narrowly on the authoring and conversion of mathematical expressions only rather than whole notebooks. The generated raw OPENMATH expressions can be fed directly into a low-level editor such as EMACS using the special OMDOC mode created as part of the CCAPS project, see elsewhere in this volume for a description.

26.18 Standardizing Context in System Interoperability

Project Home	`http://omdoc.org/examples/logics`
Authors	Michael Kohlhase
	Computer Science, International University Bremen

In this project the OMDoc format is used as a content language for the protocol-based integration of mathematical software systems, where the systems offer mathematical services by publishing service descriptions and interoperate by exchanging computation requests and results. The mechanics of the communication and domain-independent part of meaning of these messages is given by a standardized 'interlingua' (which will not concern us here), a possible implementation of the transport layer we have seen in Chapter 9. Here we are interested in the mathematical objects contained in the messages

OMDoc can help with the task of making mathematical objects interoperable, as we have seen in series of experiments of connecting the theorem proving systems ΩMEGA [BCF+97], InKa [HS96], Pvs [ORS92], *λClam* [RSG98b], TPS [ABI+96], and CoQ [Tea] to the MBase system by equipping them with an OMDoc interface. As expected, OpenMath and Content-MathML solve the problem of syntactically standardizing the representation of mathematical objects. For a semantic interoperability we also need to capture their context. This is not a problem for Content-MathML, as the context is already standardized in the MathML recommendation. For OpenMath, the context is given by the set of content dictionaries in use for representing the mathematical objects. Nevertheless mathematical software systems — such as computer algebra systems, visualization systems, and automated theorem provers — come with different conceptualizations of the mathematical objects (see [**?**] for a discussion). This has been in principle solved by supplying a flexible and structured theory level in the form of OMDoc content dictionaries that define necessary mathematical concepts (see Subsection 26.18.1 for practical considerations). For systems like theorem provers or theory development environments, where the mathematical objects are axioms, definitions, assertions, and proofs there is another problem: that of standardizing the logical language, which we will discuss in Subsection 26.18.2.

26.18.1 Context Interoperability via Theory Morphisms

As an example for the integration of two mathematical software systems we look at the task of integrating the Pvs and ΩMEGA set theory libraries. This is simpler than e.g. integrating the computer algebra systems Maple™ and Mathematica®, since all the conceptualizations and assumptions are explicitly given, but gives an intuition for the difficulties involved. We summarize the situation in Figure 26.22, where we compare symbol names for set theory concepts in the two systems. The general problem in such an integration of mathematical software systems consists in their independent growth over

PVS	ΩMEGA	PVS	ΩMEGA
set		subset?	subset
member	in		subset2
empty?	empty	strict_subset?	proper-subset
emptyset	emptyset		superset
nonempty?	not-empty	union	union
full?			union2
fullset			union-over-collection
singleton?	singleton	intersection	intersection
singleton			intersection-over-coll.
complement	set-complement	disjoint?	misses
difference	setminus	meets	
symmetric_difference		add	add-one
	exclunion	remove	

Fig. 26.22. Set Theories in ΩMEGA and PVS

time, leading to differing names, definitions, theory boundaries, and possibly conceptualizations. Most of these particulars are artefacts of constraints imposed by the system (e.g. file lengths). In this situation theory interpretations suggest themselves as a means for theory integration: We can use theory interpretations to establish inclusion into a suitably constructed **integration theory**. In Figure 26.23 we have executed this for the set theory libraries of the systems PVS, ΩMEGA, TPS, and IMPS; we provide an 'integration theory' `mbase:sets` — it provides rationally reconstructed versions of all concepts encountered in the system's libraries — and a set of theory inclusions ρ_* that interpret the system concepts in terms of `mbase:sets`. Note that since the ρ_* are monomorphisms, we can factor any existing theory inclusion (e.g. `pvs:sets` to `pvs:funcs` highlighted in Figure 26.22) via the integration theory, using the partial inverse ρ_*^{-1} of ρ_*. For an integration of a set of software systems this refactoring process is repeated recursively from terminal- to initial nodes in the `imports` relation.

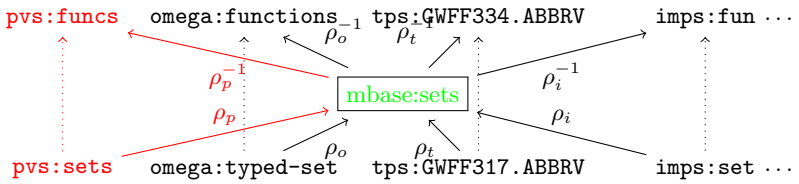


Fig. 26.23. Theory Translations for System Integration

Note that *technically* we do not need to change the interface language of the mathematical software systems[20], we only rationally reconstruct their

[20] This is important if we want to integrate proprietary software systems, where we have no control over the interfaces.

meaning in terms of the new integration theory, which can act as a gold standard for the integration. *Socially* the existence of the new standard theory may prompt a migration to the nomenclature and coverage of the integration theory. Note furthermore, that we have only treated the simple case, where the mathematical conceptualizations underlying the software systems are already explicitly given in a library. For many mathematical software systems the underlying conceptualizations and assumptions are only documented in scientific papers, user manuals, or inscribed into the code. For such systems, **interface theories** that make them explicit have to be developed to pursue the integration strategy presented above. Of course, this process needs a lot of manual labor, but leads to true interoperability of mathematical software systems, which can now re-use the work of others.

Finally note that the integration only works as smoothly as in our scenario, if the systems involved make assumptions about mathematical objects that are compatible with each other. In most cases, incompatibilities can be resolved by renaming concepts apart, e.g. one system considers set union to be a binary operation, while the other considers it as n-ary. Here, the integration theory would supply two distinct (though possibly semantically related) concepts; The theory-based integration approach allows to explicitly disambiguate the concepts and thus prevent confusion and translation errors. In very few cases, systems are truly incompatible e.g. if one assumes an axiom which the other rejects. In this case the theory based integration approach breaks down — indeed a meaningful integration seems impossible and unnecessary.

26.18.2 A Hierarchy of Logical Languages

In the example above we made use of the fact that theorem proving systems are simpler to deal with than other mathematical software systems, since they encode the underlying assumptions explicitly into mathematical libraries. Unfortunately though, this is only partially true the underlying base logics are usually not treated in this way. Fortunately, logical concepts are treated in OMDoc just like ordinary ones: by content markup in Content-MathML or OpenMath, so that there is no fundamental barrier to treating them as the theory contexts above; we only have to come up with interface theories for them. We have done just that when we equipped various logic-based systems with OMDoc interfaces observing that even though the systems are of relatively different origin, their representation languages share many features:

- TPS and Pvs are based on a simply typed λ-calculus and only use type polymorphism in the parsing stage, whereas Ωmega and λ*Clam* allow ML-style type polymorphism.
- Ωmega, InKa and Pvs share a higher sort concept, where sorts are basically unary predicates that structure the typed universe.
- Pvs and CoQ allow dependent- and record types as basic representational features.

but also differ on many others: for instance InKa, Pvs, and CoQ explicitly support inductive definitions, but by very different mechanisms and on differing levels. CoQ uses a constructive base logic, whereas the other systems are classical. The similarities are not that surprising, all of these systems come from similar theoretical assumptions (most notably the Automath project [dB80]), and inherit the basic setup (typed λ-calculus) from it. The differences can be explained by differing intuitions in the system design and in the intended applications.

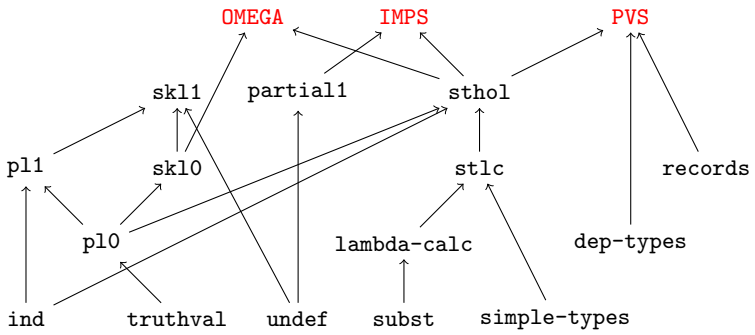


Fig. 26.24. A Hierarchy of Logical Languages

We have started to provide a standardized, well-documented set of content dictionaries for logical languages in the OMDoc distribution. These are organized hierarchically, as depicted in Figure 26.24. In essence, the structured theory mechanism in OMDoc is used to create a language hierarchy that inter-relates the various representation formats of existing theorem provers. For instance the simply typed λ-calculus can be factored out (and thus shared) of the representation languages of all theorem proving systems above. This makes the exchange of logical formulae via the OMDoc format very simple, if they happen to be in a suitable common fragment: In this case, the common (OpenMath/OMDoc) syntax is sufficient for communication.

26.18.3 Logic Interoperability via Logic Morphisms

In theoretical accounts of the integration of logical languages, one finds categorical accounts like the one described in Section 26.13 or proof-theoretic ones based on definitions like the one below. Both mesh well with the OMDoc representation format and its theory level; we will show this for the proof-theoretic account here.

Definition 26.1. *A* ***logical system*** $\mathcal{S} = (\mathcal{L}, \mathcal{C})$ *consists of a language* $\mathcal{L}$ *(i.e. a set of well-formed formulae) and a calculus* $\mathcal{C}$ *(i.e. a set of inference rules). A calculus gives us a notion of a* $\mathcal{C}$*-derivation of* $\mathbf{A}$ *from* $\mathcal{H}$*, which*

we will denote by $\mathcal{D}\colon \mathcal{H} \vdash_{\mathcal{C}} \mathbf{A}$. *Let* $\mathcal{S}$ *and* $\mathcal{S}'$ *be logical systems, then a* ***logic morphism*** $\mathcal{F}\colon \mathcal{S} \to \mathcal{S}'$ *consists of a* ***language morphism*** $\mathcal{F}^{\mathcal{L}}\colon \mathcal{L} \to \mathcal{L}'$ *and a* ***calculus morphism*** $\mathcal{F}^{\mathcal{D}}$ *from* $\mathcal{C}$*-derivations to* $\mathcal{C}'$*-derivations, such that for any* $\mathcal{C}$*-derivation* $\mathcal{D}\colon \mathcal{H} \vdash_{\mathcal{C}} \mathbf{A}$ *we have* $\mathcal{F}^{\mathcal{D}}(\mathcal{D})\colon \mathcal{F}^{\mathcal{L}}(\mathcal{H}) \vdash_{\mathcal{C}'} \mathcal{F}^{\mathcal{L}}(\mathbf{A})$.

The intuition behind this is that logic morphisms transport proofs between logical systems. Logic morphisms come in all shapes and sizes, a well-known one is the relativization morphism from sorted logics to unsorted ones, for instance the morphism $\mathcal{R}$ from sorted first-order logic ($\mathbb{S}FOL$) to unsorted first-order logic (FOL). For every sorted constant $\mathcal{R}$ introduces an axiom e.g. $\mathcal{R}([+\colon \mathbb{N} \to \mathbb{N} \to \mathbb{N}]) = \forall X, Y.\mathbb{N}(X) \wedge \mathbb{N}(Y) \Rightarrow \mathbb{N}(X + Y)$. On formulae sorted quantifications are translated into unsorted ones guarded by sort predicates, e.g. $\mathcal{R}(\forall X_{\mathbb{B}}.\mathbf{A}) = \forall X.\mathbb{B}(X) \Rightarrow \mathcal{R}(\mathbf{A})$. Finally, for proofs we have the correspondence given in Figure 26.25, where $\mathbb{A}, \mathbb{B}, \ldots$ are sort symbols.

$$\mathcal{R}\left(\frac{\mathbf{A}\colon \mathbb{B} \to \mathbb{C} \quad \mathbf{B}\colon \mathbb{B}}{\mathbf{AB}\colon \mathbb{C}}\right) = \cfrac{\cfrac{\forall X.\mathbb{B}(X) \Rightarrow \mathbb{C}(\mathbf{A}X)}{\mathbb{B}(\mathbf{B}) \Rightarrow \mathbb{C}(\mathbf{AB})} \quad \mathbb{B}(\mathbf{B})}{\mathbb{C}(\mathbf{AB})}$$

$$\mathcal{R}\left(\frac{\forall X_{\mathbb{B}}.\mathbf{A} \quad \mathbf{B}\colon \mathbb{B}}{[\mathbf{B}/X]\mathbf{A}}\right) = \cfrac{\cfrac{\forall X.\mathbb{B}(X) \Rightarrow \mathcal{R}(\mathbf{A})}{\mathbb{B}(\mathcal{R}(\mathbf{B})) \Rightarrow \mathcal{R}([\mathbf{B}/X]\mathbf{A})} \quad \mathbb{B}(\mathbf{B})}{\mathcal{R}([\mathbf{B}/X]\mathbf{A})}$$

Fig. 26.25. Relativization Morphism on Proofs

In Definition 26.1 a logical system is a two-partite object consisting of a language and a calculus. In the ontologically promiscuous OMDoc format both parts are represented largely like ordinary mathematically concepts. The notable exception is that proofs have a slightly dual representation, but inference rules of a calculus are still represented as symbols via the Curry-Howard isomorphism (see Chapter 17). Thus a logical system can be represented as an OMDoc theory as we did above, moreover, the logic morphism $\mathcal{R}$ can simply be encoded as a theory inclusion from $\mathbb{S}FOL$ to FOL mapping $\mathbb{S}FOL$ constants for inference rules to FOL terms for proofs. The condition on the form of derivations in Definition 26.1 now simply takes on the form of a type compatibility condition.

26.19 Integrating Proof Assistants as Plugins in a Scientific Editor

Project Home	`http://www.ags.uni-sb.de/~omega/projects/verimathdoc`
Authors	Serge Autexier, Christoph Benzmüller, Armin Fiedler, and Henri Lesourd Computer Science, Saarland University, Saarbrücken, Germany

In contrast to computer algebra systems (CASs), mathematical proof assistance systems have not yet achieved considerable recognition and relevance in mathematical practice. One significant shortcoming of the current systems is that they are not fully integrated or accessible from within standard mathematical text-editors and that therefore a duplication of the representation effort is typically required. For purposes such as tutoring, communication, or publication, the mathematical content is in practice usually encoded using common mathematical representation languages by employing standard mathematical editors (e.g., LaTeX and Emacs). Proof assistants, in contrast, require fully formal representations and they are not yet sufficiently linked with these standard mathematical text editors. Therefore, we have decided to extend the mathematical text editor TeXmacs [dH01] in order to provide direct access from it to the mathematics assistance system Ωmega [SBB+02, SBA05]. Generally, we aim at an approach that is not dependent on the particular proof assistant system to be integrated [ABFL06].

TeXmacs [dH01] is a scientific WYSIWYG text editor that provides professional typesetting and supports authoring with powerful macro definition facilities like in LaTeX. The internal document format of TeXmacs is a Scheme S-expression composed of TeXmacs specific markup enriched by definable macros. The full access to the document format together with the possibility to define arbitrary Scheme functions over the S-expressions makes TeXmacs an appropriate text editor for an integration with a mathematical assistance system.

The mathematical proof assistance system Ωmega [SBB+02, SBA05] provides proof development at a high level of abstraction using knowledge-based proof planning and the proofs developed in Ωmega can be verbalized in natural language via the proof explanation system *P.rex* [Fie01b]. As the base calculus of Ωmega we use the Core calculus [Aut03, Aut05], which supports proof development directly at the *assertion level* [Hua96], where proof steps are justified in terms of applications of definitions, lemmas, theorems, or hypotheses (collectively called *assertions*).

Now, consider a teacher, student, engineer, or mathematician who is about to write a new mathematical document in TeXmacs. A first crucial step in our approach is to link this new document to one or more mathematical theories provided in a mathematical knowledge repository. By providing such a

link the document is initialized and TEXMACS macros for the relevant mathematical symbols are automatically imported; these macros overload the pure syntactical symbols and link them to formal semantics. In a TEXMACS display mode, where this additional semantic information is hidden, the user may then proceed in editing mathematical text as usual. The definitions, lemmas, theorems and especially their proofs give rise to extensions of the original theory and the writing of some proof goes along with an interactive proof construction in ΩMEGA. The semantic annotations are used to *automatically* build up a corresponding formal representation in ΩMEGA, thus avoiding a duplicated encoding effort of the mathematical content. Altogether this allows for the development of mathematical documents in professional type-setting quality which in addition can be formally validated by ΩMEGA, hence obtaining *verified mathematical documents*.

Using TEXMACS's macro definition features, we encode theory-specific knowledge such as types, constants, definitions and lemmas in macros. This allows us to translate new textual definitions and lemmas into the formal representation, as well as to translate (partial) textbook proofs into formal (partial) proof plans.

Rather than developing a new user interface for the mathematical assistance system ΩMEGA, we adapt ΩMEGA to serve as a mathematical service provider for TEXMACS. The main difference is that instead of providing a user interface only for the existing interaction means of ΩMEGA, we extend ΩMEGA to support requirements that arise in the preparation of a semi-formal mathematical document. In the following we present some requirements that we identified to guide our developments.

The mathematical document should be prepared directly in interaction with ΩMEGA. This requires that (1) the semantic content of the document is accessible for a formal analysis and (2) the interactions in either direction should be localized and aware of the surrounding context.

To make the document accessible for formal analysis requires the extraction of the semantic content and its encoding in some semi-formal representation suitable for further formal processing. Since current natural language analysis technology cannot yet provide us with the support required for this purpose, we use semantic annotations in the TEXMACS document instead. Since these semantic annotations must be provided by the author, one requirement is to keep the burden of providing the annotations as low as possible.

Due to their formal nature the representations of mathematical objects, for instance, definitions or proofs, in existing mathematical assistance systems are very detailed, whereas mathematicians omit many obvious or easily inferable details in their documents: there is a big gap between common mathematical language and formal, machine-oriented representations. Thus another requirement to interfacing TEXMACS to ΩMEGA is to limit the details that must be provided by the user in the TEXMACS document to an acceptable amount.

In order to allow both the user and the proof assistance system to manipulate the mathematical content of the document we need a common rep-

resentation format for this pure mathematical content implemented both in TEXMACS and in ΩMEGA. To this end we define a language S, which includes many standard notions known from other specification languages, such as terms, formulas, symbol declarations, definitions, lemmas and theorems. The difference to standard specification languages is that our language S (i) includes a language for proofs, (ii) provides means to indicate the logical context of different parts of a document by fitting the narrative structure of documents rather than imposing a strictly incremental description of theories as used in specification languages, and (iii) accommodates various aspects of underspecification, that is, formal details that the writer purposely omitted. Given the language S, we augment the document format of TEXMACS by the language S. Thus, if we denote the document format of TEXMACS by T, we define a **semantic document format** $T + S$ as a document format still accepted by TEXMACS.

Ideally this format of the documents and especially the semantic annotations should not be specific to ΩMEGA in order to enable the combination of the TEXMACS extension with other proof assistance systems as well as the development of independent proof checking tools. However, an abstract language for proofs that is suitable for our purposes and that allows for underspecification is not yet completely fixed. So far we support the assertion-level proof construction rules provided by CORE [Aut03, Aut05]. Thus, instead of defining a fixed language S, we define a language $S(P)$ parametrized over a language P for proofs and define the document format based on $S(C)$, where C denotes the proof language of CORE. This format supports the *static representation* of semantically annotated documents, which can be professionally typeset with TEXMACS.

The TEXMACS document $T + S(C)$ and the pure semantic representation $S(C)$ in the proof assistant must be synchronized. The basic idea here is to synchronize via a diff/patch mechanism tailored to the tree structure of the TEXMACS documents. The differences between two versions ts_i and ts_{i+1} of the document in $T + S(C)$ are compiled into a patch description p of the corresponding document s_i in $S(C)$ for ts_i, such that the application of p to s_i results in s_{i+1} which corresponds to ts_{i+1}. An analogous diff/patch technique is used to propagate changes performed by the proof assistant tool to documents in $S(C)$ towards the TEXMACS document in $T + S(C)$. In order to enable the translation of the patch descriptions, a key-based protocol is used to identify the corresponding parts in $T + S(C)$ and $S(C)$.

Beyond this basic synchronization mechanism, we define a language that allows for the description of specific interactions between TEXMACS and the proof assistant. This language M is a language for structured menus and actions with an evaluation semantics which allows to flexibly compute the necessary parameters for the commands and directives employed in interaction with the proof assistants. The TEXMACS document format $T + S(C)$ is finally extended to $T + S(C) + M$, where the menus can be attached to arbitrary parts of a document and the changes of the documents performed either by

the author or by the proof assistants are propagated between $T + S(C) + M$ and $S(C) + M$ via the diff/patch mechanism. Note that this includes also the adaptation of the menus, which is a necessary prerequisite to support context-sensitive menus and actions contained therein.

The goal of the proposed integration is to use ΩMEGA as a context-sensitive reasoning and verification service accessible from within the first-class mathematical text editor TEXMACS, where the proof assistant adapts to the style an author would like to write his mathematical publication, and to hide any irrelevant system peculiarities from the user. The communication between TEXMACS and ΩMEGA is realized by an OMDOC-based interface language.

Although so far the proofs are tailored to the rules of the CORE system, the representation language in principle is parametrized over a specific language for proofs. We currently replace the CORE specific proof languages by some generic notion of proofs, in order to obtain a generic format for formalized mathematical documents. Thereby we started from a language for assertion-level proofs with underspecification [ABF+03, AF05], which we developed from previous experiences with tutorial dialogs about mathematical proofs between a computer and students [PSBKK04].

26.20 OMDoc as a Data Format for √eriFun

Project Home	http://www.verifun.de/
Authors	Normen Müller Computer Science, International University Bremen

√eriFun (Verification of Functional programs) is a semi-automated system for the verification of programs written in a simple functional programming language $\mathcal{FP}$. The system has been developed since 1998 at the university of Darmstadt for use in education and research. The main design goals are a clearly structured, didactically suited system interface (Figure 26.20), an easily portable implementation (JAVA) and an easily but also powerful proof calculus [WS02]. The system's object language consists of a simple definition principle for free data structures, called *sorts* (see Chapter 16), a recursive definition principle for *functions*, and finally a definition principle for statements, called *lemmas*, about the data structures and the functions. To prove a statement √eriFun supports the user with a couple of inference rules aggregated in *tactics*. A collection of *sorts*, *functions*, *lemmas*, and *proofs* is called a √eriFun *program*. Common file commands, which are based on the JAVA binary serialization mechanism, are provided to save and reload intermediate work.

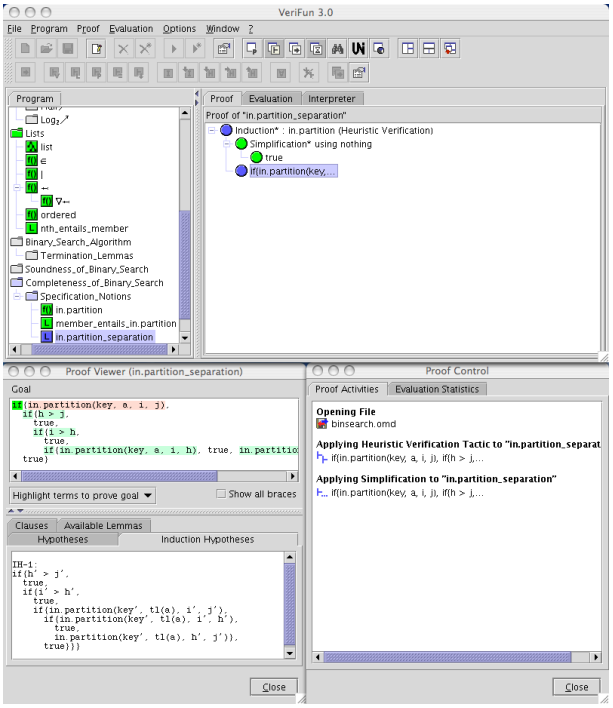


Fig. 26.26. A √eriFun session

The OMDoc interface for √eriFun described here (see [Mül05] for details) was introduced to alleviate the following drawbacks of the former I/O mechanism based on JAVA binary serialization:

- Files are only machine-readable. Thus, e.g. if the files became corrupted by any circumstance, there is no change of a manual repair.

- Files are strongly bound to the version of the system. Thus any internal system modifications make the files unreadable.
- Files are not interchangeable with other theorem provers or other mathematical software systems. Thus the information inside the files are only accessible by ✓`eriFun`.

✓`eriFun`'s interface to OMDoc can be divided into two parts: Encoding and decoding of ✓`eriFun` programs to and from OMDoc respectively.

Encoding

In a typical session with the system, a user defines a *program* by stipulating the *sorts* and the *functions* of the program, defines *lemmas* about the sorts and the functions of the program, and finally verifies these lemmas and the termination of the functions.

In general a *program* is mapped to two OMDoc files: The first one consists of the user-defined elements[21] and in the second one ✓`eriFun`'s logic comprising the predefined symbols, the type system and the proof tactics is defined. At each case one ✓`eriFun`-generated-OMDoc file is composed of one `theory` element. The name of a user-defined theory can be set by the user, whereas the name of the theory ✓`eriFun` is based on is fixed to VAFP.

Functions are declared by `symbol` elements that also introduces the type of the function (Subsection 15.2.3). The body of a function is encoded as an OpenMath object inside a `definition` element. The corresponding `symbol` element is referenced by the `definition` element in the `for` and relating termination assertions in the `existence` attribute.

Note that instead of using `name` attributes, which only allow XML simple names, we generate a unique ID. The actual ✓`eriFun` names are represented in `presentation` elements or rather their `use` elements (Listing 26.4). By using this technique we can use any character string[22] for element names. To cover the whole set of ✓`eriFun` fixities (`prefix` (the default), `infix`, `postfix`, `infixl`, `infixr`, and `outfix`) we had to extend the OMDoc format by `infixl` and `infixr`. However, it was not necessary to also add the `outfix` value, but encoding of `outfix` functions is treated slightly different: The name of the function is encoded in the `lbrack` and `rbrack` attribute respectively of the relating `presentation` element and the `use` element is left empty[23].

Lemmata are mapped to `assertion` elements, the value "`lemma`" being assigned to the `type` attribute. The formula of a lemma, analogous to function bodies, is encoded as an OpenMath object inside an `assertion` element.

---

[21] Actually there are also automatically system-generated elements included, but we may neglect those at this point.

[22] ✓`eriFun` has full Unicode [Inc03] support

[23] As a consequence the previous mentioned special encoding feature does not hold for outfix functions

Particularly convenient is the direct mapping of ✓eriFun proofs to the OMDoc presentation of proofs. Verifications of lemmas and termination analysis of functions are represented in `proof` elements. The assertion to be proven is referenced in the `for` attribute. VAFP-tactics used inside a proof to achieve the various proof steps (encoded in `derive` elements) are denoted by `method` elements. Parameters heuristically computed by the system or manually annotated by the user are encoded as OpenMath objects and appended to each proof step. Furthermore each proof step in ✓eriFun is annotated with a sequence of the form $h_1, \ldots h_n, \forall \ldots ih_1, \ldots, \forall \ldots ih_l \vdash goal$ whereas the expressions h_i are the hypotheses, the expressions $\forall \ldots ih_k$ are the induction hypotheses, and the expression *goal* is the goal-term of the sequence. Such a sequent is represented by `assumption` and `conclusion` child-elements respectively of the relating `derive` element.

Listing 26.2. A polymorphic ✓eriFun sort

```
structure list [@value] <=
  ∅,
  [ infixr ,100] :: (hd : @value, tl : list [@value])
```

Sorts are wrapped inside `adt` elements. At this point this integration process provoked two further adaptations of the OMDoc standard. On the one hand, in contrast to OMDoc, sorts in ✓eriFun could be polymorphic (Listing 26.2). This led to the additional, optional `parameters` attribute of an `adt` element (Listing 26.3). Within this new attribute one can declare by a comma separated list the names of type variables of the abstract data type.

Listing 26.3. A polymorphic OMDoc ADT

```
<adt xml:id="vf7b9f3e59-e78e-4221-8064-7fa0c5689f5d.adt" parameters="value">
  <sortdef name="vf7b9f3e59-e78e-4221-8064-7fa0c5689f5d" type="free">
    <constructor name="vf8a6673ac-c1d9-4698-b6ee-90213539a984"/>
    <constructor name="vf38164505-4983-417f-8bdc-6a42b046e933">
      <argument>
        <type system="simpletypes">
          <OMOBJ xmlns="http://www.openmath.org/OpenMath">
            <OMV name="value"/>
          </OMOBJ>
        </type>
        <selector name="vf9fc4c672-207f-45c0-ae61-1f675fde7aed" total="yes"/>
      </argument>
      <argument>
        <type system="simpletypes">
          <OMOBJ xmlns="http://www.openmath.org/OpenMath">
            <OMA>
              <OMS cd="VeriFun" name="vf7b9f3e59-e78e-4221-8064-7fa0c5689f5d"/>
              <OMV name="value"/>
            </OMA>
          </OMOBJ>
        </type>
        <selector name="vf55767f3a-b019-4308-88f9-d68ee0db595e" total="yes"/>
      </argument>
    </constructor>
  </sortdef>
</adt>
```

On the other hand, the child elements of a `constructor` element had to be expanded by an additional `type` element to specify the type of the formal parameter of the parent `constructor` element. Listing 26.4 illustrates the corresponding `presentation` elements of the ADT in Listing 26.3.

Listing 26.4. Representation of √`veriFun` names to OMDoc

```
<presentation for="#vf7b9f3e59-e78e-4221-8064-7fa0c5689f5d" role="applied">
  <use format="VeriFun">list</use>
</presentation>
<presentation for="#vf8a6673ac-c1d9-4698-b6ee-90213539a984" role="applied"
              bracket-style="math" precedence="1" fixity="prefix" lbrack="(" rbrack=")">
  <use format="VeriFun">∅</use>
</presentation>
<presentation for="#vf38164505-4983-417f-8bdc-6a42b046e933" role="applied"
              bracket-style="math" precedence="100" fixity="infixr" lbrack="(" rbrack=")">
  <use format="VeriFun">::</use>
</presentation>
<presentation for="#vf9fc4c672-207f-45c0-ae61-1f675fde7aed" role="applied"
              bracket-style="math" precedence="1" fixity="prefix" lbrack="(" rbrack=")">
  <use format="VeriFun">hd</use>
</presentation>
<presentation for="#vf55767f3a-b019-4308-88f9-d68ee0db595e" role="applied"
              bracket-style="math" precedence="1" fixity="prefix" lbrack="(" rbrack=")">
  <use format="VeriFun">tl</use>
</presentation>
```

Decoding

The decoding of a √`veriFun` program represented in OMDoc is reverse to the encoding mechanism. First we create an empty program and then start the sequential decoding of each `adt`, `symbol` and its relating `definition`, and `assertion` element back into the $\mathcal{FP}$ syntax. After a successful reconstruction of an element it is appended to the current program. Right after such an insertion we check for a `proof` element containing a reference to this new program element. If a proof exists, we re-play all the proof steps and associate the recreated √`veriFun` proof to the corresponding program element.

One aspect of this decoding exercise is worth mentioning here. The √`veriFun` system also benefited by the development of the OMDoc standard: Revelation of bugs deep in the system! Especially $\mathcal{FP}$ parser errors and inconsistencies in proof tactics applications could be discovered. Maybe those errors would never have been detected, because in most cases the user is not able to produce them manually, but this errors are automatically generated by the system. So with the assistance of the strict encoding and decoding to and from OMDoc respectively we were able to achieve a much more robust verification system.

By the integration of the open content Markup language OMDoc into the semi-automated theorem prover √`veriFun`, we made the system more reliable and facilitate the participation in the mathematical network to serve as yet another service. Functional programs and especially proof of statements created in √`veriFun` are now open to the public. The data is human-readable, machine-understandable, no longer subjected to a particular version of the

system. Thus, ✓`eriFun` generated knowledge became accessible, robust, interchangeable and transparent.

Part V

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 1.2 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 1.2 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 1.1 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 [Bos98], 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`.

BErr(87)

element	state	comments	cf.
`alternative`	aug	This element can now have `theory`, `generated-from`, and `generated-via` attributes.	154
`argument`	cha	The `sort` has been replaced by a `type` child, so that higher-order sorts can be specified.	167
`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.	150

[87] Erratum: the old `extradata` content has nothing to do with `dc:subject`

`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.	154
`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.	165
`answer`	cha	the `answer` element does not allow `symbol` children any more, if these are needed, the exercise should have its own theory.	224
`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.	203
`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.	146
`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.	191
`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	224

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.	216
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.	132
constructor	cha	The role attribute is now fixed to object. The commonname child has been replaced by an initial metadata element.	166
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.	217
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.	104
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.	195
dc:description	aug	The dc:description can now have the optional xml:id, and CSS attributes	105
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.	148
effect	aug	allows an optional xml:id attribute	218

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.	155
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.	223
extradata	cha	The content of the old extradata element can now be directly in the metadata element as additional elements.	
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.	203
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.	223
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.	173
inclusion	aug	allows optional attributes xml:id, conservativity, and conservativity-just for stating and justifying conservativity.	190
imports	lib	the xml:id is now optional. New optional attributes conservativity and conservativity-just for stating and justifying conservativity.	159
input	aug	allows an optional xml:id attribute	218
legacy	new	An element for encapsulating legacy mathematics, can be used wherever m:math and om:OMOBJ are allowed.	127
loc	dep	The catalogue mechanism has been eliminated.	

`m:math`	new	Content-MathML is now allowed wherever OpenMath objects were allowed before.	121
`map`	new	this element allows to map its style directives over a list of e.g. arguments	203
`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.	224
`measure`	aug	allows an optional `xml:id` attribute	149
`metacomment`	dep	This element is superseded by the `omtext` element.	133
`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`.	98
`obligation`	aug	allows an optional `xml:id` attribute	189
`omdoc`	aug	This element can now have a `theory`, `generated-from`, and `generated-via` attributes.	96
`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 14.6 for details. Furthermore, the element can now have the attributes, `modules`, `theory`, `generated-from`, and `generated-via`.	100
`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.	219
`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.	200

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.	114
om:OMFOREIGN	new	The om:OMFOREIGN element can be used to encapsulate arbitrary XML data in OpenMath attributions.	117
om:OMR	new	In the OpenMath2 standard, this element is the main vehicle of the structure sharing representation.	118
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.	133
ordering	aug	Now allows the optional xml:id and terminating attributes. The latter points to a termination assertion.	149
output	aug	allows an optional xml:id attribute	218
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	195
phrase	new	used to mark up phrases in CMPs and supply them with identifiers and links to context that can be used for presentation and referencing.	134
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.	205

`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.	216
`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.	171
`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.	180
`recognizer`	cha	The `role` attribute was fixed to `object`. The `commonname` child has been replaced by an initial `metadata` element.	167
`ref`	aug	`ref` now has an optional `xml:id` attribute that identifies it.	100
`selector`	cha	The `role` attribute was fixed to `object`. The `commonname` child has been replaced by an initial `metadata` element.	167
`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.	223
`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.	166
`dc:subject`	aug	The `dc:subject` can now have the optional `dc:id`, and CSS attributes	105
`style`	aug	The `style` element now allows a `map` element in the body	201

`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.	144
`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.	136
`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.	158
`dc:title`	aug	The `dc:title` can now have the optional `dc:id`, and CSS attributes.	104
`tgroup`	new	The `tgroup` can be used to structure theories like documents.	158

`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.	147
`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.	189
`theory`	aug	the `theory` element can now be nested.	158
`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.	206
`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.	134
`xslt`	cha	the content of this element need not be escaped any more, it is now a valid XSLT fragment.	201

EErr(87)

A.2 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	203
`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`.	154
`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.	167
`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`.	150
`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`.	146
`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	191
`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.	130
`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`).	216
`constructor`	aug	new optional child `recognizer` for a recognizer predicate	166

`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.	217
`dc:date`	aug	new optional `who` attribute that can be used to specify who did the `action` on this date.	105
`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.	104
`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`.	195
`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`.	148
`element`	new	presentation of XML elements	203
`FMP`	aug	now allows multiple `conclusion` elements, to represent general Gentzen-type sequents (not only natural deduction.) `FMP` now allows an optional `id` attribute.	131
`hypothesis`	cha	new required attribute `discharged-in` to specify the `derive` element that discharges this hypothesis.	173
`measure`	new	specifies a measure function (as an OMOBJ)	149
`metadata`	aug	new optional attribute `inherits` that allows to inherit metadata from other declarations	98
`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.	174
`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`.	100
`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.	219
`omstyle`	new	for specifying the style of OMDoc elements	200
`omtext`	cha	the `from` is deprecated, we only leave the `for` attribute, to specify the referential character of the `type`.	133
`ordering`	new	specifies a well-founded ordering (as an OMOBJ)	149
`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	205

`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 216
`proof`	cha	attribute `theory` is now optional, since the element can appear inside a `theory` element.	171
`proofobject`	cha	attribute `theory` is now optional, since the element can appear inside a `theory` element.	171
`recognizer`	new	specifies the recognizer predicate of a sort.	167
`recurse`	new	recursive calls to presentation in `style`.	203
`ref`	cha	attribute `kind` renamed to `type`.	100
`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`).	167
`signature`	dep	for the moment	
`sortdef`	cha	has a mandatory `name` attribute, otherwise the defined symbol has no name.	166
`style`	new	allows to specify style information in `presentation` and `omstyle` elements using a simplified OMDoc-internalized version of XSLT.	201
`symbol`	aug	new optional attribute `generated-by` to show that it is generated by expanding a `definition`.	144
`text`	new	presentation of text in `omstyle`.	203

`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.	189
`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.	147
`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`.	206
`value-of`	new	presentation of values in `style`.	203
`with`	new	used to supply fragments of text in `CMP`s with `style` and `id` attributes that can be used for presentation and referencing.	134
`xslt`	new	allows to embed XSLT into `presentation` and `omstyle` elements.	201

B

Quick-Reference Table to the OMDoc Elements

Element	p.	Mod.	Required Attribs	Optional Attribs	D C	Content
adt	165	ADT		xml:id, type, style, class, theory, generated-from, generated-via	+	sortdef+
alternative	154	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	224	QUIZ	verdict	xml:id, style, class	+	CMP*, FMP*
m:apply	122	MML		id, xlink:href	−	bvar?, ⟪*CMel*⟫*
argument	167	ADT	sort		+	selector?
assertion	150	ST		xml:id, type, theory, generated-from, generated-via, style, class	+	CMP*, FMP*
assumption	132	MTXT		xml:id, inductive, style, class	+	CMP*, (OMOBJ |m:math |legacy)?
attribute	203	PRES	name		−	(value-of| text)*
axiom	146	ST	name	xml:id, type, generated-from, generated-via, style, class	+	CMP*, FMP*
axiom-inclusion	191	CTH	from, to	xml:id, style, class, theory, generated-from, generated-via	+	morphism?, (path-just| obligation*)
m:bvar	122	MML		id, xlink:href	−	ci*

m:ci	121	MML		id, xlink:href	–	PCDATA
m:cn	121	MML		id, xlink:href	–	([0-9]|,|.) (*|e([0-9]|,|.)*)?
choice	224	QUIZ		xml:id, style, class	+	CMP*, FMP*
CMP	130	MTXT		xml:lang, xml:id	–	(text| OMOBJ |m:math |legacy | with | term | omlet)*
code	216	EXT		xml:id, for, theory, generated-from, generated-via, requires, style, class	+	input?, output?, effect?, data+
conclusion	132	MTXT		xml:id, style, class	+	CMP*, (OMOBJ |m:math |legacy)?
constructor	166	ADT	name	type, scope, style, class, theory, generated-from, generated-via	+	argument*, recognizer?
dc:contributor	104	DC		xml:id, role, style, class	–	⟪*text*⟫
dc:creator	104	DC		xml:id, role, style, class	–	⟪*text*⟫
m:csymbol	121	MML	definitionURL	id, xlink:href	–	EMPTY
data	217	EXT		format, href, size, original	–	<![CDATA[...]]>
dc:date	105	DC		action, who	–	ISO 8601 norm
dd	138	RT		xml:id, style, class, index, verbalizes	+	CMPcontent
di	138	RT		xml:id, style, class, index, verbalizes	+	dt+,dd*
dl	138	RT		xml:id, style, class, index, verbalizes	+	li*
dt	138	RT		xml:id, style, class, index, verbalizes	+	CMPcontent
decomposition	195	DG	links	theory, generated-from, generated-via	–	EMPTY
definition	148	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	105	DC		xml:lang	–	CMPcontent
derive	172	PF		xml:id, style, class	–	CMP*, FMP?, method?

effect	218	EXT		xml:id, style, class	–	CMP*,FMP*
element	203	PRES	name	xml:id, cr, ns	–	(attribute| element| text| recurse)*
example	155	ST	for	xml:id, type, assertion, proof, style, class, theory, generated-from, generated-via	+	CMP*| (OMOBJ |m:math |legacy)?
exercise	223	QUIZ		xml:id, type, for, from, style, class, theory, generated-from, generated-via	+	CMP*, FMP*, hint?, (solution*|mc*)
FMP	131	MTXT		logic, xml:id	–	(assumption*, conclusion*)|OMOBJ |m:math |legacy
dc:format	106	DC			–	fixed: "application/omdoc+xml"
hint	223	QUIZ		xml:id, style, class, theory, generated-from, generated-via	+	CMP*, FMP*
hypothesis	173	PF		xml:id, style, class, inductive	–	CMP*, FMP*
dc:identifier	106	DC		scheme	–	ANY
ide	139	RT	index	xml:id,sort-by,see seealso, links, style, class	–	idp*
idp	139	RT		xml:id,sort-by,see seealso, links, style, class	–	CMPcontent
idt	139	RT		style, class	–	CMPcontent
idx	139	RT		xml:id,sort-by,see seealso, links, style, class	–	idt?,idp+
ignore	99	DOC		type, comment	–	ANY
imports	159	CTH	from	xml:id, type, style, class	+	morphism?
inclusion	190	CTH	for	xml:id	–	
input	218	EXT		xml:id, style, class	–	CMP*,FMP*
insort	166	ADT	for		–	
dc:language	106	DC			–	ISO 8601 norm
li	138	RT		xml:id, style, class, index, verbalizes	–	Math Vernacular
cc:license	108	CC		jurisdiction	–	permissions, prohibitions, requirements

link	138	RT		xml:id, style, class, index, verbalizes	–	Math Vernacular
m:math	121	MML		id, xlink:href	–	⟪*CMel*⟫+
mc	224	QUIZ		xml:id, style, class, theory, generated-from, generated-via	–	choice, hint?, answer
measure	149	ST		xml:id	–	OMOBJ |m:math |legacy
metadata	98	DC		inherits	–	(dc-element)*
method	174	PF	xref		–	(OMOBJ |m:math |legacy| premise | proof | proofobject)*
morphism	185	CTH		xml:id, base, consistency, exhaustivity, type, hiding, style, class	–	requation*, measure?, ordering?
note	139	RT		type,xml:id, style, class, index, verbalizes	–	Math Vernacular
obligation	189	CTH	induced-by, assertion	xml:id	–	EMPTY
om:OMA	114	OM		id, cdbase	–	⟪*OMel*⟫*
om:OMATTR	116	OM		id, cdbase	–	⟪*OMel*⟫
om:OMATP	116	OM		cdbase	–	(OMS, (⟪*OMel*⟫ | om:OMFOREIGN))+
om:OMB	117	OM		id, class, style, class	–	#PCDATA
om:OMBIND	115	OM		id, cdbase	–	⟪*OMel*⟫, om:OMBVAR, ⟪*OMel*⟫?
om:OMBVAR	116	OM			–	(om:OMV | om:OMATTR)+
om:OMFOREIGN	117	OM		id, cdbase	–	ANY
omdoc	96	DOC		xml:id,type, version, style, class, xmlns, theory, generated-from, generated-via	+	(top-level element)*
om:OME	117	OM		xml:id	–	(⟪*OMel*⟫)?
om:OMR	118	OM	href		–	
om:OMF	117	OM		id, dec, hex	–	#PCDATA
omgroup	100	DOC		xml:id, type, style, class, modules, theory, generated-from, generated-via	+	top-level element*
ol	138	RT		xml:id, style, class, index, verbalizes	–	li*
om:OMI	117	OM		id, class, style	–	[0-9]*

omlet	219	EXT		id, argstr, type, function, action, data, style, class	+	ANY
om:OMOBJ	114	OM		id, cdbase, class, style	–	⟪*OMel*⟫?
omstyle	200	PRES	element	for, xml:id, xref, style, class	–	(style|xslt)*
om:OMS	114	OM	cd, name	class, style	–	EMPTY
omtext	133	MTXT		xml:id, type, for, from, style, theory, generated-from, generated-via	+	CMP+, FMP?
om:OMV	114	OM	name	class, style	–	EMPTY
ordering	149	ST		xml:id	–	OMOBJ |m:math |legacy
output	218	EXT		xml:id, style, class	–	CMP*,FMP*
p	138	RT		xml:id, style, class, index, verbalizes	–	Math Vernacular
param	221	EXT	name	value, valuetype	–	EMPTY
path-just	195	DG	local, globals	for, xml:id	–	EMPTY
cc:permissions	109	CC		reproduction, distribution, derivative_works	–	EMPTY
premise	174	PF	xref		–	EMPTY
presentation	205	PRES	for	xml:id, xref, fixity, role, lbrack, rbrack, separator, bracket-style, style, class, precedence, crossref-symbol	–	(use | xslt | style)*
private	216	EXT		xml:id, for, theory, generated-from, generated-via, requires, reformulates, style, class	+	data+
cc:prohibitions	109	CC		commercial_use	–	EMPTY
proof	171	PF		xml:id, for,theory, generated-from, generated-via, style, class	+	(symbol | definition | omtext | derive | hypothesis)*
proofobject	180	PF		xml:id, for, theory, generated-from, generated-via, style, class	+	CMP*, (OMOBJ |m:math |legacy)
dc:publisher	105	DC		xml:id, style, class	–	ANY
ref	100	DOC		xref, type	–	ANY

recognizer	167	ADT	name	type, scope, role, style, class	+	
recurse	203	PRES		select	–	EMPTY
dc:relation	106	DC			–	ANY
requation	149	ST		xml:id, style, class	–	(OMOBJ |m:math |legacy),(OMOBJ |m:math |legacy)
cc:requirements	109	CC		notice, copyleft, attribution	–	EMPTY
dc:rights	106	DC			–	ANY
selector	167	ADT	name	type, scope, role, total, style, class	+	
solution	223	QUIZ		xml:id, for, style, class, theory, generated-from, generated-via	+	(CMP*, FMP*) | proof
sortdef	166	ADT	name	role, scope, style, class	+	(constructor|insort)*
dc:source	106	DC			–	ANY
style	201	PRES	format	xml:lang, requires	–	(element | text | recurse | value-of)*
dc:subject	105	DC		xml:lang	–	CMPcontent
symbol	144	ST	name	role, scope, style, class,generated-from,generated-via	+	type*
table	138	RT		xml:id, style, class, index, verbalizes	–	tr*
term	136	MTXT	cd, name	xml:id, role, style, class	–	CMP content
text	203	PRES			–	#PCDATA
td	138	RT		xml:id, style, class, index, verbalizes	–	Math Vernacular
th	138	RT		xml:id, style, class, index, verbalizes	–	Math Vernacular
theory	158	ST	xml:id	cdbase, style, class	+	(statement|theory)*
theory-inclusion	189	CTH	from, to	xml:id, style, class, theory, generated-from, generated-via	+	(morphism, decomposition?)
tr	138	RT		xml:id, style, class, index, verbalizes	–	(td|th)*
dc:title	104	DC		xml:lang	–	CMPcontent
tgroup	158	DOC		xml:id, type, style, class, modules, generated-from, generated-via	+	top-level or theory-constitutive element*
type	147	ST	system	xml:id, for, style, class	–	CMP*, (OMOBJ |m:math |legacy)

dc:type	106	DC			–	fixed: "Dataset" or "Text" or "Collection"
ul	138	RT		xml:id, style, class, index, verbalizes	–	li*
use	206	PRES	format	xml:lang, requires, fixity, lbrack, rbrack, separator, crossref-symbol, element, attributes	–	(use | xslt | style)*
value-of	203	PRES	select		–	EMPTY
phrase	134	MTXT		xml:id, style, class, index, verbalizes, type	–	CMP content
xslt	201	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-inclusions or theory-inclusions 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.	
inherits	metadata	URI reference
	points to a metadata element from which this one inherits.	
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	
type	ref	include, cite
	specifies whether to replace the ref element by the fragment referenced by href attribute or to merely cite it.	
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.3 to D.15), a definition of the most common attributes that occur in several of the modules (see Appendix D.2), and the sub-language driver files which we will introduce next.

D.1 The Sub-Language Drivers

The driver files set up the grammars for the OMDoc sub-languages (see Section 22.3 for a discussion) in layers. The RelaxNG grammar for "Basic OMDoc" sets up the language and loads the relevant modules.

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.2: OMDoc Basic)
# $Id: omdoc-basic.rnc 8009 2008-09-07 19:02:21Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/omdoc-basic.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2007 Michael Kohlhase, released under the GNU Public License (GPL)

start = omdoc

include "omdocattribs.rnc"
include "omdocmobj.rnc"
include "omdocdoc.rnc"
include "omdocdc.rnc"
include "omdoccc.rnc"
include "omdocmtxt.rnc"
include "omdocrt.rnc"
```

The RelaxNG grammar for "Content Dictionary OMDoc" adds modules PRES and ST.

```
# A RelaxNG for Open Mathematical documents (OMDoc 1.2: OMDoc Content Dictionaries)
# $Id: omdoc-cd.rnc 8009 2008-09-07 19:02:21Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/omdoc-cd.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2007 Michael Kohlhase, released under the GNU Public License (GPL)

include "omdoc-basic.rnc"
include "omdocpres.rnc"
include "omdocst.rnc"
```

The RelaxNG grammar for "Educational OMDoc" adds modules PF and QUIZ to that:

```
# A RelaxNG for Open Mathematical documents (OMDoc 1.2: OMDoc Education)
# $Id: omdoc-education.rnc 8009 2008-09-07 19:02:21Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/omdoc-education.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2007 Michael Kohlhase, released under the GNU Public License (GPL)

include "omdoc-mathweb.rnc"
include "omdocquiz.rnc"
```

The RelaxNG grammar for "Educational OMDoc" starts with "Content Dictionary OMDoc" adds modules PF and EXT:

```
# A RelaxNG for Open Mathematical documents (OMDoc 1.2: OMDoc MathWeb)
# $Id: omdoc-mathweb.rnc 8009 2008-09-07 19:02:21Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/omdoc-mathweb.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2007 Michael Kohlhase, released under the GNU Public License (GPL)

include "omdoc-cd.rnc"
include "omdocext.rnc"
include "omdocpf.rnc"
```

The RelaxNG grammar for "Educational OMDoc" starts with "Content Dictionary OMDoc" adds modules PF and EXT:

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.2: OMDoc Specification)
# $Id: omdoc-spec.rnc 8009 2008-09-07 19:02:21Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/omdoc-spec.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2007 Michael Kohlhase, released under the GNU Public License (GPL)

# default namespace omdoc = "http://www.mathweb.org/omdoc"

include "omdoc-cd.rnc"
include "omdoccth.rnc"
include "omdocdg.rnc"
include "omdocpf.rnc"
include "omdocadt.rnc"
```

Finally, the The RelaxNG grammar for full OMDoc only needs to add modules EXT and QUIZ:

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.2)
# $Id: omdoc.rnc 8009 2008-09-07 19:02:21Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/omdoc.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2007 Michael Kohlhase, released under the GNU Public License (GPL)

include "omdoc-spec.rnc"
```

```
include "omdocext.rnc"
include "omdocquiz.rnc"
```

D.2 Common Attributes

The RelaxNG grammar for OMDoc separates out declarations for commonly used attributes.

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.2) Common attributes
# $Id: omdocattribs.rnc 8009 2008-09-07 19:02:21Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/omdocattribs.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2007 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://www.mathweb.org/omdoc"
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}
idrest.attribs = css.attribs, nonlocal-attribs*, attribute xml:base {xsd:anyURI}?
id.attrib = attribute xml:id {xsd:ID}?, idrest.attribs

omdoc.toplevel.attribs = id.attrib, attribute generated-from {omdocref}?

# The current XML-recommendation doesn't yet support the
# three-letter short names for languages (ISO 693-2). So
# the following section will be using the two-letter
# (ISO 693-1) encoding for the languages.
#
#       en : English,    de : German,    fr : French,
#       la : Latin,      it : Italian,   nl : Dutch,
#       ru : Russian,    pl : Polish,    es : Spanish,
#       tr : Turkish,    zh : Chinese,   ja : Japanese,
#       ko : Korean      ...

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}?
```

D.3 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 1.2) Module MOBJ
# $Id: omdocmobj.rnc 8009 2008-09-07 19:02:21Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/omdocmobj.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2007 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://www.mathweb.org/omdoc"

namespace om = "http://www.openmath.org/OpenMath"

# we include the OpenMath 2 schema, but we also allow CSS attributes, etc.
include "openmath2.rnc" {common.attributes = attribute id {xsd:ID}?,idrest.attribs}

# we include the MathML2 schema
include "mathml2/mathml2.rnc"

# the legacy element, it can encapsulate the non-migrated formats
legacy = (ss| element legacy {id.attrib,
                        attribute formalism {xsd:anyURI}?,
                        attribute format {xsd:anyURI},
                        Anything}) # to allow everything

omdocmobj.class = legacy | OMOBJ | math
```

D.4 Module MTXT: Mathematical Text

BErr(88)
BErr(89)
BErr(90)

The RNC module MTXT provides infrastructure for mathematical vernacular (see Chapter 14 for a discussion).

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.2) Module MTXT
# $Id: omdocmtxt.rnc 8672 2010-08-22 13:12:47Z clange $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/omdocmtxt.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2007 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://www.mathweb.org/omdoc"

omdoc.class |= omtext

#attribute for is a whitespace-separated list of URIrefs
for.attrib = attribute for {omdocrefs}
fori.attrib = attribute for {omdocrefs}?
from.attrib = attribute from {omdocref}
```

[88] Erratum: The `type` attributes on `phrase` and `omtext` were not conforming to the spec

[89] Erratum: attribute value `trasition` forgotten from rnc

[90] Erratum: the `verbalizes` attribute had been forgotten for the `phrase` element

```
verbalizes.attrib = attribute verbalizes {omdocrefs}
parallel.attribs = verbalizes.attrib?, attribute index {xsd:NMTOKEN}?
omdocmtxt.MC.content = metadata?,CMP*
omdocmtxt.MCF.content = omdocmtxt.MC.content,FMP*

# what can go into a mathematical text (to be extended in other modules)
omdoc.mtext.class = text | phrase | term | omdocmobj.class

rsttype = "abstract" | "introduction" | "annote" | "transition" |
          "conclusion" | "thesis" | "comment" | "antithesis" |
          "elaboration" | "motivation" | "evidence" | "note" | "notation"

statementtype = "axiom" | "definition" | "example" | "proof" |
                "derive" | "hypothesis"

assertiontype = "theorem" | "lemma" | "corollary" | "proposition" |
                "conjecture" | "false-conjecture" | "obligation" |
                "postulate" | "formula" | "assumption" | "rule"
omtext.type.attrib = attribute type {rsttype | statementtype | assertiontype | xsd:anyURI}
omtext = element omtext {omdoc.toplevel.attribs,
                         omtext.type.attrib? &
                         attribute for {omdocref}?,
                         attribute from {omdocref}?,
                         verbalizes.attrib?,
                         metadata?,CMP+,FMP*}
# attribute 'for' is a URIref, to omdocdoc.class's it is needed by the 'type' attribute

CMP = (ss| element CMP {xml.lang.attrib, id.attrib, (omdoc.mtext.class)*})

phrase = (ss| element phrase {id.attrib, parallel.attribs,
                              omtext.type.attrib? &
                              (omdoc.mtext.class)*})
# identifies a text passage and allows to attatch style and type information to it

term = (ss| element term {id.attrib,
                          attribute role {text}?,
                          attribute cdbase {xsd:anyURI}?,
                          attribute cd {xsd:NCName},
                          attribute name {xsd:NCName},
                          (omdoc.mtext.class)*})

FMP = (ss| element FMP {id.attrib, attribute logic {xsd:NMTOKEN}?,
                        ((assumption*,conclusion*)|omdocmobj.class)})

# If FMP contains a omdocmobj.class then this is the assertion,
# if it contains (assumption*,conclusion*), then it is a
# logical sequent (A1,..., An |- C1,...,Cm):
# all the Ai entail one of the Ci

assumption = (ss| element assumption {id.attrib,
                                      attribute inductive {"yes" | "no"}?,
                                      (omdocmobj.class)})
conclusion = (ss| element conclusion {id.attrib, (omdocmobj.class?)})
```

EErr(90)
EErr(89)
EErr(88)

D.5 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 1.2) Module DOC
# $Id: omdocdoc.rnc 8373 2009-06-07 04:19:16Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/omdocdoc.rnc $
```

```
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2007 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://www.mathweb.org/omdoc"
# extend the stuff that can go into a mathematical text
omdoc.mtext.class |= ignore | ref

ss = ignore | ref
omdoc.class |= ss
omdoc.meta.class |= notAllowed

metadata = element metadata {id.attrib,
                             attribute inherits {omdocref}?,
                             (omdoc.meta.class)*}

Anything = (AnyElement|text)*
AnyElement = element * {AnyAttribute,(text|AnyElement)*}
AnyAttribute = attribute * { text }*

# this element can be used in lieu of a comment, it is read
# by the style sheet, (comments are not) and can therefore
# be transformed by them

ignore = element ignore {attribute type {xsd:string}?,
                         attribute comment {xsd:string}?,
                         Anything}

ref = element ref {id.attrib,
                   xref.attrib,
                   attribute type {xsd:string}?}

# the types supported (there may be more over time) are
# - 'include' (the default) for in-text replacement
# - 'cite' for a reference with a generated label

group.attribs = attribute type {xsd:anyURI}?, attribute modules {xsd:anyURI}?

group.elts = metadata?,(omdoc.class | omgroup)*

# grouping defines the structure of a document
omgroup = element omgroup {group.attribs,omdoc.toplevel.attribs,group.elts}

# finally the definition of the OMDoc root element
omdoc = element omdoc {omdoc.toplevel.attribs,group.attribs,
                       attribute version {xsd:string {pattern = "1.2"}}?,
                       group.elts}
```

D.6 Module DC: Dublin Core Metadata

The RNC module DC includes an extension of the Dublin Core vocabulary for bibliographic metadata, see Sections 12.1 and 12.2 for a discussion.

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.2) Module DC
# $Id: omdocdc.rnc 8009 2008-09-07 19:02:21Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/omdocdc.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2007 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace dc = "http://purl.org/dc/elements/1.1/"

# Persons in Dublin Core Metadata
omdocdc.person.content = text
```

```
# the rest of Dublin Core content
omdocdc.rest.content = (text | AnyElement)*

omdoc.meta.class |= ss | dc.contributor | dc.creator | dc.rights
                   | dc.subject | dc.title | dc.description | dc.publisher
                   | dc.date | dc.type | dc.format | dc.identifier
                   | dc.source | dc.language | dc.relation

# the MARC relator set; see http://www.loc.gov/marc/relators
dcrole = attribute role {"act" | "adp" | "aft" | "ann" | "ant" | "app" | "aqt" |
          "arc" | "arr" | "art" | "asg" | "asn" | "att" | "auc" | "aud" | "aui" |
          "aus" | "aut" | "bdd" | "bjd" | "bkd" | "bkp" | "bnd" | "bpd" | "bsl" |
          "ccp" | "chr" | "clb" | "cli" | "cll" | "clt" | "cmm" | "cmp" | "cmt" |
          "cnd" | "cns" | "coe" | "col" | "com" | "cos" | "cot" | "cov" | "cpc" |
          "cpe" | "cph" | "cpl" | "cpt" | "cre" | "crp" | "crr" | "csl" | "csp" |
          "cst" | "ctb" | "cte" | "ctg" | "ctr" | "cts" | "ctt" | "cur" | "cwt" |
          "dfd" | "dfe" | "dft" | "dgg" | "dis" | "dln" | "dnc" | "dnr" | "dpc" |
          "dpt" | "drm" | "drt" | "dsr" | "dst" | "dte" | "dto" | "dub" | "edt" |
          "egr" | "elt" | "eng" | "etr" | "exp" | "fac" | "flm" | "fmo" | "fnd" |
          "fpy" | "frg" | "hnr" | "hst" | "ill" | "ilu" | "ins" | "inv" | "itr" |
          "ive" | "ivr" | "lbt" | "lee" | "lel" | "len" | "let" | "lie" | "lil" |
          "lit" | "lsa" | "lse" | "lso" | "ltg" | "lyr" | "mdc" | "mod" | "mon" |
          "mrk" | "mte" | "mus" | "nrt" | "opn" | "org" | "orm" | "oth" | "own" |
          "pat" | "pbd" | "pbl" | "pfr" | "pht" | "plt" | "pop" | "ppm" | "prc" |
          "prd" | "prf" | "prg" | "prm" | "pro" | "prt" | "pta" | "pte" | "ptf" |
          "pth" | "ptt" | "rbr" | "rce" | "rcp" | "red" | "ren" | "res" | "rev" |
          "rpt" | "rpy" | "rse" | "rsp" | "rst" | "rth" | "rtm" | "sad" | "sce" |
          "scl" | "scr" | "sec" | "sgn" | "sng" | "spk" | "spn" | "spy" | "srv" |
          "stl" | "stn" | "str" | "ths" | "trc" | "trl" | "tyd" | "tyg" | "voc" |
          "wam" | "wdc" | "wde" | "wit"}?
dclang = id.attrib, xml.lang.attrib

# first the Dublin Core Metadata model of the
# Dublin Metadata initiative (http://purl.org/dc)

dc.contributor = element contributor {dclang,dcrole,omdocdc.person.content}
dc.creator = element creator {dclang,dcrole,omdocdc.person.content}
dc.title = element title {dclang,(omdoc.mtext.class)*}
dc.subject = element subject {dclang,(omdoc.mtext.class)*}
dc.description = element description {dclang,(omdoc.mtext.class)*}
dc.publisher = element publisher {id.attrib,omdocdc.rest.content}
dc.type = element type {("Dataset" | "Text" | "Collection")}
dc.format = element format {("application/omdoc+xml")}
dc.source = element source {omdocdc.rest.content}
dc.language = element language {omdocdc.rest.content}
dc.relation = element relation {omdocdc.rest.content}
dc.rights = element rights {omdocdc.rest.content}
dc.date = element date {attribute action {xsd:NMTOKEN}?,
                        attribute who {omdocref}?,
                        xsd:dateTime}
dc.identifier = element identifier {attribute scheme {xsd:NMTOKEN},text}
```

D.7 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.

BErr(91)
BErr(92)
BErr(93)

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.2) Module ST
# $Id: omdocst.rnc 8713 2010-09-22 05:49:27Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/omdocst.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2007 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://www.mathweb.org/omdoc"

omdocst.scope.attrib = attribute scope {"global" | "local"}?

omdocst.constitutive.class     = symbol | axiom | definition | imports
omdocst.nonconstitutive.class = assertion | type | alternative | example | theory

theory-unique = xsd:NCName
just-by.attrib = attribute just-by {omdocref}

omdoc.class |= omdocst.nonconstitutive.class

omdocst.constitutive.attribs = id.attrib, attribute generated-from {omdocref}?

sym.role.attrib = attribute role {"type" | "sort" | "object" |
                                  "binder" | "attribution" | "application" | "constant" |
                                  "semantic-attribution" | "error"}

symbol = element symbol {omdocst.scope.attrib,
                         attribute name {theory-unique}?,
                         omdocst.constitutive.attribs,
                         sym.role.attrib?,
                         metadata?,type*}

forname.attrib = attribute for { list {xsd:NCName+}}
axiom = element axiom {omdocst.constitutive.attribs,
                       forname.attrib?,
                       attribute type {xsd:string}?,
                       omdocmtxt.MCF.content}

#informal definitions
def.informal = attribute type {"informal"}?

#simple definitions
def.simple = (attribute type {"simple"},(omdocmobj.class))

#implicit definitions
exists.attrib = attribute existence {omdocref}
unique.attrib = attribute uniqueness {omdocref}
def.implicit = (attribute type {"implicit"}, exists.attrib?, unique.attrib?, FMP*)

#definitions by (recursive) equations
exhaust.attrib = attribute exhaustivity {omdocref}
consist.attrib = attribute consistency {omdocref}
def.pattern = attribute type {"pattern"}? & exhaust.attrib? & consist.attrib? & requation+
def.inductive = attribute type {"inductive"}? & exhaust.attrib? & consist.attrib? &
          requation+ & measure? & ordering?

def.eq = def.pattern | def.inductive
```

[91] Erratum: simple definitions should not have an `existence` attribute, furthermore pattern definitions should not have `MEASURE` and `ORDERING` children

[92] Erratum: the `type` element needs to allow a `for` attribute

[93] Erratum: the `TGROUP` element should not contain `OMGROUP` children

```
#all definition forms, add more by extending this.
defs.all = def.informal | def.simple | def.implicit | def.eq

# 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 = element definition {omdocst.constitutive.attribs,
            forname.attrib,
            omdocmtxt.MC.content,(defs.all)}

requation = (ss| element requation {id.attrib,omdocmobj.class,omdocmobj.class})
measure = (ss| element measure {id.attrib,omdocmobj.class})
ordering = (ss| element ordering {id.attrib,attribute terminating {omdocref}?,omdocmobj.class})

# the non-constitutive statements, they need a theory attribute
omdoc.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"

assertion = element assertion {omdoc.toplevel.attribs,
                               attribute type {assertiontype}?,
                               attribute status {ded.status.class}?,
                               attribute just-by {omdocrefs}?,
                               omdocmtxt.MCF.content}
# 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 = element type {omdoc.toplevel.attribs, just-by.attrib?,
                     attribute system {omdocref}?,
                     attribute for {omdocref}?,
                     omdocmtxt.MC.content,
                     (omdocmobj.class),
                     (omdocmobj.class)?}

alternative = element alternative {omdoc.toplevel.attribs, for.attrib,
                                   omdocmtxt.MC.content,(defs.all),
                                   attribute entailed-by {omdocref},
                                   attribute entails {omdocref},
                                   attribute entailed-by-thm {omdocref},
                                   attribute entails-thm {omdocref}}
# 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)

example = element example {omdoc.toplevel.attribs, for.attrib,
                           attribute type {"for" | "against" }?,
                           attribute assertion {omdocref}?,
                           omdocmtxt.MC.content,
                           (omdocmobj.class)*}
theory = element theory {id.attrib,
                         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"}?,
                         metadata?,
                         (omdoc.class | omdocst.constitutive.class | tgroup)*}

omdocsth.imports.model = id.attrib,from.attrib,metadata?
imports = (ss| element imports {omdocsth.imports.model})
```

```
tgroup = element tgroup {omdocst.constitutive.attribs,group.attribs,
                                                    metadata?,
127                                                 (omdoc.class | omdocst.constitutive.class | tgroup)*}
```

EErr(93)
EErr(92)
EErr(91)

D.8 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 1.2) Module ADT
2 # $Id: omdocadt.rnc 8009 2008-09-07 19:02:21Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/omdocadt.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2007 Michael Kohlhase, released under the GNU Public License (GPL)

7 default namespace omdoc = "http://www.mathweb.org/omdoc"
omdoc.class |= adt

omdocadt.sym.attrib = id.attrib,omdocst.scope.attrib,attribute name {xsd:NCName}

12 # adts are abstract data types, they are short forms for groups of symbols
# and their definitions, therefore, they have much the same attributes.

adt = element adt {omdoc.toplevel.attribs,
                   attribute parameters {list {xsd:NCName*}}?, metadata?, sortdef+}
17
adttype = "loose" | "generated" | "free"
sortdef = (ss| element sortdef {omdocadt.sym.attrib,
                                attribute role {"sort"}?,
                                attribute type {adttype}?,
22                              metadata?,(constructor | insort)*,recognizer?})
insort = (ss| element insort {attribute for {omdocref}})
# for is a reference to a sort symbol element

constructor = (ss| element constructor {omdocadt.sym.attrib,
27                                       sym.role.attrib?,
                                         metadata?,argument*})
recognizer = (ss| element recognizer {omdocadt.sym.attrib,
                                      sym.role.attrib?,
                                      metadata?})
32

argument = (ss| element argument {type,selector?})
# sort is a reference to a sort symbol element p

37 selector = (ss| element selector {omdocadt.sym.attrib,
                                     sym.role.attrib?,
                                     attribute total {"yes" | "no"}?,
                                     metadata?})
```

D.9 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 1.2) Module PF
# $Id: omdocpf.rnc 8009 2008-09-07 19:02:21Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/omdocpf.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2007 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://www.mathweb.org/omdoc"

omdocpf.opt.content |= proof | proofobject
omdoc.class         |= proof | proofobject

proof = element proof {omdoc.toplevel.attribs, fori.attrib,
                       metadata?,(omtext|symbol|definition|derive|hypothesis)*}
proofobject = element proofobject {omdoc.toplevel.attribs, fori.attrib,
                                   metadata?,(omdocmobj.class)}
omdocpf.just.content = method?, premise*, (proof | proofobject)*

derive.type.attr = attribute type {("conclusion" | "gap")}

derive     = (ss| element derive {id.attrib,derive.type.attr?,
                                  omdocmtxt.MCF.content,method?})
hypothesis = (ss| element hypothesis {id.attrib,
                                      attribute inductive {"yes" | "no"}?,
                                      omdocmtxt.MCF.content})

method = (ss| element method {xref.attrib?, (omdocmobj.class|premise|proof|proofobject)*})
# 'xref' is a pointer to the element defining the method

premise = (ss| element premise {xref.attrib,
                 attribute rank {xsd:nonNegativeInteger}})

# 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.10 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 1.2) Module CTH
# $Id: omdoccth.rnc 8009 2008-09-07 19:02:21Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/omdoccth.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2007 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://www.mathweb.org/omdoc"

omdocst.constitutive.class |= inclusion
omdocsth.imports.model &= morphism?,
                          attribute type { "local" | "global"}?,
                          attribute conservativity {"conservative" | "monomorphism" | "definitional"}?,
                          attribute conservativity-just {omdocref}?

omdoc.toplevel.attribs      &= attribute generated-via {omdocref}?
omdocst.constitutive.attribs &= attribute generated-via {omdocref}?

omdoc.class |= theory-inclusion | axiom-inclusion
omdoccth.theory-inclusion.justification = obligation*
omdoccth.axiom-inclusion.justification = obligation*
```

```
fromto.attrib = from.attrib, attribute to {omdocref}
# attributes 'to' and 'from' are URIref

morphism = (ss| element morphism {id.attrib,attribute hiding {omdocrefs}?, attribute base {omdocrefs}?,def.eq?})
# base points to some other morphism it extends

inclusion = element inclusion {id.attrib,attribute via {omdocref}}
# via points to a theory-inclusion

theory-inclusion = element theory-inclusion {omdoc.toplevel.attribs,fromto.attrib,
                                             metadata?,morphism?,
                                             (omdoccth.theory-inclusion.justification)}

axiom-inclusion = element axiom-inclusion {omdoc.toplevel.attribs,fromto.attrib,
                                           metadata?,morphism?,
                                           (omdoccth.axiom-inclusion.justification)}

obligation = (ss| element obligation {id.attrib,
                                      attribute induced-by {omdocref},
                                      attribute assertion {omdocref}})

# attribute 'assertion' is a URIref, points to an assertion
# that is the proof obligation induced by the axiom or definition
# specified by 'induced-by'.
```

BErr(94)

D.11 Module DG: Development Graphs

The RNC presented in this section deals with the module CTH of development graphs (see Section 18.5 for a discussion).

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.2) Module CTH
# $Id: omdoccth.rnc 8009 2008-09-07 19:02:21Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/omdoccth.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2007 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://www.mathweb.org/omdoc"

omdocst.constitutive.class |= inclusion
omdocsth.imports.model &= morphism?,
                          attribute type { "local" | "global"}?,
                          attribute conservativity {"conservative" | "monomorphism" | "definitional"}?,
                          attribute conservativity-just {omdocref}?

omdoc.toplevel.attribs &= attribute generated-via {omdocref}?
omdocst.constitutive.attribs &= attribute generated-via {omdocref}?

omdoc.class |= theory-inclusion | axiom-inclusion
omdoccth.theory-inclusion.justification = obligation*
omdoccth.axiom-inclusion.justification = obligation*

fromto.attrib = from.attrib, attribute to {omdocref}
# attributes 'to' and 'from' are URIref

morphism = (ss| element morphism {id.attrib,attribute hiding {omdocrefs}?, attribute base {omdocrefs}?,def.eq?})
# base points to some other morphism it extends

inclusion = element inclusion {id.attrib,attribute via {omdocref}}
# via points to a theory-inclusion
```

[94] Erratum: The DG module RelaxNG schema had been forgotten

```
theory-inclusion = element theory-inclusion {omdoc.toplevel.attribs,fromto.attrib,
                                             metadata?,morphism?,
                                             (omdoccth.theory-inclusion.justification)}

axiom-inclusion = element axiom-inclusion {omdoc.toplevel.attribs,fromto.attrib,
                                           metadata?,morphism?,
                                           (omdoccth.axiom-inclusion.justification)}

obligation = (ss| element obligation {id.attrib,
                                      attribute induced-by {omdocref},
                                      attribute assertion {omdocref}})

# attribute 'assertion' is a URIref, points to an assertion
# that is the proof obligation induced by the axiom or definition
# specified by 'induced-by'.
```

EErr(94)

D.12 Module RT: Rich Text Structure

The RNC module RT provides text structuring elements for mathematical text below the level of mathematical statements (see Section 14.6 for a discussion).

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.2) Module DOC
# $Id: omdocrt.rnc 8009 2008-09-07 19:02:21Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/omdocrt.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2007 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://www.mathweb.org/omdoc"

omdoc.mtext.class |= ss|ul|ol|dl|p|note|link|table|idx

omdocrt.common.attrib = id.attrib, fori.attrib, parallel.attribs

ul = element ul {omdocrt.common.attrib, metadata?,li+}
ol = element ol {omdocrt.common.attrib, metadata?,li+}
dl = element dl {omdocrt.common.attrib, metadata?,di+}
li = element li {omdocrt.common.attrib, metadata?,(omdoc.mtext.class)*}
di = element di {omdocrt.common.attrib, metadata?,dt+,dd*}
dt = element dt {omdocrt.common.attrib, metadata?,(omdoc.mtext.class)*}
dd = element dd {omdocrt.common.attrib, metadata?,(omdoc.mtext.class)*}


p = element p {omdocrt.common.attrib, (omdoc.mtext.class)*}
note = element note {omdocrt.common.attrib,
                     attribute type {xsd:NMTOKEN}?,
                     (omdoc.mtext.class)*}


# a simplified html table
table = element table {omdocrt.common.attrib, tr+}
tr = (ss| element tr {omdocrt.common.attrib, (td|th)+})
td = (ss| element td {omdocrt.common.attrib, (omdoc.mtext.class)*})
th = (ss| element th {omdocrt.common.attrib, (omdoc.mtext.class)*})

link = element link {omdocrt.common.attrib,
                     attribute href {xsd:anyURI},
                     (omdoc.mtext.class)*}

# index
index.att = attribute sort-by {text}?,
            attribute see {omdocrefs}?,
```

```
            attribute seealso {omdocrefs}?,
            attribute links { list {xsd:anyURI*}}?
idx = element idx {(id.attrib|xref.attrib),idt?,ide+}
ide = element ide {attribute index {xsd:NCName}?,index.att,idp*}
idt = element idt {idrest.attribs,omdoc.mtext.class*}
idp = element idp {index.att,omdoc.mtext.class*}
```

D.13 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).

BErr(95)

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.2) Module EXT
# $Id: omdocext.rnc 8009 2008-09-07 19:02:21Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/omdocext.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2007 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://www.mathweb.org/omdoc"

omdoc.mtext.class |= omlet
omdocext.class = private | code | omlet
omdoc.class |= omdocext.class

omdocext.private.attrib = fori.attrib, attribute requires {omdocref}?

private = element private {omdoc.toplevel.attribs,omdocext.private.attrib,
                           attribute reformulates {omdocref}?,
                           metadata?,data+}
# reformulates is a URIref to the omdoc elements that are reformulated by the
# system-specific information in this element
code = element code {omdoc.toplevel.attribs,omdocext.private.attrib,
                     (metadata?,data+,input?,output?,effect?)}
input  = (ss| element input  {id.attrib, (omdocmtxt.MCF.content)})
output = (ss| element output {id.attrib, (omdocmtxt.MCF.content)})
effect = (ss| element effect {id.attrib, (omdocmtxt.MCF.content)})

data = (ss| element data {id.attrib,
                          attribute format {xsd:string}?,
                          attribute href {xsd:anyURI}?,
                          attribute size {xsd:string}?,
                          attribute pto {xsd:string}?,
                          attribute pto-version {xsd:string}?,
                          attribute original {"external" | "local"}?,
                          Anything})

omlet = (ss| element omlet {id.attrib,
                            attribute action     {"display" | "execute" | "other"}?,
                            attribute show       {"new" | "replace" | "embed" | "other"}?,
                            attribute actuate    {"onPresent" | "onLoad" | "onRequest" | "other"}?,
                            metadata?,
                            (omdoc.mtext.class | param)*,
                            (attribute data {xsd:anyURI}|(private|code))})

param = (ss| element param {id.attrib,
   attribute name      {xsd:string},
   attribute value     {xsd:string}?,
   attribute valuetype {"data" | "ref" | "object"}?,
   (omdocmobj.class)?})
```

[95] Erratum: we have to allow the `metadata` element in `omlet`

EErr(95)

D.14 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 1.2) Module PRES
# $Id: omdocpres.rnc 8009 2008-09-07 19:02:21Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/omdocpres.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2007 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://www.mathweb.org/omdoc"

# we include the XSLT 2 schema
include "xslt10.rnc"

#xslt10 = external "xslt10.rnc"

omdoc.class |= presentation |omstyle

crossref.attrib = attribute crossref-symbol
                    {"no" | "yes" | "brackets" | "separator" | "lbrack" | "rbrack" | "all"}
fixity.attrib = attribute fixity {"prefix" | "infix" | "postfix" | "assoc" | "infixl" | "infixr"}
format.attrib = attribute format {xsd:string}, attribute requires {omdocref}?, xml.lang.attrib
bracket-style.attrib = attribute bracket-style {"lisp" | "math"}
lbrack.attrib = attribute lbrack {xsd:string}
rbrack.attrib = attribute rbrack {xsd:string}
separator.attrib = attribute separator {xsd:string}
precedence.attrib = attribute precedence {xsd:nonNegativeInteger}
role.attrib = attribute role {"applied" | "binding" | "key"}

presentation = element presentation {omdoc.toplevel.attribs,
                                     attribute for {omdocref},
                                     role.attrib?,
                                     (xref.attrib?|
                                     (fixity.attrib?,
                                     lbrack.attrib?,
                                     rbrack.attrib?,
                                     separator.attrib?,
                                     bracket-style.attrib?,
                                     precedence.attrib?,
                                     crossref.attrib?,
                                     CMP*,
                                     (use | xslt | style)*))}

omdocpres.use.mix = elt | txt | recurse | value-of | map

use = (ss| element use {format.attrib,
   bracket-style.attrib?,
   precedence.attrib?,
   fixity.attrib?,
   lbrack.attrib?,rbrack.attrib?,separator.attrib?,
   attribute element {xsd:string}?,
   attribute attributes {xsd:string}?,
   crossref.attrib?,
   (text | omdocpres.use.mix)*})
# the attributes in the <use> element overwrite those in the
# <presentation> element, therefore, they do not have defaults

omstyle = element omstyle {omdoc.toplevel.attribs,
```

```
    attribute for {omdocref}?,
    attribute element {xsd:string}?,
    (xslt | style)*}

xslt = (ss| element xslt {format.attrib, xref.attrib?, template.model})

style = (ss| element style {format.attrib, (omdocpres.use.mix)*})
# this element contains mock xslt expressed in the elements below

elt = (ss| element element {attribute name {xsd:NMTOKEN},
                            attribute crid {xsd:string}?,
                            attribute cr {"yes" |"no"}?,
                            attribute ns {xsd:anyURI}?,
                            (attrb | omdocpres.use.mix)*})

map = (ss| element map {attribute select {xsd:string}?,
                        attribute lbrack {xsd:string}?,
                        attribute rbrack {xsd:string}?,
                        attribute precedence {xsd:string}?,
                  separator?,
                  (omdocpres.use.mix)*})

separator = (ss| element separator {(omdocpres.use.mix)*})

attrb = (ss| element attribute {attribute name {xsd:NMTOKEN},
                                attribute ns {xsd:anyURI}?,
                                (attribute select {xsd:string} |
                                 (txt | value-of)*)})

txt = (ss| element text {text})

value-of = (ss| element value-of {attribute select {xsd:string}})
recurse = (ss| element recurse {attribute select {xsd:string}?})
```

D.15 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 1.2) Module QUIZ
# $Id: omdocquiz.rnc 8373 2009-06-07 04:19:16Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/omdocquiz.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2007 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://www.mathweb.org/omdoc"
omdoc.class |= exercise | hint | mc | solution

exercise = element exercise {id.attrib, fori.attrib,
                             omdocmtxt.MCF.content,
                             hint*,
                             (solution*|mc*)}

omdocpf.opt.content = notAllowed

hint = element hint {omdoc.toplevel.attribs, fori.attrib, omdocmtxt.MCF.content}
solution = element solution {omdoc.toplevel.attribs, fori.attrib, metadata?,(omdoc.class | omgroup)*}
mc = element mc {omdoc.toplevel.attribs,fori.attrib,choice,hint?,answer}

choice = (ss| element choice {id.attrib,omdocmtxt.MCF.content})

answer = (ss| element answer {id.attrib,
```

```
attribute verdict {"true" | "false"}?,
omdocmtxt.MCF.content})
```

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 [BCC+04].

```
# RELAX NG Schema for OpenMath 2
# $Id: openmath2.rnc 8009 2008-09-07 19:02:21Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/openmath2.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 compound 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.

```
# A RelaxNG schema for MathML2
# $Id: mathml2.rnc 8009 2008-09-07 19:02:21Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/mathml2/mathml2.rnc $
# (c) 2005 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace m = "http://www.w3.org/1998/Math/MathML"
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"
namespace xlink = "http://www.w3.org/1999/xlink"
namespace local = ""

non-mathml-attribs = attribute * - (local:*|xlink:href) {xsd:string}

MathML.Common.attrib = attribute class {xsd:NMTOKENS}?,
                       attribute style {xsd:string}?,
                       attribute id {xsd:ID}?,
                       attribute xlink:href {xsd:anyURI}?,
                       non-mathml-attribs*


include "mathml2-presentation.rnc"
include "mathml2-content.rnc"

Presentation-expr.class = PresExpr.class | ContExpr.class

Content-expr.class = ContExpr.class | PresExpr.class

PresExpr.class = Presentation-token.class |
                 Presentation-layout.class |
                 Presentation-script.class |
                 Presentation-table.class |
                 mspace | maction | merror | mstyle

ContExpr.class = Content-tokens.class |
                 Content-arith.class |
                 Content-functions.class |
                 Content-logic.class |
                 Content-constants.class |
                 Content-sets.class |
                 Content-relations.class |
                 Content-elementary-functions.class |
                 Content-calculus.class |
                 Content-linear-algebra.class |
                 Content-vector-calculus.class |
                 Content-statistics. class |
                 Content-constructs.class |
                 semantics

Browser-interface.attrib = attribute baseline {xsd:string}?,
                           [a:default = "scroll"]
                           attribute overflow {"scroll" | "elide" | "truncate" | "scale"}?,
                           attribute altimg {xsd:anyURI}?,
                           attribute alttext {xsd:string}?,
                           attribute type {xsd:string}?,
                           attribute name {xsd:string}?,
                           attribute height {xsd:string}?,
                           attribute width {xsd:string}?

math.attlist = Browser-interface.attrib,
               attribute macros {xsd:string}?,
               [a:default = "inline"]
               attribute display {"block" | "inline"}?,
               MathML.Common.attrib

math.content = PresExpr.class | ContExpr.class

math = element math {math.attlist,math.content*}
```

E.2.1 Presentation MathML

```
# A RelaxNG schema for MathML2 Presentation Elements
# $Id: mathml2-presentation.rnc 8009 2008-09-07 19:02:21Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/mathml2/mathml2-presentation.rnc $
# (c) 2005 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace m = "http://www.w3.org/1998/Math/MathML"
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"

# Simple sizes
simple-size= "small" | "normal" | "big"

# Centering values
centering.values = "left" | "center" | "right"

# The named spaces
# this is also used in the value of the "width" attribute on the "mpadded" element
named-space = "veryverythinmathspace" |
              "verythinmathspace" |
              "thinmathspace" |
              "mediummathspace" |
              "thickmathspace" |
              "verythickmathspace" |
              "veryverythickmathspace"

# Thickness
thickness = "thin" | "medium" | "thick"

# number with units used to specified lengths
length-with-unit =
    xsd:string #{pattern="(-?([0-9]+|[0-9]*\.[0-9]+)*(em|ex|px|in|cm|mm|pt|pc|%))|0"}
length-with-optional-unit =
   xsd:string #{pattern="-?([0-9]+|[0-9]*\.[0-9]+)*(em|ex|px|in|cm|mm|pt|pc|%)?"}

# This is just "infinity" that can be used as a length
infinity = "infinity"

# colors defined as RGB
RGB-color = xsd:string {pattern="#(([0-9]|[a-f]){3}|([0-9]|[a-f]){6})"}

# This schema module defines sets of attributes common to several elements
# of presentation MathML.

# The mathematics style attributes. These attributes are valid on all
#     presentation token elements except "mspace" and "mglyph", and on no
#     other elements except "mstyle".

Token-style.attrib = attribute mathvariant
                       {"normal" | "bold" | "italic" | "bold-italic" | "double-struck" |
                        "bold-fraktur" | "script" | "bold-script" | "fraktur" |
                        "sans-serif" | "bold-sans-serif" | "sans-serif-italic" |
                        "sans-serif-bold-italic" | "monospace"}?,
                     attribute mathsize {simple-size | length-with-unit}?,
# For both of the following attributes the types should be more restricted
                     attribute mathcolor {xsd:string}?,
                     attribute mathbackground {xsd:string}?

# These operators are all related to operators. They are valid on "mo" and "mstyle".

Operator.attrib =
# this attribute value is normally inferred from the position of
# the operator in its "<mrow">
   attribute form {"prefix" | "infix" | "postfix"}?,
   # set by dictionnary, else it is "thickmathspace"
   attribute lspace {length-with-unit | named-space}?,
```

```
    # set by dictionary, else it is "thickmathspace"
    attribute rspace {length-with-unit | named-space}?,
    # set by dictionnary, else it is "false"
    attribute fence {xsd:boolean}?,
    # set by dictionnary, else it is "false"
    attribute separator {xsd:boolean}?,
    # set by dictionnary, else it is "false"
    attribute stretchy {xsd:boolean}?,
    # set by dictionnary, else it is "true"
    attribute symmetric {xsd:boolean}?,
    # set by dictionnary, else it is "false"
    attribute movablelimits {xsd:boolean}?,
    # set by dictionnary, else it is "false"
    attribute accent {xsd:boolean}?,
    # set by dictionnary, else it is "false"
    attribute largeop {xsd:boolean}?,
    attribute minsize {length-with-unit | named-space}?,
    attribute maxsize {length-with-unit | named-space | infinity | xsd:float}?

mglyph = element mglyph {attribute alt {xsd:string}?,
                         attribute fontfamily {xsd:string}?,
                         attribute index {xsd:positiveInteger}?}

# This is the XML schema module for the token elements of the
# presentation part of MathML.

Glyph-alignmark.class = malignmark|mglyph

# "mi" is supposed to have a default value of its "mathvariant" attribute set to "italic"
mi = element mi {Token-style.attrib, MathML.Common.attrib,(Glyph-alignmark.class|text)*}

# "mo"
mo = element mo {Operator.attrib,Token-style.attrib,MathML.Common.attrib,
                 (text|Glyph-alignmark.class)*}

# "mn"
mn = element mn {Token-style.attrib, MathML.Common.attrib,(text|Glyph-alignmark.class)*}

# "mtext"
mtext = element mtext {Token-style.attrib, MathML.Common.attrib,(text|Glyph-alignmark.class)*}

# ms (the values of "lquote" or "rquote" are not restricted to be one character strings ...)
ms = element ms {[a:default="&quot;"] attribute lquote {xsd:string}?,
                 [a:default="&quot;"] attribute rquote {xsd:string}?,
                 Token-style.attrib,MathML.Common.attrib,
                 (text|Glyph-alignmark.class)*}

# And the group of any token
Presentation-token.class = mi | mo | mn | mtext | ms

# This is an XML Schema module for the presentation elements of MathML
# dealing with subscripts and superscripts.

# "msub"
msub = element msub {attribute subscriptshift {length-with-unit}?, MathML.Common.attrib,
                     Presentation-expr.class,(Presentation-expr.class)}

# "msup"
msup = element msup {attribute supscriptshift {length-with-unit}?, MathML.Common.attrib,
                     Presentation-expr.class,Presentation-expr.class}

# "msubsup"
msubsup = element msubsup {MathML.Common.attrib,
                     attribute subscriptshift {length-with-unit}?,
                     attribute supscriptshift {length-with-unit}?,
                     Presentation-expr.class,Presentation-expr.class}
```

```
# "munder"
munder = element munder {MathML.Common.attrib,
                         attribute accentunder {xsd:boolean}?,
                         Presentation-expr.class,Presentation-expr.class}

# "mover"
mover = element mover {MathML.Common.attrib,
                       attribute accent {xsd:boolean}?,
                       Presentation-expr.class,Presentation-expr.class}

# "munderover"
munderover = element munderover {MathML.Common.attrib,
                                 attribute accentunder {xsd:boolean}?,
                                 attribute accent {xsd:boolean}?,
                                 Presentation-expr.class,
                                 Presentation-expr.class,
                                 Presentation-expr.class}

# "mmultiscripts", "mprescripts" and "none"

Presentation-expr-or-none.class = Presentation-expr.class | none

mmultiscripts = element mmultiscripts{MathML.Common.attrib,
                                      Presentation-expr.class,
                                      (Presentation-expr-or-none.class,
                                       Presentation-expr-or-none.class)*,
                                      (mprescripts,
                                       (Presentation-expr-or-none.class,
                                       Presentation-expr-or-none.class)*)?}
none = element none {empty}
mprescripts = element mprescripts {empty}

Presentation-script.class = msub|msup|msubsup|munder|mover|munderover|mmultiscripts

mspace = element mspace {[a:defaultValue = "0em"]
                         attribute width {length-with-unit | named-space}?,
                         [a:defaultValue = "0ex"]
                         attribute height {length-with-unit}?,
                         [a:defaultValue = "0ex"]
                         attribute depth {length-with-unit}?,
                         [a:defaultValue="auto"]
                         attribute linebreak {"auto" | "newline" | "indentingnewline" |
                                              "nobreak" | "goodbreak" | "badbreak"}?,
                         MathML.Common.attrib}

# This is the XML schema module for the layout elements of the
# presentation part of MathML.

# "mrow"
mrow = element mrow {MathML.Common.attrib,(Presentation-expr.class)*}

# "mfrac"

mfrac = element mfrac {attribute bevelled {xsd:boolean}?,
                       [a:defaultValue = "center"]
                       attribute denomalign {centering.values}?,
                       [a:defaultValue = "center"]
                       attribute numalign {centering.values}?,
                       [a:defaultValue="1"]
                       attribute linethickness {length-with-optional-unit|thickness}?,
                       MathML.Common.attrib,
                       Presentation-expr.class,Presentation-expr.class}
# "msqrt"
msqrt = element msqrt {MathML.Common.attrib,(Presentation-expr.class)*}

# "mroot"
mroot = element mroot {MathML.Common.attrib,Presentation-expr.class,Presentation-expr.class}
```

```
199
    # "mpadded"
    mpadded-space = xsd:string {pattern="(\+|-)?([0-9]+|[0-9]*\.[0-9]+)(((%?)*(width|lspace|height|depth))|(em|ex|px|in|cm|mm|pt|pc

    # MaxF: definition from spec seems wrong,
204 #       fixing to ([+|-] unsigned-number (%[pseudo-unit]|pseudo-unit|h-unit)) | namedspace | 0

    mpadded-width-space = xsd:string {pattern="((\+|-)?([0-9]+|[0-9]*\.[0-9]+)(((%?) *(width|lspace|height|depth)?)|(width|lspace|h
    mpadded = element mpadded {attribute width {mpadded-width-space},
                               # should have default=0 below but '0' is not in value space
209                            # see bug #425
                               attribute lspace {mpadded-space}?,
                               attribute height {mpadded-space}?,
                               attribute depth {mpadded-space}?,
                               MathML.Common.attrib,
214                            (Presentation-expr.class)*}

    # "mphantom"
    mphantom = element mphantom.attlist {MathML.Common.attrib,Presentation-expr.class*}

219 # "mfenced"
    mfenced = element mfenced {[a:defaultValue= "("] attribute open {xsd:string}?,
                               [a:defaultValue = ")"] attribute close {xsd:string}?,
                               [a:defaultValue = ","] attribute separators {xsd:string}?,
                               MathML.Common.attrib,
224                            (Presentation-expr.class)*}

    # "menclose"
    menclose = element menclose {[a:defaultValue="longdiv"]
                                 attribute notation {"actuarial"|"longdiv"|"radical"|
229                                                  "box"|"roundedbox"|"circle"|
                                                     "left"|"right"|"top"|"bottom"|
                                                     "updiagonalstrike"|"downdiagonalstrike"|
                                                     " verticalstrike "|" horizontalstrike "}?,
                                 MathML.Common.attrib,
234                              (Presentation-expr.class)*}

    # And the group of everything
    Presentation-layout.class = mrow|mfrac|msqrt|mroot|mpadded|mphantom|mfenced|menclose

239 # This is an XML Schema module for tables in MathML presentation.

    Table-alignment.attrib = [a:defaultValue = "baseline"]
            attribute rowalign
                {xsd:string {pattern="(top|bottom|center|baseline|axis)(top|bottom|center|baseline|axis)*"}}?,
244         [a:defaultValue = "center"]
            attribute columnalign
                {xsd:string {pattern="(left|center|right)( left|center|right)*"}}?,
            attribute groupalign {xsd:string}?

249 mtr = element mtr {Table-alignment.attrib, MathML.Common.attrib,mtd+}

    mlabeledtr = element mlabeledtr {Table-alignment.attrib,MathML.Common.attrib,mtd*}

    # "mtd"
254 mtd = element mtd {Table-alignment.attrib,
                       [a:defaultValue="1"] attribute columnspan {xsd:positiveInteger}?,
                       [a:defaultValue="1"] attribute rowspan {xsd:positiveInteger}?,
                       MathML.Common.attrib,
                       Presentation-expr.class*}
259
    # "mtable"
    mtable = element mtable {Table-alignment.attrib,
                             [a:defaultValue="axis"] attribute align {xsd:string}?,
                             [a:defaultValue="true"]
264                          attribute alignmentscope {xsd:string {pattern="(true|false)( true| false)*"}}?,
                             [a:defaultValue="auto"] attribute columnwidth {xsd:string}?,
```

```
                              [a:defaultValue="auto"] attribute width {xsd:string}?,
                              [a:defaultValue="1.0ex"] attribute rowspacing {xsd:string}?,
                              [a:defaultValue="0.8em"] attribute columnspacing {xsd:string}?,
                              [a:defaultValue="none"] attribute rowlines {xsd:string}?,
                              [a:defaultValue="none"] attribute columnlines {xsd:string}?,
                              [a:defaultValue="none"] attribute frame {"none" | "solid" | "dashed"}?,
                              [a:defaultValue="0.4em 0.5ex"] attribute framespacing {xsd:string}?,
                              [a:defaultValue="false"] attribute equalrows {xsd:boolean}?,
                              [a:defaultValue="false"] attribute equalcolumns {xsd:boolean}?,
                              [a:defaultValue="false"] attribute displaystyle {xsd:boolean}?,
                              [a:defaultValue="right"]
                              attribute side {"left"|"right"|"leftoverlap"|"rightoverlap"}?,
                              [a:defaultValue="0.8em"] attribute minlabelspacing {length-with-unit}?,
                              MathML.Common.attrib,
                              (mtr|mlabeledtr)*}

    # "maligngroup"
    maligngroup = element maligngroup {
          attribute groupalign {"left" | "center" | "right" | "decimalpoint"}?,
          MathML.Common.attrib}

    # "malignmark"

    malignmark = element malignmark {[a:defaultValue="left"] attribute edge {"left" | "right"}?,
                                     MathML.Common.attrib}
    Presentation-table.class = mtable|maligngroup|malignmark

    # "mstyle"
    mstyle = element mstyle {attribute scriptlevel {xsd:integer}?,
                             attribute displaystyle {xsd:boolean}?,
                             [a:defaultValue="0.71"] attribute scriptsizemultiplier {xsd:decimal}?,
                             [a:defaultValue="8pt"] attribute scriptminsize {length-with-unit}?,
                             attribute color {xsd:string}?,
                             [a:defaultValue="transparent"] attribute background {xsd:string}?,
                             [a:defaultValue="0.0555556em"] attribute veryverythinmathspace {length-with-unit}?,
                             [a:defaultValue="0.111111em"] attribute verythinmathspace {length-with-unit}?,
                             [a:defaultValue="0.166667em"] attribute thinmathspace {length-with-unit}?,
                             [a:defaultValue="0.222222em"] attribute mediummathspace {length-with-unit}?,
                             [a:defaultValue="0.277778em"] attribute thickmathspace {length-with-unit}?,
                             [a:defaultValue="0.333333em"] attribute verythickmathspace {length-with-unit}?,
                             [a:defaultValue="0.388889em"] attribute veryverythickmathspace {length-with-unit}?,
                             [a:defaultValue="1"] attribute linethickness {length-with-optional-unit|thickness}?,
                             Operator.attrib,Token-style.attrib,MathML.Common.attrib,
                             Presentation-expr.class*}

    # This is the XML Schema module for the MathML "merror" element.

    merror = element merror {MathML.Common.attrib,Presentation-expr.class*}

    # This is the XML Schema module for the MathML "maction" element.

    maction = element maction {attribute actiontype {xsd:string}?,
                               [a:defaultValue="1"] attribute selection {xsd:positiveInteger}?,
                               MathML.Common.attrib,
                               Presentation-expr.class*}
```

E.2.2 Content MathML

```
# A RelaxNG schema for MathML2 Content Elements
# $Id: mathml2-content.rnc 8009 2008-09-07 19:02:21Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/mathml2/mathml2-content.rnc $
# (c) 2005 Michael Kohlhase, released under the GNU Public License (GPL)

```

```
default namespace m = "http://www.w3.org/1998/Math/MathML"
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"

Definition.attrib = attribute encoding {xsd:string}?,
                    attribute definitionURL {xsd:anyURI}?

# This is the XML schema module for the token elements of the content part of MathML.

Content-token.content = text|Presentation-expr.class
# the content of "cn" may have <sep> elements in it

sep = element sep {empty}

cn = element cn {######attribute base {xsd:positiveInteger [1,...,36]},
                 attribute type {"e-notation"|"integer"|"rational"|"real" |
                                 "complex-cartesian"|"complex-polar"|"constant" }?,
                 Definition.attrib,
                 MathML.Common.attrib,
                 (text|sep|Presentation-expr.class)*}
# "ci"
ci = element ci {attribute type {xsd:string}?,
                 Definition.attrib,
                 MathML.Common.attrib,
                 Content-token.content}
# "csymbol"

csymbol = element csymbol {Definition.attrib, MathML.Common.attrib, Content-token.content}

# And the group of everything
Content-tokens.class = cn|ci|csymbol

# This is an XML Schema module for the "arithmetic" operators of content MathML.

Arith.type = Definition.attrib, MathML.Common.attrib

abs = element abs {Arith.type}
conjugate = element conjugate {Arith.type}
arg = element arg {Arith.type}
real = element real {Arith.type}
imaginary = element imaginary{Arith.type}

floor = element floor {Arith.type}
ceiling = element ceiling {Arith.type}

power = element power {Arith.type}
root = element root {Arith.type}

minus = element minus {Arith.type}
plus = element plus {Arith.type}
sum = element sum {Arith.type}
times = element times {Arith.type}
product = element product {Arith.type}

max = element max {Arith.type}
min = element min {Arith.type}

factorial = element factorial {Arith.type}
quotient = element quotient {Arith.type}
divide = element divide {Arith.type}
rem = element rem {Arith.type}
gcd = element gcd {Arith.type}
lcm = element lcm {Arith.type}

Content-arith.class = abs|conjugate|factorial|arg|real|imaginary|
                      floor|ceiling|quotient|divide|rem|minus|
                      plus|times|power|root|max|min|gcd|lcm|
                      sum|product
```

```
# This is an XML Schema module for operators dealing with functions in content MathML.

Functions.type = Definition.attrib, MathML.Common.attrib

# "compose"
compose = element compose {Functions.type}

# Domain, codomain and image
domain = element domain {Functions.type}
codomain = element codomain {Functions.type}
image = element image {Functions.type}

# "domainofapplication"
domainofapplication = element domainofapplication{Definition.attrib,
                                                   MathML.Common.attrib,
                                                   Content-expr.class}
# identity
ident = element ident {Functions.type}

Content-functions.class = compose|domain|codomain|image|domainofapplication|ident

# This is an XML Schema module for the logic operators of content MathML.

Logic.type = Definition.attrib,MathML.Common.attrib

and = element and {Logic.type}
or = element or {Logic.type}
xor = element xor {Logic.type}
not = element not {Logic.type}
exists = element exists {Logic.type}
forall = element forall {Logic.type}
implies = element implies {Logic.type}

Content-logic.class = and|or|xor|not|exists|forall|implies

# This is an XML Schema module for the basic constructs of content MathML.

apply = element apply {MathML.Common.attrib,Content-expr.class*}

interval = element interval {MathML.Common.attrib,
                             [a:defaultvalue = "closed"]
                             attribute closure {"closed"|"open"|"open-closed"|"closed-open"}?,
                             (Content-expr.class),(Content-expr.class)}
inverse = element inverse {Definition.attrib,MathML.Common.attrib}
condition = element condition {Definition.attrib,Content-expr.class+}

declare = element declare {attribute type {xsd:string}?,
                           attribute scope {xsd:string}?,
                           attribute nargs {xsd:nonNegativeInteger}?,
                           attribute occurrence {"prefix"|"infix"|"function-model"}?,
                           Definition.attrib,
                           Content-expr.class+}

lambda = element lambda {MathML.Common.attrib,Content-expr.class+}

#"piecewise" and its inner elements
otherwise = element otherwise {Content-expr.class,MathML.Common.attrib}
piece = element piece {MathML.Common.attrib,Content-expr.class+}
piecewise = element piecewise {MathML.Common.attrib,piece*,(otherwise,piece)*}
bvar = element bvar {MathML.Common.attrib,Content-expr.class+}
degree = element degree {MathML.Common.attrib,Content-expr.class+}


Content-constructs.class = apply|interval|inverse|condition|declare|lambda|piecewise|bvar|degree

# This is the XML Schema module for the basic constants of MathML content.
```

```
Constant.type = Definition.attrib , MathML.Common.attrib

# Basic sets
naturalnumbers= element naturalnumbers {Constant.type}
primes= element primes{Constant.type}
integers  = element integers {Constant.type}
rationals  = element rationals{Constant.type}
reals  = element reals {Constant.type}
complexes = element complexes {Constant.type}

#Empty set
emptyset = element emptyset {Constant.type}

# Basic constants
exponentiale = element exponentiale {Constant.type}
imaginaryi = element imaginaryi {Constant.type}
pi  = element pi {Constant.type}
eulergamma = element eulergamma {Constant.type}

# Boolean constants
true  = element true {Constant.type}
false  = element false {Constant.type}

# Infinty
infinit  = element infinity {Constant.type}

# NotANumber
notanumber = element notanumber {Constant.type}

Content-constants.class = naturalnumbers|primes|integers|rationals|reals|
                          complexes|emptyset|exponentiale|imaginaryi|pi|
                          eulergamma|true|false| infinit |notanumber

# This is an XML Schema module for the elementary functions in content MathML.

Elementary-functions.type = Definition.attrib,MathML.Common.attrib

# Exp and logs

exp= element exp {Elementary-functions.type}
ln  = element ln {Elementary-functions.type}
log  = element log {Elementary-functions.type}

# special element of the base of logarithms

logbase = element logbase {MathML.Common.attrib,Content-expr.class}

# Trigonometric functions

sin  = element sin {Elementary-functions.type}
cos  = element cos {Elementary-functions.type}
tan  = element tan {Elementary-functions.type}
sec  = element sec {Elementary-functions.type}
csc  = element csc {Elementary-functions.type}
cot  = element cot {Elementary-functions.type}

arcsin  = element arcsin {Elementary-functions.type}
arccos  = element arccos {Elementary-functions.type}
arctan  = element arctan {Elementary-functions.type}
arccot  = element arccot {Elementary-functions.type}
arccsc  = element arccsc {Elementary-functions.type}
arcsec  = element arcsec {Elementary-functions.type}

# Hyperbolic trigonometric functions

sinh  = element sinh {Elementary-functions.type}
cosh  = element cosh {Elementary-functions.type}
```

```
tanh = element tanh {Elementary-functions.type}
sech = element sech {Elementary-functions.type}
csch = element csch {Elementary-functions.type}
coth = element coth {Elementary-functions.type}
arccosh = element arccosh {Elementary-functions.type}
arccoth = element arccoth {Elementary-functions.type}
arccsch = element arccsch {Elementary-functions.type}
arcsech = element arcsech {Elementary-functions.type}
arcsinh = element arcsinh {Elementary-functions.type}
arctanh = element arctanh {Elementary-functions.type}

# And the group of everything

Content-elementary-functions.class =
    exp|ln|log|logbase|sin|cos|tan|sec|csc|cot|
    arcsin|arccos|arctan|arcsec|arccsc|arccot|
    sinh|cosh|tanh|sech|csch|coth|
    arccosh|arccoth|arccsch|arcsech|arcsinh|arctanh

# This is an XML Schema module for the relational operators of content MathML.

# a common type for all this
Relations.type = Definition.attrib, MathML.Common.attrib

eq = element eq {Relations.type}
neq = element neq {Relations.type}
leq = element leq {Relations.type}
lt = element lt {Relations.type}
geq = element geq {Relations.type}
gt = element gt {Relations.type}
equivalent = element equivalent {Relations.type}
approx = element approx {Relations.type}
factorof = element factorof {Relations.type}

# And the group of everything
Content-relations.class = eq|neq|leq|lt|geq|gt|equivalent|approx|factorof

# "annotation"
annotation = element annotation {attribute encoding {xsd:string}?,MathML.Common.attrib,text}

# "annotation-xml"
anyElement = element * {(attribute * {text}|text| anyElement)*}
annotation-xml = element annotation-xml {Definition.attrib,MathML.Common.attrib, anyElement}

# "semantics"
semantics = element semantics {attribute encoding {xsd:string}?,
                               attribute definitionURL {xsd:anyURI}?,
                               MathML.Common.attrib,
                               Content-expr.class,
                               (annotation|annotation-xml)*}

# This is an XML Schema module for the part of content MathML dealing with sets and lists.

# "set" ("type" could be "multiset" or "normal" or anything else)
set = element set {attribute type {xsd:string}?,
                   MathML.Common.attrib, Content-expr.class*}

# "list"
lst = element list {attribute order {"lexicographic"|"numeric"}?,
                    MathML.Common.attrib,
                    Content-expr.class*}
# "union"
union = element union {Definition.attrib, MathML.Common.attrib,Content-expr.class*}
intersect = element intersect {Definition.attrib, MathML.Common.attrib,Content-expr.class*}
in = element in {Definition.attrib, MathML.Common.attrib}
notin = element notin {Definition.attrib, MathML.Common.attrib}
subset = element subset {Definition.attrib, MathML.Common.attrib}
```

```
prsubset = element prsubset {Definition.attrib, MathML.Common.attrib}
notsubset = element notsubset {Definition.attrib, MathML.Common.attrib}
notprsubset = element notprsubset {Definition.attrib, MathML.Common.attrib}
setdiff = element setdiff {Definition.attrib, MathML.Common.attrib}
card = element card {Definition.attrib, MathML.Common.attrib}
cartesianproduct = element cartesianproduct {Definition.attrib, MathML.Common.attrib}

# And the group of everything

Content-sets.class = set|lst|union|intersect|in|notin|subset|
                     prsubset|notsubset|notprsubset|setdiff|card|cartesianproduct

# This is an XML Schema module for the linear algebra part of content MathML.

# "vector"

vector = element vector {MathML.Common.attrib,Content-expr.class*}
matrix = element matrix {MathML.Common.attrib,matrixrow+}
matrixrow = element matrixrow {MathML.Common.attrib,Content-expr.class+}
determinant = element determinant {Definition.attrib,MathML.Common.attrib}
transpose = element transpose {Definition.attrib,MathML.Common.attrib}
mselector = element selector {Definition.attrib,MathML.Common.attrib}
vectorproduct = element vectorproduct {Definition.attrib,MathML.Common.attrib}
scalarproduct = element scalarproduct {Definition.attrib,MathML.Common.attrib}
outerproduct = element outerproduct {Definition.attrib,MathML.Common.attrib}

Content-linear-algebra.class = vector|matrix|determinant|transpose|mselector|
                               vectorproduct|scalarproduct|outerproduct

# This is an XML Schema module for the calculus operators of content MathML.

calculus.type = Definition.attrib, MathML.Common.attrib

int = element int {calculus.type}
diff = element diff {calculus.type}
partialdiff = element partialdiff {calculus.type}
limit = element limit {calculus.type}
lowlimit = element lowlimit {calculus.type,Content-expr.class+}
uplimit = element uplimit {calculus.type,Content-expr.class+}
tendsto = element tendsto {calculus.type,attribute type {xsd:string}?}

Content-calculus.class = int|diff|partialdiff|limit|lowlimit|uplimit|tendsto

# This is an XML Schema module for the vector calculus operators of content MathML.

divergence = element divergence {Definition.attrib,MathML.Common.attrib}
grad = element grad {Definition.attrib,MathML.Common.attrib}
curl = element curl {Definition.attrib,MathML.Common.attrib}
laplacian = element laplacian {Definition.attrib,MathML.Common.attrib}

# And the group of everything

Content-vector-calculus.class = divergence|grad|curl|laplacian

# This is an XML Schema module for the statistical operators of content MathML.

mean = element mean {Definition.attrib,MathML.Common.attrib}
sdev = element sdev {Definition.attrib,MathML.Common.attrib}
variance = element variance {Definition.attrib,MathML.Common.attrib}
median = element median {Definition.attrib,MathML.Common.attrib}
mode = element mode {Definition.attrib,MathML.Common.attrib}
moment = element moment {Definition.attrib,MathML.Common.attrib}
momentabout = element momentabout {Definition.attrib,MathML.Common.attrib,Content-expr.class+}

Content-statistics.class = mean|sdev|variance|median|mode|moment|momentabout
```

F

The Errata

In the following we will tabulate the errata in document order. Their location will be referenced by the section they appear in rather than the page number, since we do not expect the former to change in the errata correction process.

4.2 wrong reference
4.2 wrong cross-reference for “line 16”
4.3 `for` attribute on `definition` should be of type `NCNames`
4.3 should be ”definiendum” not ”definiens”
4.3 should be definiendum-applied not definiens-applied
4.4 `for` attribute on `definition` should be of type `NCNames`
4.4 `for` attribute on `definition` should be of type `NCNames`
4.4 should be ”definiendum” not ”definiens”
5. `for` attribute on `definition` should be of type `NCNames`
6. `for` attribute on `definition` should be of type `NCNames`
6. `for` attribute on `definition` should be of type `NCNames`
7. `for` attribute on `definition` should be of type `NCNames`
7. `for` attribute on `definition` should be of type `NCNames`
7. `for` attribute on `definition` should be of type `NCNames`, totally reworked example
8.1 `for` attribute on `axiom` should be of type `NCNames`
8.1 `for` attribute on `definition` should be of type `NCNames`
8.1 forgot to thread through attribute renaming
8.1 The attribute on the `assertion` element should be `just-by`, not `proofs`. We were also missing some fragment identifiers.
11.1 Typo: “Backus Naur form” instead of “Bachus Naur Form”
11.1 ref does permit an xml:id attribute (and this should remain, as that is important for talking about refs from an RDF point of view)
11.1 omdoc and omgroup can have an optional theory attribute as well
11.2 RDF as a general data model is independent from XML; RDF/XML is just one of its possible serializations.
11.2 correct name

12. The content Model for `dc:creator` and `cd:contributor` is simple text
12.1 wrong attribute name
12.4 `for` attribute on `definition` should be of type `NCNames`
13.1.1 It should be made clear that this inheritance mechanism is extended by the OMDoc format. See section **3.1** of the errata document for details
14. added the attribute xml:id to the `CMP` element; added the attribute from to the `omtext` element
14. added the attribute cdbase to the `term` element
14.1 should be "definiendum" not "definiens"
14.3 `note`
14.3 omtext can also be an assumption, obligation or rule as all of these can be expressed in informal as well as formal way
14.3 And there should also be ¡omtext type="assertion"¿ for generic assertions, corresponding to the ¡assertion¿ element without a type.
14.5 also need `cdbase` for identifying
14.5 Should be Deffiniendum instead of deffiniens
14.5 should be "definiendum" not "definiens"
14.6 the index attribute should be optional
15.1 "Definiendum" and "Definiens" should switched
15.2.1 scope is deprecated
15.2.2 the `for` attribute in the `axiom` element must reference symbol names
15.2.2 `for` attribute on `axiom` should be of type `NCNames`
15.2.3 examples reference wrong listings
15.2.4 Note that this use of the `for` attribute is different from the other usages, which are URI references.
15.2.4 `for` attribute on `definition` should be of type `NCNames`, also corrected cd attribute.
15.2.4 `for` attribute on `definition` should be of type `NCNames`
15.2.4 `for` attribute on `definition` should be of type `NCNames`
15.3 deleted spurious `for` attribute on the `assertion` element, `alternative` should have the same content as definition
15.3.2 `for` attribute on `definition` should be of type `NCNames`
15.3.3 fixed the target of the for attribute
15.4 added the `axiom` element to the list; cf. discussion on `omdoc-dev` on May 16, 2008
15.4 added the `alternative` element to the list
15.4 `for` attribute on `definition` should be of type `NCNames`
15.5 `for` attribute on `definition` should be of type `NCNames`
15.5 should be "definendum" not "definiens"
15.5 `for` attribute on `definition` should be of type `NCNames`
15.5 should be "definiendum" not "definiens"
15.6 the `xml:id` attribute on the `theory` element should be optional
15.6.1 The symbol name `af` should be `aa`
15.6.1 `for` attribute on `definition` should be of type `NCNames`
15.6.1 `for` attribute on `definition` should be of type `NCNames`

15.6.2 This specification of the inheritance mechanism is to wishy washy. See section **3.1** of the errata document for a clarification.
16..2 The `for` attribute contains a URI reference according to the RelaxNG schema; the locality restriction here contradicts that and needs to be removed.
17.1 `for` attribute on `definition` should be of type `NCNames`
17.1 made the for attribute in the `proofobject` element required; added the rank attribute to the `premise` element
17.2 `for` attribute on `definition` should be of type `NCNames`
17.2 `for` attribute on `definition` should be of type `NCNames`
17.3 `for` attribute on `definition` should be of type `NCNames`
17.4 `for` attribute on `definition` should be of type `NCNames`
18. changed the order of type and hiding attributes in the `morphism` element; removed the consistency and consistency-just attributes from the `morphism`, `inclusion`, `theory-inclusion`, and `axiom-inclusion` elements; changed the contents of the `theory-inclusion` element to (morphism?, obligation*); changed the contents of the `morphism` element to (requation+, measure?, ordering?); added the element `obligation`
18.1 noted special case
18.1 Clarified wording
18.2 added missing word
18.2 Fixed value of the `conservativity` attribute
18.2 Fixed value of the `conservativity` attribute
18.2 Fixed value of the `conservativity` attribute
18.5.2 added the optional for attribute for the decomposition element; removed the by attribute from the `theory-inclusion` element; changed the contents of the `theory-inclusion` element to (morphism?, (decomposition* — obligation*))
19..2 added CMP* to content of `presentation` element
19.4 The `for` attribute should be `#X4` instead of `#X` in listings 19.5 and 19.6
20.1 The reference `reformulates="ALGX0"` should be a URI reference, i.e. `#ALGX0`
20.2 Wrong Content Model for `omlet`
22.1 `for` attribute on `definition` should be of type `NCNames`
22.1 `for` attribute on `definition` should be of type `NCNames`
22.2 `for` attribute on `definition` should be of type `NCNames`
26.4 reference to QED
26.15 The domain is `kwarc.eecs.iu-bremen.de`
26.15.4 correct example given
1.1 the old `extradata` content has nothing to do with `dc:subject`
4.4 The `type` attributes on `phrase` and `omtext` were not conforming to the spec
4.4 attribute value `trasition` forgotten from rnc
4.4 the `verbalizes` attribute had been forgotten for the `phrase` element

4.7 simple definitions should not have an `existence` attribute, furthermore pattern definitions should not have `measure` and `ordering` children
4.7 the `type` element needs to allow a `for` attribute
4.7 the `tgroup` element should not contain `omgroup` children
4.10 The DG module RelaxNG schema had been forgotten
4.13 we have to allow the `metadata` element in `omlet`

References

ABC+03a. Ron Ausbrooks, Stephen Buswell, David Carlisle, Stéphane Dalmas, Stan Devitt, Angel Diaz, Max Froumentin, Roger Hunter, Patrick Ion, Michael Kohlhase, Robert Miner, Nico Poppelier, Bruce Smith, Neil Soiffer, Robert Sutor, and Stephen Watt. Mathematical Markup Language (MathML) version 2.0 (second edition). W3C recommendation, World Wide Web Consortium, 2003.

ABC+03b. Ron Ausbrooks, Stephen Buswell, David Carlisle, Stéphane Dalmas, Stan Devitt, Angel Diaz, Max Froumentin, Roger Hunter, Patrick Ion, Michael Kohlhase, Robert Miner, Nico Poppelier, Bruce Smith, Neil Soiffer, Robert Sutor, and Stephen Watt. Mathematical Markup Language (MathML) version 2.0 (second edition). W3C recommendation, World Wide Web Consortium, 2003.

ABD03. Andrea Asperti, Bruno Buchberger, and James Harold Davenport, editors. *Mathematical Knowledge Management, MKM'03*, number 2594 in LNCS. Springer Verlag, 2003.

ABF+03. Serge Autexier, Christoph Benzmüller, Armin Fiedler, Helmut Horacek, and Quoc Bao Vo. Assertion level proof representation with underspecification. In Fairouz Kamareddine, editor, *Proceedings of MKM Symposium*, Heriot-Watt, Edinburgh, November 2003.

ABFL06. Serge Autexier, Christoph Benzmüller, Armin Fiedler, and Henri Lesourd. Integrating proof assistants as reasoning and verification tools into a scientific wysiwig editor. *Proceedings of UITP'05*, 2006.

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

Abn96. S. Abney. Partial parsing via finite-state cascades, 1996. `http://citeseer.ifi.unizh.ch/abney96partial.html`.

ABT04. Andrea Asperti, Grzegorz Bancerek, and Andrej Trybulec, editors. *Mathematical Knowledge Management, MKM'04*, number 3119 in LNAI. Springer Verlag, 2004.

AF05. Serge Autexier and Armin Fiedler. Textbook proofs meet formal logic - the problem of underspecification and granularity. In Michael Kohlhase, editor, *Proceedings of MKM'05*, volume 3863 of *LNAI*, IUB Bremen, Germany, june 2005. Springer.

AH05. Serge Autexier and Dieter Hutter. Formal software development in maya. In Dieter Hutter and Werner Stephan, editors, *Festschrift in Honor of J. Siekmann*, volume 2605 of *LNAI*. Springer, february 2005.

AHL+00. S. Autexier, D. Hutter, B. Langenstein, H. Mantel, G. Rock, A. Schairer, W. Stephan, R. Vogt, and A. Wolpers. Vse: Formal methods meet industrial needs. *International Journal on Software Tools for Technology Transfer, Special issue on Mechanized Theorem Proving for Technology*, 3(1), september 2000.

AHMS99. S. Autexier, D. Hutter, H. Mantel, and A. Schairer. System description: INKA 5.0 – a logical voyager. In H. Ganzinger, editor, *16th International Conference on Automated Deduction, CADE-16*, volume 1732 of *Lecture Notes in Artificial Intelligence*, Trento, 1999. Springer.

AHMS00. Serge Autexier, Dieter Hutter, Heiko Mantel, and Axel Schairer. Towards an evolutionary formal software-development using CASL. In C. Choppy and D. Bert, editors, *Proceedings Workshop on Algebraic Development Techniques, WADT-99*, number 1827 in LNCS, pages 73–88. Springer Verlag, 2000.

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.

AKC03. Andrea Asperti, Michael Kohlhase, and Claudio Sacerdoti Coen. Prototype n. d2.b document type descriptors: OMDoc proofs. Mowgli deliverable, The MoWGLI Project, 2003.

Alt01. Modularization of xhtml. W3C recommendation, The World Wide Web Consortium, 2001.

AM02. Serge Autexier and Till Mossakowski. Integrating holcasl into the development graph manager maya. In Alessandro Armando, editor, *Frontiers of Combinning Systems (FROCOS'02)*, number 2309 in LNAI, pages 2–17. Springer Verlag, 2002.

And02. Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Kluwer Academic Publishers, second edition, 2002.

APCS01. Andrea Asperti, Luca Padovani, Claudio Sacerdoti Coen, and Irene Schena. HELM and the semantic math-web. In Richard. J. Boulton and Paul B. Jackson, editors, *Theorem Proving in Higher Order Logics: TPHOLs'01*, volume 2152 of *LNCS*, pages 59–74. Springer Verlag, 2001.

Aut03. Serge Autexier. *Hierarchical Contextual Reasoning*. PhD thesis, Saarland University, 2003.

Aut05. Serge Autexier. The core calculus. In Robert Nieuwenhuis, editor, *Proceedings of the 20th International Conference on Automated Deduction (CADE-20)*, volume 3632 of *LNAI*, Tallinn, Estonia, july 2005. Springer.

Bar80. Hendrik P. Barendregt. *The Lambda-Calculus: Its Syntax and Semantics*. North-Holland, 1980.

Bau99. Judith Baur. Syntax und Semantik mathematischer Texte — ein Prototyp. Master's thesis, Fachrichtung Computerlinguistik, Universität des Saarlandes, SaarbrückenGermany, 1999.

BB01. P. Baumgartner and A. Blohm. Automated deduction techniques for the management of personalized documents. In Buchberger and Caprotti [BC01b].

BC01a. Henk Barendregt and Arjeh M. Cohen. Electronic communication of mathematics and the interaction of computer algebra systems and proof assistants. *Journal of Symbolic Computation*, 32:3–22, 2001.

BC01b. Bruno Buchberger and Olga Caprotti, editors. *Electronic Proceedings of the First International Workshop on Mathematical Knowledge Management: MKM'2001*, 2001.

BCC+04. Stephen Buswell, Olga Caprotti, David P. Carlisle, Michael C. Dewar, Marc Gaetano, and Michael Kohlhase. The Open Math standard, version 2.0. Technical report, The Open Math Society, 2004.

BCD+02. R. Bradford, R. M. Corless, J. H. Davenport, D. J. Jeffrey, and S. M. Watt. Reasoning about the elementary functions of complex analysis. *Annals of Mathematics and Artificial Intelligence*, 36:303 – 318, 2002.

BCF+97. C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. ΩMEGA: Towards a mathematical assistant. In McCune [McC97], pages 252–255.

BDD+99. Stephen Buswell, Stan Devitt, Angel Diaz, Patrick Ion, Robert Miner, Nico Poppelier, Bruce Smith, Neil Soiffer, Robert Sutor, and Stephen Watt. Mathematical Markup Language (MathML) 1.01 specification. W3c recommendation, World Wide Web Consortium (W3C), 1999.

Ber91. Paul Bernays. *Axiomatic Set Theory*. Dover Publications, 1991.

BF06. Jon Borwein and William M. Farmer, editors. *Mathematical Knowledge Management, MKM'06*, number 4108 in LNAI. Springer Verlag, 2006.

BL98. Tim Berners-Lee. The semantic web, 1998.

BLFM98. Tim Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifiers (URI), Generic Syntax. RFC 2717, Internt Engineering Task Force, 1998.

Blo56. B.S. Bloom, editor. *Taxonomy of educational objectives: The classification of educational goals: Handbook I, cognitive domain*. Longmans, Green, New York, Toronto, 1956.

BM79. R. S. Boyer and J S. Moore. *A Computational Logic*. ACM monograph series. Academic Press, New York, 1979.

BM01. Paul V. Biron and Ashok Malhotra. XML Schema Part 2: Datatypes. W3C recommendation, World Wide Web Consortium, May 2001.

Bos98. Cascading style sheets, level 2; css2 specification. W3C recommendation, World Wide Web Consortium (W3C), 1998.

Bou74. Nicolas Bourbaki. *Algebra I*. Elements of Mathematics. Springer Verlag, 1974.

BPSM97. Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML). W3C Recommendation TR-XML, World Wide Web Consortium, December 1997.

BPSM+04. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, and John Cowan. Extensible Markup Language (XML) 1.1. W3C Recommendation REC-xml11-20040204, World Wide Web Consortium, 2004.

Bra99. Namespaces in xml. W3C recommendation, The World Wide Web Consortium, 1999.

Bug05. Bugzilla. web page at `http://www.bugzilla.org`, seen 2005.

BvHHS90. A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam system. In M. E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 647–648. Springer-Verlag, 1990. Lecture Notes in Artificial Intelligence No. 449. Also available from Edinburgh as DAI Research Paper 507.

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

CCC+00. Olga Caprotti, Arjeh M. Cohen, Hans Cuypers, Manfred N. Riem, and Hans Sterk. Using OPENMATH servers for distributing mathematical computations. In Wei Chi Yang, Sung-Chi Chu, and Jen-Chung Chuan, editors, *ATCM 2000: Proceedings of the Fifth Asian Technology Conference in Mathematics,*, pages 325–336, Chiang-Mai, Thailand, 2000. ATCM, Inc.

CCR00. Olga Caprotti, Arjeh M. Cohen, and Manfred Riem. Java Phrasebooks for Computer Algebra and Automated Deduction. *Bulletin of the ACM Special Interest Group on Symbolic and Automated Mathematics (SIGSAM)*, 34(2):43–48, 2000.

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

CD99. James Clark and Steve DeRose. XML Path Language (XPath) Version 1.0. W3C recommendation, The World Wide Web Consortium, November 1999.

CGG+92. Bruce W. Char, Keith O. Geddes, Gaston H. Gonnet, Benton L. Leong, Michael B. Monagan, and Stephen M. Watt. *First leaves: a tutorial introduction to Maple V*. Springer Verlag, Berlin, 1992.

CGM+04. R. Conejo, E. Guzman, E. Millan, M. Trella, J. L. Perez de-la Cruz, and A. Rios. SIETTE: A Web-Based Tool for Adaptive Teaching. *International Journal of Artificial Intelligence in Education (IJAIED 2004)*, 14:29–61, 2004.

CKOS03. Edmund Clarke, Michael Kohlhase, Joël Ouaknine, and Klaus Sutner. System description: Analytica 2. In Volker Sorge and Olga Caprotti, editors, *Proceedings of the 11th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus-2003)*, 2003.

Cla97. James Clark. Comparison of sgml and xml. World Wide Web Consortium Note, 1997.

Cla99. Associating style sheets with xml documents version 1.0. W3C recommendation, World Wide Web Consortium (W3C), 1999.

Cla05. James Clark. nXML mode. web page at `http://www.thaiopensource.com/nxml-mode/`, seen 2005.

CM98. A.M. Cohen and L. Meertens. The ACELA project: Aims and plans. In N. Kajler, editor, *Computer-Human interaction in Symbolic Computation*, Texts and Monographs in Symbolic Computation, pages 7–23. Springer Verlag, 1998.

Coe05. Claudio Sacerdoti Coen. Explanation in natural language of $\overline{\lambda}\mu\overline{\mu}$-terms. In Kohlhase [Koh05a].

CoF04. CoFI (The Common Framework Initiative). CASL *Reference Manual*. LNCS 2960 (IFIP Series). Springer, 2004.

Com. Userland Com. XML Remote Procedure Call Specification. http://www.xmlrpc.com/.

Con01. IMS Global Learning Consortium. Learnig resource metadata specification, 2001.

Cor. Microsoft Corp. Microsoft internet explorer. web page at http://www.microsoft.com/windows/ie.

Cow04. XML information set (second edition), February 2004.

Crea. Creative Commons. web page at http://creativecommons.org. seen August 2006.

Creb. Metadata Commons Worldwide. web page at http://creativecommons.org/learn/technology/metadata.

Crec. Creative Commons Worldwide. web page at http://creativecommons.org/worldwide.

Dah01. Ingo Dahn. Slicing book technology - providing online support for textbooks. In *The 20th ICDE World Conference on Open Learning and Distance Education*, 2001.

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

de 94. N. G. de Bruijn. The mathematical vernacular, a language for mathematics with typed sets. In R. P Nederpelt, J. H. Geuvers, and R. C. de Vrijer, editors, *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*, pages 865 – 935. Elsevier, 1994.

Des05. Deskzilla. web page at http://www.deskzilla.com, seen 2005.

dH01. Joris Van der Hoeven. Gnu TeXMacs: A free, structured, wysiwyg and technical text editor. *Cahiers GUTenberg*, pages 39–40, May 2001.

DMOT01. Steve DeRose, Eve Maler, David Orchard, and Ben Trafford. XML linking language (XLink version 1.0). W3C recommendation, W3C, 2001.

DOM. Document object model DOM. web page at http://www.w3.org/DOM/.

DUB03a. The DCMI Usage Board. DCMI metadata terms. DCMI recommendation, Dublin Core Metadata Initiative, 2003.

DUB03b. The DCMI Usage Board. DCMI type vocabulary. DCMI recommendation, Dublin Core Metadata Initiative, 2003.

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

Duc98. Denys Duchier. The NeGra tree bank. Private communication, 1998.

DW05. Mark Davis and Ken Whistler. Unicode collation algorithm, 2005. Unicode Technical Standard #10.

ea07. Peter Murray-Rust et al. Chemical markup language (CML). http://cml.sourceforge.net/, seen January 2007.

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

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.

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

FH97. Amy P. Felty and Douglas J. Howe. Hybrid interactive theorem proving using NuPRL and HOL. In McCune [McC97], pages 351–365.

FH01. Armin Fiedler and Helmut Horacek. Argumentation in explanations to logical problems. In Vassil N. Alexandrov, Jack J. Dongarra, Benjoe A. Juliano, Renè S. Renner, and C. J. Kenneth Tan, editors, *Computational Science — ICCS 2001*, number 2074 in LNCS, pages 969–978, San Francisco, CAUSA, 2001. Springer Verlag.

FHJ+99a. A. Franke, S. Hess, C. Jung, M. Kohlhase, and V. Sorge. Agent-Oriented Integration of Distributed Mathematical Services. *Journal of Universal Computer Science*, 5(3):156–187, March 1999. Special issue on Integration of Deduction System.

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

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

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

Fie01a. Armin Fiedler. Dialog-driven adaptation of explanations of proofs. In Bernhard Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1295–1300, Seattle, WAUSA, 2001. Morgan Kaufmann.

Fie01b. Armin Fiedler. *User-adaptive Proof Explanation.* Phd thesis, Naturwissenschaftlich-Technische Fakultät I, Universität des Saarlandes, Saarbrücken, Germany, 2001.

FK99. Andreas Franke and Michael Kohlhase. System description: MathWeb, an agent-based communication layer for distributed automated theorem proving. In Harald Ganzinger, editor, *Automated Deduction — CADE-16*, number 1632 in LNAI, pages 217–221. Springer Verlag, 1999.

Fre91. Free Software Foundation. Gnu general public license, 1991.

Fre99. Free Software Foundation. GNU lesser general public license, 1999.

GB92. J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39:95–146, 1992. Predecessor in: LNCS 164, 221–256, 1984.

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

GGP03. Georgi Goguadze and Alberto González Palomo. Adapting mainstream editors for semantic authoring of mathematics. Presented at the Mathematical Knowledge Management Symposium, Heriot-Watt University, Edinbourgh, Scotland, November 2003.

GGPM05. Georgi Goguadze, Alberto González Palomo, and Erica Melis. Interactivity of exercises in activemath. In *Accepted to the International Conference on Computers in Education (ICCE 2005)*, Singapore, 2005.

GHMN03. Martin Gudgin, Marc Hadley, Jean-Jacques Moreau, and Henrik Frystyk Nielsen. Soap 1.2 part 1: Adjuncts, 2003.

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

GMMW03. Paul Grosso, Eve Maler, Jonathan Marsh, and Norman Walsh. Xpointer framework. W3C recommendation, World Wide Web Consortium W3C, 25 March 2003.

GMUC03. G. Goguadze, E. Melis, C. Ullrich, and P. Cairns. Problems and solutions for markup for mathematical examples and exercises. In A. Asperti, B. Buchberger, and J.H. Davenport, editors, *International Conference on Mathematical Knowledge Management, MKM03*, LNCS 2594, pages 80–93. Springer-Verlag, 2003.

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

GPa. Alberto González Palomo. Algebra.

GPb. Alberto González Palomo. Qmath history. `http://www.matracas.org/qmath/history.html`.

GR02. J. Goguen and G. Rosu. Institution morphisms. *Formal aspects of computing*, 13:274–307, 2002.

Gra96. Peter Graf. *Term Indexing*. Number 1053 in LNCS. Springer Verlag, 1996.

Gro99. The Open eBook Group. Open ebook[tm] publication structure 1.0. Draft recommendation, The OpenEBook Initiative, 1999.

Gro00. 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), 2000.

GUM+04. G. Goguadze, C. Ullrich, E. Melis, J. Siekmann, Ch. Gross, and R. Morales. LeActiveMath Structure and Metadata Model. Deliverable D6, LeActiveMath Consortium, 2004. accessible from http://www.leactivemath.org/.

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

Har03. Eliotte Rusty Harold. *Effective XML*, chapter 15. Addison Wesley, 2003.

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

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

HKW96. Reiner Hähnle, Manfred Kerber, and Christoph Weidenbach. Common syntax of dfg-schwerpunktprogramm "deduktion". Interner Bericht 10/96, Universität Karlsruhe, Fakultät für Informatik, 1996.

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

Hua96. Xiaorong Huang. *Human Oriented Proof Presentation: A Reconstructive Approach*. Number 112 in DISKI. Infix, Sankt Augustin, Germany, 1996.

Hut00. Dieter Hutter. Management of change in verification systems. In *Proceedings 15th IEEE International Conference on Automated Software Engineering, ASE-2000*, pages 23–34. IEEE Computer Society, 2000.

IAN. Root-zone whois information. http://www.iana.org/cctld/cctld-whois.htm.

IEE02. IEEE Learning Technology Standards Committee. 1484.12.1-2002 IEEE standard for Learning Object Metadata, 2002.

Inc03. Unicode Inc., editor. *The Unicode Standard, Version 4.0*. Addison-Wesley, 2003.

JEN08. Jena — A Semantic Web Framework for Java, seen June 2008. web page at http://jena.sf.net.

JFF02. Dean Jackson, Jon Ferraiolo, and Jun Fujisawa. Scalable vector graphics (svg) 1.1 specification. W3c candidate recommendation, World Wide Web Consortium (W3C), April 2002.

JSP. JavaServer Pages. web page at http://java.sun.com/products/jsp.

KA03. Michael Kohlhase and Romeo Anghelache. Towards collaborative content management and version control for structured mathematical knowledge. In Asperti et al. [ABD03], pages 147–161.

KAB+04. E. Klieme, H. Avenarius, W. Blum, P. Döbrich, H. Gruber, M. Prenzel, K. Reiss, K. Riquarts, J. Rost, H. Tenorth, and H. J. Vollmer. The development of national educational standards - an expertise. Technical report, Bundesministerium für Bildung und Forschung / German Federal Ministry of Education and Research, 2004.

Kay. Michael Kay. Saxon, the xslt and xquery processor. Web page at saxon.sf.net.

KBKB+04. B. Krieg-Brückner, B. Krämer, D. Basin, J. Siekmann, and M. Wirsing. Multimedia Instruction in Safe and Secure Systems. Abschlussbericht, Universität Bremen, 2004. BMBF project 01NM070, 2001-2004.

KBLL+04. Bernd Krieg-Brückner, Arne Lindow, Christoph Lüth, Achim Mahnke, and George Russell. Semantic interrelation of documents via an ontology. In G. Engels and S. Seehusen, editors, *DeLFI 2004*, volume P-52 of *LNI*, pages 271–282. Springer-Verlag, 2004.

KD03a. Michael Kohlhase and Stan Devitt. Bound variables in mathml. W3C Working Group Note, 2003. http://www.w3.org/TR/mathml-bvar, pubs = mkohlhase.

KD03b. Michael Kohlhase and Stan Devitt. Structured types in mathml 2.0. W3C Note, 2003.

KF00. M. Kohlhase and A. Franke. MBase: Representing knowledge and context for the integration of mathematical software systems. *Journal of Symbolic Computation*, 2000.

KF01. Michael Kohlhase and Andreas Franke. MBase: Representing knowledge and context for the integration of mathematical software systems. *Journal of Symbolic Computation; Special Issue on the Integration of Computer Algebra and Deduction Systems*, 32(4):365–402, 2001.

KK04. Andrea Kohlhase and Michael Kohlhase. CPoint: Dissolving the author's dilemma. In Asperti et al. [ABT04], pages 175–189.

KK06. Andrea Kohlhase and Michael Kohlhase. Communities of Practice in MKM: An Extensional Model. In Borwein and Farmer [BF06], pages 179–193.

KM96. M. Kaufmann and J S. Moore. ACL2: An industrial strength version of Nqthm. In *Compass'96: Eleventh Annual Conference on Computer Assurance*, page 23, Gaithersburg, Maryland, 1996. National Institute of Standards and Technology.

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

Koha. Michael Kohlhase. CodeML: An open markup format the content and presentatation of program code. Internet Draft at `https://svn.omdoc.org/repos/codeml/doc/spec/codeml.pdf`.

Kohb. Michael Kohlhase. OMDoc: An open markup format for mathematical documents (latest released version). Specification, `http://www.omdoc.org/pubs/spec.pdf`.

Kohc. Michael Kohlhase. The OMDoc Document Type Definition. `http://omdoc.org/dtd/omdoc.dtd`.

Kohd. Michael Kohlhase. The OMDoc RelaxNG schema. `http://omdoc.org/rnc/omdoc.rnc`.

Kohe. Michael Kohlhase. The OMDoc XML schema. `http://omdoc.org/rnc/omdoc.xsd`.

Kohf. Michael Kohlhase. XSL style sheets for OMDoc. `http://omdoc.org/xsl/`.

Koh05a. Michael Kohlhase, editor. *Mathematical Knowledge Management, MKM'05*, number 3863 in LNAI. Springer Verlag, 2005.

Koh05b. Michael Kohlhase. Semantic markup for TEX/LATEX. 2005.

Koh05c. Michael Kohlhase. Inference rules. OMDoc Content Dictionary at `https://svn.omdoc.org/repos/omdoc/trunk/examples/logics/inference-rules.omdoc`, seen Jan 2005.

Koh06. Andrea Kohlhase. What if PowerPoint became emPowerPoint (through CPoint)? In Caroline M. Crawford, editor, *Society for Information Technology and Teacher Education, 17th International Conference SITE 2006*, pages 2934–2939. SITE, AACE, 2006. Orlando (USA), 2006-03-20/24.

Koh08. Michael Kohlhase. OMDoc mailing lists. `http://omdoc.org/resources/mailing-lists.html`, seen May 2008.

KR93. Hans Kamp and Uwe Reyle. *From Discourse to Logic*. Kluwer, Dordrecht, 1993.

KZ95. D. Kapur and H. Zhang. An overview of rewrite rule laboratory (RRL). *J. Computer and Mathematics with Applications*, 29(2):91–114, 1995.

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

LC01. Bo Leuf and Ward Cunningham. *The Wiki Way: Collaboration and Sharing on the Internet*. Addison-Wesley Professional, 2001.

Len04. Richard Lennox. Development of an RDF/XML based data model for bibliographic data. Dissertation for Bachelor of Science in Computer Science, 2004. http://richardlennox.net/dissertation.pdf.

Lib04. P. Libbrecht. Authoring web content in activemath: From developer tools and further. In Alexandra Christea and Franca Garzotto, editors, *Proceedings of the Second International Workshop on Authoring Adaptive and Adaptable Educational Hypermedia, AH-2004: Workshop Proceedings, Part II, CS-Report 04-19*, pages 455–460. Technische Universiteit Eindhoven, 2004.

Lom05. Cyprien Lomas. 7 things you should know about social bookmarking. `http://www.educause.edu/ir/library/pdf/ELI7001.pdf`, 2005. Seen March 2006.

LS99. Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. W3C recommendation, World Wide Web Consortium (W3C), 1999.

MAH06. Till Mossakowski, Serge Autexier, and Dieter Hutter. Development graphs – proof management for structured specifications. *Journal of Logic and Algebraic Programming*, 67(1–2):114–145, 2006.

MAR03. MARC code list for relators, sources, description conventions, 2003. Web Version at `http://www.loc.gov/marc/relators`.

Mat. Using the `mathweb.org` subversion repository. Web page at `http://www.mathweb.org/svn.html`.

Max. Maxima - a gpl cas based on doe-macsyma. web page at `http://maxima.sourceforge.net`.

MBa. Mbase. `http://mbase.mathweb.org:8000`.

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

MBG+03. Erica Melis, Jochen Büdenbender, George Goguadze, Paul Libbrecht, and Carsten Ullrich. Knowledge representation and management in activemath. *Annals of Mathematics and Artificial Intelligence*, 38:47–64, 2003. see `http://www.activemath.org`.

McC97. William McCune, editor. *Proceedings of the 14th Conference on Automated Deduction*, number 1249 in LNAI, Townsville, Australia, 1997. Springer Verlag.

Mei00. Andreas Meier. System description: TRAMP: Transformation of machine-found proofs into ND-proofs at the assertion level. In David McAllester, editor, *Automated Deduction – CADE-17*, number 1831 in LNAI, pages 460–464. Springer Verlag, 2000.

Mes89. J. Meseguer. General logics. In *Logic Colloquium 87*, pages 275–329. North Holland, 1989.

MG04. E. Melis and G. Goguadze. Towards adaptive generation of faded examples. In *International Conference on Intelligent Tutoring Systems*, number 3220 in LNCS, pages 762–771. Springer-Verlag, 2004.

MGDT05. Till Mossakowski, Joseph Goguen, Razvan Diaconescu, and Andrzej Tarlecki. What is a logic? In Jean-Yves Beziau, editor, *Logica Universalis*, pages 113–133. Birkhäuser, 2005.

MGH+05. Erica Melis, Giorgi Goguadze, Martin Homik, Paul Libbrecht, Carsten Ullrich, and Stefan Winterstein. Semantic-aware components and services of activemath. *British Journal of Educational Technology*, 2005.

MGP04. M. Mavrikis and A. González Palomo. Mathematical, interactive exercise generation from static documents. *Electronic Notes in Computer Science*, 93:183–201, 2004.

Mil07. Bruce Miller. `LaTeXML`: A LaTeX to xml converter. Web Manual at `http://dlmf.nist.gov/LaTeXML/`, seen September2007.

Mit03. Nilo Mitra. Soap 1.2 part 0: Primer, 2003.

Miz06. Mizar language. web page at `http://mizar.org/language`, seen III 2006.

Miz08. Mizar mathematical library. Web Page at `http://www.mizar.org/library`, seen May 2008.

MKH05. E. Melis, P. Kärger, and M. Homik. Interactive Concept Mapping in ActiveMath (iCMap). In Djamshid Tavangarian Jörg M. Haake, Ulrich Lucke, editor, *Delfi 2005: 3. Deutsche eLearning Fachtagung Informatik*, volume 66 of *LNI*, pages 247–258. Gesellschaft für Informatik e.V. (GI), -, 2005. accepted.

MMLW. T. Mossakowski, Christian Maeder, Klaus Lüttich, and Stefan Wölfl. The heterogeneous tool set. Submitted for publication.

Mon. MONET – Mathematics on the net, an EU funded project. web page at `http://monet.nag.co.uk`.

Mos02. Till Mossakowski. Heterogeneous development graphs and heterogeneous borrowing. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures (FOSSACS02)*, number 2303 in LNCS, pages 310–325. Springer Verlag, 2002.

Mos04. T. Mossakowski. Hetcasl - heterogeneous specification. language summary, 2004.

Mos05. T. Mossakowski. Heterogeneous specification and the heterogeneous tool set. Habilitation thesis, University of Bremen, 2005.

Moz. The mozart programming system.

MS96. M.A. McRobbie and J.K. Slaney, editors. *Proceedings of the 13th Conference on Automated Deduction*, number 1104 in LNAI, New Brunswick, NJ, USA, 1996. Springer Verlag.

MSLK01. M. Murata, S. St. Laurent, and D. Kohn. Xml media types. RFC 3023, January 2001.

MTea04. Robert Meersman, Zahir Tari, and Angelo Corsaro et al., editors. *On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops*, number 3292 in LNCS. Springer Verlag, 2004.

Mül05. Normen Müller. OMDoc-Repräsentation von Programmen und Beweisen in VeriFun. Master's thesis, Programmiermethodik, Technische Universität Darmstadt, 2005.

MVW05. Jonathan Marsh, Daniel Veillard, and Norman Walsh. `xml:id` version 1.0. W3C recommendation, World Wide Web Consortium, September 2005.

NS81. Alan Newell and Herbert A. Simon. Computer science as empirical inquiry: Symbols and search. *Communications of the Association for Computing Machinery*, 19:113–126, 1981.

OAI02. The open archives initiative protocol for metadata harvesting, June 2002.

Odl95. A.M. Odlyzko. Tragic loss or good riddance? the impending demise of traditional scholarly journals. *International Journal of Human-Computer Studies*, 42:71–122, 1995.

OM . OpenMath. web page at `http://www.openmath.org`.

OMC08. OpenMath content dictionaries. web page at `http://www.openmath.org/cd/`, seen June2008.

OMDa. The omdoc subversion repository. Repository at `https://svn.omdoc.org/repos/omdoc`.

OMDb. The OMDoc wiki. `http://www.mathweb.org/omdoc/wiki/`.

Org. The Mozilla Organization. Mozilla. web page at `http://www.mozilla.org`.

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

Pau94. Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*. LNCS. Springer Verlag, 1994.

PB04. F. Piroi and B. Buchberger. An environment for building mathematical knowledge libraries, 2004. `http://citeseer.ifi.unizh.ch/piroi04environment.html`.

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

Pfe01. Frank Pfenning. Logical frameworks. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume I and II. Elsevier Science and MIT Press, 2001.

Pie80. John R. Pierce. *An Introduction to Information Theory. Symbols, Signals and Noise*. Dover Publications Inc., 1980.

PN90. Lawrence C. Paulson and Tobias Nipkow. Isabelle tutorial and user's manual. Technical Report 189, Computer Laboratory, University of Cambridge, January 1990.

PS99. F. Pfenning and C. Schürmann. System description: Twelf — A meta-logical framework for deductive systems. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, pages 202–206, Trento, Italy, 1999. Springer-Verlag LNAI 1632.

PSBKK04. Manfred Pinkal, Jörg Siekmann, Christoph Benzmüller, and Ivana Kruijff-Korbayova. Dialog: Natural language-based interaction with a mathematics assistance system. Project proposal in the Collaborative Research Centre SFB 378 on Resource-adaptive Cognitive Processes, 2004.

QED96. The QED project. `http://www-unix.mcs.anl.gov/qed/`, 1996.

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

RHJ98. Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.0 Specification. W3C Recommendation REC-html40, World Wide Web Consortium, April 1998.

ROM. ROML, The RIACA OpenMath Library. web page at `http://crystal.win.tue.nl/download/`.

RSG98a. J.D.C. Richardson, A. Smaill, and I. Green. System description: Proof planning in higher-order logic with lambda-clam. In C. Kirchner and H. Kirchner, editors, *Conference on Automated Deduction (CADE'98)*, volume 1421 of *Lecture Notes in Computer Science*, pages 129–133. Springer-Verlag, 1998.

RSG98b. Julian D.C. Richardson, Alan Smaill, and Ian M. Green. System description: Proof planning in higher-order logic with λ*clam*. In Claude Kirchner and Hélène Kirchner, editors, *Proceedings of the 15th Conference on Automated Deduction*, number 1421 in LNAI. Springer Verlag, 1998.

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

SBA05. Jörg Siekmann, Christoph Benzmüller, and Serge Autexier. Computer supported mathematics with omega. *Journal of Applied Logic, special issue on Mathematics Assistance Systems*, december 2005.

SBB+02. Jörg Siekmann, Christoph Benzmüller, Vladimir Brezhnev, Lassaad Cheikhrouhou, Armin Fiedler, Andreas Franke, Helmut Horacek, Michael Kohlhase, Andreas Meier, Erica Melis, Markus Moschner, Immanuel Normann, Martin Pollet, Volker Sorge, Carsten Ullrich, Claus-Peter Wirth, and Jürgen Zimmer. Proof development with OMEGA. In Andrei Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction (CADE-18)*, number 2392 in LNAI, pages 144–149, Copenhagen, Denmark, 2002. Springer.

SBB+06. Sebastian Schaffert, Diana Bischof, Tobias Bürger, Andreas Gruber, Wolf Hilzensauer, and Sandra Schaffert. Learning with semantic wikis. In Max Völkel, Sebastian Schaffert, and Stefan Decker, editors, *Proceedings of the 1st Workshop on Semantic Wikis, European Semantic Web Conference 2006*, volume 206 of *CEUR Workshop Proceedings*, Budva, Montenegro, June 2006.

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

Sch04. Klaus Schneider. *Verification of Reactive Systems*. Springer Verlag, 2004.

Sch06. Sebastian Schaffert. IkeWiki: A semantic wiki for collaborative knowledge management. Technical report, Salzburg Research Forschungsgesellschaft, 2006.

Sci. Design Science. Mathplayer ¡display math in your browser¿. web page at `http://www.dessci.com/en/products/mathplayer`.

SHB+99. Jörg Siekmann, Stephan M. Hess, Christoph Benzmüller, Lassaad Cheikhrouhou, Armin Fiedler, Helmut Horacek, Michael Kohlhase, Karsten Konrad, Andreas Meier, Erica Melis, Martin Pollet, and Volker Sorge. $\mathcal{LOUI}$: $\mathcal{L}$ovely ΩMEGA $\mathcal{U}$ser $\mathcal{I}$nterface. *Formal Aspects of Computing*, 11:326–342, 1999.

Smo95. G. Smolka. The Oz programming model. In Jan van Leeuwen, editor, *Computer Science Today*, volume 1000 of *LNCS*, pages 324–343. Springer-Verlag, Berlin, 1995.

SPH97. J. Siekmann, F. Pfenning, and X. Huang, editors. *Proceedings of the First International Workshop on Proof Transformation and Presentation*, Schloss DagstuhlGermany, 1997.

SS98. G. Sutcliffe and C. Suttner. The TPTP problem library: CNF release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

SSY94. Geoff Sutcliffe, Christian Suttner, and Theodor Yemenis. The TPTP problem library. In Alan Bundy, editor, *Proceedings of the 12th Con-*

ference on Automated Deduction, number 814 in LNAI, Nancy, France, 1994. Springer Verlag.

Sta02. Richard M. Stallman. *GNU Emacs Manual*. GNU Press, 15 edition, 2002. online at `http://www.gnu.org/manual/emacs-21.2`.

Sut01. G. Sutcliffe. The CADE-17 ATP system competition. *Journal of Automated Reasoning*, 27(3):227–250, 2001.

Sut06. Klaus Sutner. Converting MATHEMATICA notebooks to OMDoc. In OMDoc – *An open markup format for mathematical documents [Version 1.2]*, number 4180 in LNAI, chapter 26.17. Springer Verlag, 2006.

SZS04. G. Sutcliffe, J. Zimmer, and S. Schulz. TSTP Data-Exchange Formats for Automated Theorem Proving Tools. In W. Zhang and V. Sorge, editors, *Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems*, number 112 in Frontiers in Artificial Intelligence and Applications, pages 201–215. IOS Press, 2004.

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

The. The Apache Software Foundation. Xalan-java. Web page at `http://xml.apache.org/xalan-j`.

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

Tob. Richard Tobin. Rxp - an XML parser available under the GPL. System Home page at `http://www.cogsci.ed.ac.uk/~richard/rxp.html`.

TS06. Robert Tolksdorf and Elena Paslaru Bontas Simperl. Towards wikis as semantic hypermedia. In Dirk Riehle and James Noble, editors, *Proceedings of the 2006 International Symposium on Wikis*, ACM Press, August 2006.

Ull04. C. Ullrich. Description of an instructional ontology and its application in web services for education. In *Poster Proceedings of the 3rd International Semantic Web Conference, ISWC2004*, pages 93–94, Hiroshima, Japan, 2004.

Ull05. C. Ullrich. Tutorial planning: Adapting course generation to today's needs. In M. Grandbastian, editor, *Young Researcher Track Proceedings of 12th International Conference on Artificial Intelligence in Education*, pages 155–160, Amsterdam, 2005.

ULWM04. C. Ullrich, P. Libbrecht, S. Winterstein, and M. Mühlenbrock. A flexible and efficient presentation-architecture for adaptive hypermedia: Description and technical evaluation. In Kinshuk, C. Looi, E. Sutinen, D. Sampson, I. Aedo, L. Uden, and E. Kähkönen, editors, *Proceedings of the 4th IEEE International Conference on Advanced Learning Technologies (ICALT 2004)*, pages 21–25, 2004.

Vat. Irène Vatton. Welcome to amaya. web page at `http://www.w3.org/Amaya`.

Veia. Daniel Veillard. The XML c parser and toolkit of gnome; libxml. System Home page at `http://xmlsoft.org`.

Veib. Daniel Veillard. The xslt c library for gnome: libxslt. Web page at `http://xmlsoft.org/XSLT/`.

VKS+07. Max Völkel, Malte Kiesel, Sebastian Schaffert, Björn Decker, and Eyal Oren. Semantic wiki state of the art paper – ontoworld. `http://ontoworld.org/index.php/Semantic_Wiki_State_of_The_Art_Paper`, seen January 2007.

VKV+06. Max Völkel, Markus Krötzsch, Denny Vrandečić, Heiko Haller, and Rudi Studer. Semantic Wikipedia. In *Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh, Scotland, May 23–26, 2006*, May 2006.

Vli03. Eric van der Vlist. *Relax NG*. O'Reilly, 2003.

Wei97. Christoph Weidenbach. SPASS: Version 0.49. *Journal of Automated Reasoning*, 18(2):247–252, 1997. Special Issue on the CADE-13 Automated Theorem Proving System Competition.

Wir. Wiris cas. web page at `http://www.wiris.com/overview/products/wiris-cas.html`.

WM99. Norman Walsh and Leonard Muellner. *DocBook: The Definitive Guide*. O'Reilly, 1999.

Wol00. Stephen Wolfram. Mathematical notation, past and future. In *International MathML Conference*, 2000.

Wol02. Stephen Wolfram. *The Mathematica Book*. Cambridge University Press, 2002.

WR10. Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*, volume I. Cambridge University Press, Cambridge, Great Britain; second edition, 1910.

WS02. Christoph Walther and Stephan Schweitzer. The VeriFun Tutorial. Technical Report VFR 02/04, Programmiermethodik, Technische Universität Darmstadt, 2002.

XML. XML schema. Web page at `http://www.w3.org/XML/Schema`.

XSL99. Xsl transformations (xslt) version 1.0. W3c recommendation, W3C, 1999.

Yac. The yacas computer algebra system. web page at `http://www.xs4all.nl/~apinkus/yacas.html`.

Zim04. Jürgen Zimmer. A Framework for Agent-based Brokering of Reasoning Services. In Raul Monroy, Gustavo Arroyo Figueroa, and L. Enrique Sucar, editors, *Proceedings of the Mexican International Conference on Artificial Intelligence 2004*. Springer-Verlag, 2004. to appear.

ZK02. Jürgen Zimmer and Michael Kohlhase. System Description: The Mathweb Software Bus for Distributed Mathematical Reasoning. In Andrei Voronkov, editor, *Automated Deduction — CADE-18*, number 2392 in LNAI, pages 247–252. Springer Verlag, 2002.

Index

ΩMEGA, XIII, 23, 25, 253, 300, 321–323, 326–329, 449
$\overline{\lambda}\mu E\mu$
  calculus, 170
$\mathcal{T}$-theorem, 187
`dc:*`
  element, 340
`om:*`
  element, 343
`-->`, 8, 98, 229
`.rnc`, 246
`.rng`, 246
`<!--`, 8, 98, 229
`&`, 8
`'`, 8
`>`, 8
`<`, 8
`"`, 8
OPENMATH
  object, 114

639
  ISO, 103
  ISO (), 106, 130, 254
8601
  ISO, 103
  ISO (), 40, 105, 354, 355

abbreviation
  namespace, 8, 245
Abelian
  semigroup, 161
about data
  data, 93, 98, 103
abstract, 105
  data
    type, 274
  data type, 60, 165, 233
  structure, 43
  syntax, 43
`abstract`
  attribute value
    for `type` on `omtext`, 133
Abstract Data Types
  RNC Module ADT, 378
  spec Module ADT, 93, 155, 165, 166, 232, 233, 353–355, 358
ACL2, 293
`action`
  attribute
    on `dc:date`, 105, 347, 348, 351
    on `omlet`, 220, 221, 349, 361
active
  document, 215
ACTIVEMATH, XIII, 37, 71, 73, 80, 96, 102, 257, 258, 261, 281–288, 307, 313
actual
  parameter
    theory, 62
actualization, 60
`actuate`
  attribute, 220
    on `omlet`, 220
acyclic, 160
  directed (), 118, 119, 131, 169, 231
`adt`

element, 146, 152, 165–168, 231, 332, 333, 339, 344, 347
ADT (Abstract Data Types)
RNC Module, 378
spec Module, 93, 155, 165, 166, 232, 233, 353–355, 358
advisor
thesis, 107
against
attribute value
for type on example, 47, 155
agent
web, 5
Alan
Bundy, 26
Algebra
Fundamental Theorem of, 14
ALGEBRA, 265
algebra
system, 29
algebraic
hierarchy, 183
specification, 23, 27, 60, 337
all
attribute value
for crossref-symbol on presentation, 208, 209
α-conversion, 125
alphabet, 156
alternative
attribute value
for type on omgroup, 349
alternative
element, 154, 155, 338, 347
alternative
element, 154
AMAYA, 16
analysis
formal, 327
natural language, 327
ANALYTICA, 217
analyzer, 311
anchor
named, 10
Andrzej
Trybulec, 27
animal, 72
annotated
semantically (), 328
annotation
semantic, 328
m:annotation
element, 122
m:annotation-xml
element, 17, 122, 211
answer
element, 224, 339
ant
attribute value
for role on dc:*, 107
antecedent
bibliographic, 107
scientific, 107
antithesis
attribute value
for type on omtext, 133
mq:anyorder
attribute, 269–271
applet, 219
Java, 221
application, 18, 114, 122
XML, 6
web, 279
XML, 3, 14, 90, 228, 234
application
attribute value
for role on symbol, 145
application/omdoc+xml, 41
applied
attribute value
for role on presentation, 207, 349
apply
element, 270
m:apply
element, 16, 18, 122, 145, 207, 210, 252
archiving
document, 37
argument
element, 60, 167, 338, 347
arith1, 29, 233
arith1.ocd, 20
artefacts
electronic, 108
artificial
intelligence, 301
asked
frequently (), 237

assertion, 75, 169, 172, 321
  level, 326
assertion
  attribute value
    for type on omtext, 134
assertion
  element, 75, 76, 83, 147, 149–152, 154–157, 172, 174–176, 189, 195, 243, 298, 331, 333, 338, 347, 362, 368
assertion
  attribute
    on example, 50, 76, 155–157
    on obligation, 189
assertion-level proof
  underspecification, 329
assertional
  element, 155, 156, 179, 181, 188
assertions
  type, 152
assessment, 223
assistant
  mathematical (), 25
  proof, 326
assoc
  attribute value
    for fixity on presentation, 207
assumption, 172
  local, 173
assumption
  attribute value
    for type on assertion, 151, 343
    for type on omtext, 134
assumption
  element, 132, 153, 176, 332, 339
ATP, 293
attribute, 7
  CSS, 91, 100, 102, 136, 340, 344, 345, 372
  default value, 243
  node, 7
  type, 9
attribute
  element, 203, 339, 347
attribute-value
  pair, 203
attributes
  attribute
    on use, 209, 351
attribution
  attribute value
    for role on symbol, 116, 145, 345
attribution
  attribute
    on cc:requirements, 110
  attribute value, 145
augmented, 338
aural, 91
aut
  attribute value
    for role on dc:*, 107
author, 107
authoring
  semantic (), 306
automated
  concept formation
    system, 81
  deduction, 27, 337
  proof
    assistant, 81
  theorem
    prover, 25, 81
    provers, 321
automated theorem prover, 29
AutoMath, 23
axiom, 23, 93, 133, 141, 172, 185, 233, 321
  commutativity, 161
  implicit, 147
  inclusion, 190, 194
  system, 143
axiom
  attribute value
    for type on omtext, 42, 133, 343
axiom
  element, 73, 74, 146–148, 155, 161, 168, 173–176, 339, 347
axiom inclusion, 193
axiom-inclusion
  element, 187, 191, 195, 196, 231, 297, 298, 339, 343, 347, 364, 365
axioms, 148
  Peano, 142, 143, 165

background
  scientific, 2
Backus Naur form
  notation, 97

backward
reasoning, 178
balanced
bracketing
structure, 7
base
content dictionary, 163
knowledge, 173, 301
morphism, 187
URI, 160
`base`
attribute
on `morphism`, 67, 187
Berners-Lee
Tim, 32
Bibliographic
Record
Schema, 262
bibliographic
antecedent, 107
binary
document model, 228
`binary`
attribute value
for `format` on `data`, 217
`binder`
attribute value
for `role` on `symbol`, 116, 145, 345
`binder`
attribute value
for `role`, 116
binding, 122, 125, 210
LaTeXML, 311
object, 145
operator, 116, 117
`binding`
attribute value
for `role` on `presentation`, 207, 349
binding structure, 18, 114
binomial
coefficient, 209
body, 116
bottom-up
proof
step, 178
bound
occurrence, 126
variable, 18, 114, 116, 125, 210
box
layout, 91
Boyer-Moore
corpus, 294
`bracket-style`
attribute
on `presentation`, 207
on `use`, 209
bracketing
balanced (), 7
`brackets`
attribute value
for `crossref-symbol` on
`presentation`, 208, 209
browser, 5, 249
browsing, 261
BugZilla, 238
building blocks, 89
bullet
symbol, 100
bulleted
list, 138
Bundy
Alan, 26
`m:bvar`
element, 18, 122, 126, 145, 207, 210
by pointing
semantics, 19
byte array, 117

C
programming
language, 18
calculus, 326
$\overline{\lambda}\mu E\mu$, 170
formal, 30
logical, 169
process, 301
Carnegie Mellon University, 71
Cartesian
product, 45
CAS, 29, 277, 278, 280
CASC competition, 293
cascades
finite-state, 263
Cascading
Style Sheet, 91
cascading
style sheet, 338

CASL, 23, 59, 165, 233, 296, 297, 300, 303
catalog
  XML, 9, 40, 239, 242
Catholic
  church, 129
CC
  license, 108
  metadata, 108
CC (Creative Commons Metadata)
  spec Module, 93, 103, 108, 355, 357, 358
cc:, 40, 109
cc:license
  element, 108, 109
cc:permissions
  element, 41, 109
cc:prohibitions
  element, 41, 109
cc:requirements
  element, 41, 109
CCAPS, 305, 320
cd
  attribute value
    for module on omdoc, 44
cd
  attribute
    on om:OMS, 31, 114, 125
    on OMS, 18, 45, 232
    on term, 136, 345
cd*
  attribute
    on theory, 54
cd2omdoc.xsl, 249
CDATA, 77
  section, 8
CDATA, 217
cdbase
  attribute, 115
    on om:OMOBJ, 19
    on om:theory, 158
    on OMS, 158
    on term, 136
    on theory, 160, 162, 163, 345
omcd:CDDefinition
  element, 115
omcd:CDName
  element, 54
cdreviewdate
  attribute
    on theory, 163, 345
cdrevision
  attribute
    on theory, 162, 345
cdstatus
  attribute
    on theory, 162, 345
omcd:CDURL
  element, 54
cdurl
  attribute
    on theory, 163, 345
cdversion
  attribute
    on theory, 162, 345
chain
  local, 193, 195
challenge problems
  induction, 293, 294
change
  management, 192, 233, 273, 275
  management of, 299
changed, 337
chapter, 22, 100
character
  hash, 10
  lists of, 63
character data
  parsed, 97
checking
  proof, 30
chemistry
  vernacular, 130
choice
  element, 224, 339
church
  Catholic, 129
m:ci
  element, 16, 121, 126, 210, 252
cite
  attribute value
    for type on ref, 101
CLAM, 293
clarity
  conceptual, XI
class
  CSS, 200
  equivalence, 228

`class`
  attribute, 91, 92, 96, 100, 139, 200, 224, 338
    on `dc:title`, 77
    on `omstyle, presentation`, 200, 201
    on `phrase`, 135, 201
    on `presentation`, 201, 206
    on `ref`, 78, 102
class definition
  CSS, 77
classical
  first-order
    logic, 176
`classid`
  old attribute on `code` (deprecated in OMDoc 1.2), 340, 347
  old attribute on `private` (deprecated in OMDoc 1.2), 350
clause
  copyleft, 41
`clb`
  attribute value
    for `role` on `dc:*`, 107
client, 279
clipboard
  system, 261
closing
  tag, 7, 41
`cmml`
  attribute value
    for `format` on `use`, 201
`CMP`
  element, 7, 41, 42, 47, 49, 99, 104, 105, 130–135, 137, 138, 146, 148, 150, 155, 173, 174, 180, 188, 201, 205, 218, 219, 223, 224, 230, 231, 252, 254, 257, 288, 295, 339, 340, 343, 345, 347, 350, 351, 354, 355, 358, 359
`omcd:CMP`
  element, 20
`m:cn`
  element, 16, 121
co-reference, 17
code
  country, 130, 254
  fragment, 77, 130
`code`
  element, 77, 174, 201, 216–218, 220–222, 231, 340, 347, 350, 366
`codebase`
  old attribute on `code` (deprecated in OMDoc 1.2), 340, 347
  old attribute on `private` (deprecated in OMDoc 1.2), 350
coefficient
  binomial, 209
collaborator, 107
`Collection` as Dublin Core Type, 106
`Collection`, 106
collection
  multi-format, 222
color
  text, 91
comma, 208
comment, 98
  persistent, 99
  source, 98
  XML, 8, 98, 229
`comment`
  attribute value
    for `type` on `omtext`, 99, 133
commented
  mathematical
    property, 20
`commercial_use`
  attribute
    on `cc:permission`, 109
common
  greatest (), IX
communication, 277, 326
  standard, 256
community, 27, 337
commutativity
  axiom, 161
comorphism, 302
`competency`
  element, 283
complete
  configuration, 274
Complex Theories
  DG Module CTH, 380
  RNC Module CTH, 379
  spec Module CTH, 93, 183, 185, 187, 191, 192, 232, 233, 296, 303, 353, 355, 356, 358
Component

Presentation, 281
Tutorial, 281
component
reuse, 296
shared, 296
composition, 187
computation
request, 321
computer
graphics, IX
science, 71
computer algebra, IX
system, 19, 25, 29, 81, 117, 216, 217, 252, 321
computer science
vernacular, 130
computer-supported
education, 27, 337
concatenation
strings, 156
concept, 73, 75, 135, 144
extension, 73
mathematical, 144
concept formation
automated (), 81
conceptual
clarity, XI
structure, 256
conceptual clarity, 89
`conclusion`
attribute value
for `type` on `derive`, 172
for `type` on `omtext`, 133
`conclusion`
element, 132, 153, 176, 332, 340, 348
configuration
complete, 274
consistent, 274
management, 273
Conjecture
Kepler's, IX
conjecture, 151
false, 157
`conjecture`
attribute value
for `type` on `assertion`, 151
for `type` on `omtext`, 134, 343
ConneXions, 271
consequence, 30
morphism, 297
relation, 30
conservative, 188
extension, 143, 154
principle of (), 23
`conservative`
attribute value
for `conservativity`, 188, 189
for `definitional`, 188
conservativity, 187
`conservativity`
attribute, 188
on `axiom-inclusion`, 339
on `imports`, 187, 341
on `inclusion`, 341
on `theory-inclusion`, 346
`conservativity-just`
attribute
on `axiom-inclusion`, 339
on `imports`, 187, 341
on `inclusion`, 341
on `theory-inclusion`, 346
consistency, 143, 185
`consistency`
attribute
on `definition`, 149
on `morphism`, 185, 342
consistent
configuration, 274
`constant`
attribute value
for `role` on `symbol`, 145
constitutive, 141, 233
property, 20
theory element, 142
constraint
solver, 81
construct
programming, 278
constructor, 45, 60
symbol, 165, 166
term, 165
`constructor`
element, 166, 167, 333, 340, 347
Content
markup, 30
content, 95, 116
dictionary, 18, 113, 125, 135, 278
language, 321

management, 72
markup, 4, 28, 30, 31
navigation, 307
OMDoc, 72
search, 307
semantic, 327
structure, 134
content dictionary, 122
content dictionary, 249
content dictionaries
OpenMath (), 294
content dictionary, 18, 19, 21, 31, 45, 46, 114, 115, 130
base, 163
format, 162
metadata, 162
OMDoc, 44, 162, 232
OpenMath, 162
status, 162
version number, 162
Content in
PowerPoint, 305
content OMDoc, 231
Content MathML, 10, 15–19, 21, 22, 27, 29, 31, 37, 42, 43, 113, 120–123, 125–127, 132, 134, 136, 162, 201, 207, 211, 249, 251, 252, 270, 312, 321, 323, 337, 342
context, 151, 178
dynamic, 278, 279
dynamic (), 280
markup, 22, 28
mathematical, 134
static, 278, 279
static (), 280
context-free
grammar, 8, 241
`contradictory-axioms`
attribute value
for `status` on `assertion`, 153
`contrast`
attribute value
for `type` on `omgroup`, 349
`dc:contributor`
element, 104, 105, 107, 110, 340, 348
control
version, 273
controlled
refinement, 256
convention
structural, 2
copyleft
clause, 41
`copyleft`
attribute
on `cc:requirements`, 110
CoQ, 253, 321, 323, 324
Core, 326, 328, 329, 428
Core, 326
corollary, 151
`corollary`
attribute value
for `type` on `assertion`, 151
for `type` on `omtext`, 134, 343
corpus
Boyer-Moore, 294
Dmac, 294
correct, 72
correctness
management, 173
`counter-equivalent`
attribute value
for `status` on `assertion`, 153
counter-example, 76, 157
`counter-satisfiable`
attribute value
for `status` on `assertion`, 153
`counter-theorem`
attribute value
for `status` on `assertion`, 153
country
code, 130, 254
Course Capsules, 314
courseware, 71
CPoint, 71, 256, 305–309, 314
CPointAuthor, 307, 308
CPointBasic, 308
CPointGraphs, 307
CPointImport, 308
CPointNotes, 308
CPointStudent, 308
`cr`
attribute
on `element`, 206, 341
`created`
attribute value
for `action` on `dc:date`, 105
Creative Commons

Initiative, 98, 109
license, 41, 108
namespace, 40, 109
URI, 109
Creative Commons Metadata
spec Module CC, 93, 103, 108, 355, 357, 358
dc:creator
element, 104, 105, 107, 110, 340
cref
attribute
on om:*, 343
crid
attribute
on element, 206, 341
cross-reference, 5, 17, 72, 102, 104, 118, 131, 133, 139, 172, 175, 206, 208, 343, 349, 362
crossref-symbol
attribute
on presentation, use, 208
on presentation, 209
CSS
attribute, 91, 100, 102, 136, 340, 344, 345, 372
class, 200
class definition, 77
directive, 91, 338
markup, 219
property, 102
style sheet, 91, 238
CSS, 6, 15, 16, 77, 91, 92, 96, 100, 102, 135, 136, 173, 200, 219, 338, 340, 344, 345, 362, 372
m:csymbol
element, 121–123, 126, 134
CTH (Complex Theories)
DG Module, 380
RNC Module, 379
spec Module, 93, 183, 185, 187, 191, 192, 232, 233, 296, 303, 353, 355, 356, 358
Curry-Howard
isomorphism, 325
cyclic, 101

DAG, 118, 131, 169, 172, 174, 175
explosion, 118
data
about data, 93, 98, 103
abstract (), 274
table, 138
data
attribute value
for valuetype on param, 222
data
element, 77, 217, 218, 220, 222, 231, 340, 344, 347, 348, 350, 363, 365, 366
data
attribute
on omlet, 220, 221, 349
data type
abstract, 60, 165, 233
Dataset as Dublin Core Type, 105
Dataset, 40, 105, 106, 231
dataset
attribute value
for type on omgroup (deprecated in OMDoc 1.2), 100, 342
date
review, 163
dc:date
element, 105, 110, 340, 347, 348, 351
dateTime, 105
dc
attribute
on action:date, 105
DC (Dublin Core Metadata)
RNC Module, 374
spec Module, 93, 103, 104, 232, 234, 354–359
dc:, 40, 104
dc:*
element, 340
dc:contributor
element, 104, 105, 107, 110, 340, 348
dc:creator
element, 104, 105, 107, 110, 340
dc:date
element, 105, 110, 340, 347, 348, 351
dc:description
element, 45, 54, 105, 146, 231, 339, 340, 345, 347, 350
dc:format
element, 106, 110
dc:identifier
element, 106

dc:language
element, 106, 110
dc:publisher
element, 105, 340
dc:relation
element, 106
dc:rights
element, 106, 108, 110
dc:source
element, 54, 106, 110
dc:subject
element, 105, 146, 339, 340, 344, 345
dc:title
element, 77, 100, 104, 231, 345
dc:type
element, 105, 106, 110, 231
dd
element, 138
de, 106, 130
de-referencing, 6
decision
procedure, 25
declaration, 135
DOCTYPE, 243
document type, 9, 10, 243
local, 178
namespace, 7, 8, 40, 90
namespace prefix, 44
symbol, 22, 147, 162
term, 147, 346
type, 147
decomposition, 62, 194
decomposition
element, 195, 196, 340, 346, 348, 351
deduction
automated, 27, 337
natural, 180, 348
natural (), 85, 177
default
namespace, 90
default
attribute value
for format on use, 201
default value
attribute, 243
defined
symbol, 148
definiendum, 143, 148
definiens, 143, 157, 180
definiens
attribute value
for role on term, 45, 136
defining
occurrence, 125, 210
definite
description
operator, 66
definition, 23, 93, 133, 137, 141, 143, 157, 233, 321
document type, 44, 227, 241, 244
implicit, 143
inductive, 144
loose, 144
simple, 143
definition
attribute value
for type on omtext, 42, 133, 343
definition
element, 45, 49, 50, 148, 149, 154, 155, 158, 161, 173, 174, 180, 185, 281, 331, 333, 340, 342, 347, 348, 350
definition by description, 148
definitional, 188
form, 23
theory
inclusion, 188
definitional
attribute value
for conservativity, 188
definitionURL, 121
definitionURL
attribute
on m:annotation, 122
on m:ci, 126
on m:csymbol, 122, 126, 134
dependency
graph, 307
deprecated, 337
derivative_works
attribute
on cc:permissions, 109
derive
attribute value
for type on omtext, 134
derive
element, 76, 85, 172–178, 332, 348
derived

inference
rule, 180
description
definite (), 66
list, 138
logic, 301
service, 321
dc:description
element, 45, 54, 105, 146, 231, 339, 340, 345, 347, 350
Deskzilla, 238
development
graph, 62, 192, 233, 296, 297, 299
proof (), 25
theory (), 216
time, 296
Development Graphs
RNC Module DG, 380
spec Module DG, 93, 183, 192, 195, 232, 303, 354, 357
sped Module DG, 296
DG (Development Graphs)
RNC Module, 380
spec Module, 93, 183, 192, 195, 232, 303, 354, 357
sped Module, 296
DG Module
CTH (Complex Theories), 380
Dgrl, 297, 299, 300
di
element, 138
dictionary
content, 18, 113, 125, 135, 278
didactic
figure, 79, 80
differential
equation, 14, 29
difficulty
element, 283
Digital
rights
management, 108
digital
universal (), X
Digital Media in Education, 305
directed
acyclic
graph, 118, 119, 131, 169, 231
directive
CSS, 91, 338
discharged-in
old attribute on hypothesis (deprecated in OMDoc 1.2), 341, 348
discourse
structure, 256
discourse theory, 141
display
attribute value
for action on omlet, 220, 221
display:none
attribute value
for style, 158
distinction
presentation vs. content, 95
distribution, 41
distribution
attribute
on cc:permissions, 109
distributivity, 20
m:divide
element, 16
dl
element, 130, 138
Dmac
corpus, 294
DOC (Document Structure)
RNC Module, 373
spec Module, 89, 93, 95, 130, 232, 234, 355–358
DocBook, 234, 278
DOCTYPE
declaration, 243
DOCTYPE, 242, 243, 245
document
active, 215
archiving, 37
fragment, 103
hypertext, 5
individualized, 256
interactive, 256
knowledge-centered, 72, 254
lexical (), 228
library, 262
management, 6, 103
manager, 279
markup, 1
mathematical, 2, 21, 326

model, 228
multilingual, 254
narrative-structured, 72, 254
object model, 219, 227
preparation language, 256
retrieval, 37
reuse, 273
root, 7, 9, 89, 105, 232
semantic (), 328
server, 279
sharing, 273
source, 4
structure, 233, 275
structured, 276
target, 4
tree, 6
view, 262
document model
binary, 228
document object
model, 18
Document Structure
RNC Module DOC, 373
spec Module DOC, 89, 93, 95, 130, 232, 234, 355–358
document type, 242
declaration, 9, 10, 243
definition, 44, 227, 241, 244
document type definition, 8
document-unique, 9
documents
multilingual, 130
DOM, 295
JAXP, 295
domain
top-level, 109
dominate, 119
DRM, 108
DSSSL, 6
dt
element, 138
DTD, 8, 41, 44, 91, 96, 127, 227, 238, 241, 243, 369
module, 244
normalization, 243
DTD Module
PF (Proofs and Arguments), 244
DTD module, 242
Dublin Core, 98, 315
namespace, 40, 104
URI, 104
Dublin Core Metadata
RNC Module DC, 374
spec Module DC, 93, 103, 104, 232, 234, 354–359
DVI, 4
dynamic
context, 278, 279
manager, 280
HTML, 219
mathematical
context, 277

editing, 261
editor, 107
invasive, 256
scientific, 326
editorial
note, 308
edt
attribute value
for role on dc:*, 107
education, XII
computer-supported, 27, 337
Educational OMDoc, 234
effect
element, 77, 218, 340
effective
URI, 160
elaboration
attribute value
for type on omtext, 133
electronic
artefacts, 108
element, 6
assertional, 155, 156, 179, 181, 188
empty, 7
module, 244
theory-constitutive, 142
token, 121
top-level, 96
element
element, 202, 203, 206, 341, 346, 348, 361
element
attribute
on omstyle, 200
on use, 209, 351

elements
Euclid's, XI
EMACS, 246, 256, 314–316, 326
EMACS, 320
embed
attribute value
for show on omlet, 220
emitter, 311
empty
element, 7
namespace, 90
string, 156
en, 106, 130
en
attribute value
for xml:lang, 130
xml:en
attribute value
for lang, 41
encoding
UTF-8, 40
encoding
attribute
on m:annotation, 122
on om:OMFOREIGN, 117
endnote, 139
entailed-by
attribute
on alternative, 154
entailed-by-thm
attribute
on alternative, 154
entailment
system, 302
entails
attribute
on alternative, 154
entails-thm
attribute
on alternative, 154
entity
mnemonic, 10
module inclusion, 244
parameter, 44, 242, 243
XML, 8, 9, 241
entry
index, 139
enumeration
attribute value
for type on omgroup, 43, 100
equation
differential, 14, 29
recursive, 149
equivalence
class, 228
equivalent
attribute value
for status on assertion, 153
error
in-place (), 99, 117
mathematical, 99
operator, 117
semantic, 117
error
attribute value
for role on symbol, 117, 145, 345
escaping
XML, 8
Euclid, XI
Euclid's
elements, XI
Euclid's algorithm, IX
evaluation
semantics, 328
evidence
higher-level, 174
evidence
attribute value
for type on omtext, 133
evolutionary
process, 297
example, 74, 93
example
attribute value
for type on omtext, 42, 47, 133
example
element, 47, 50, 74–76, 155–157, 284, 341
execute
attribute value
for action on omlet, 220
exercise, 223
exercise
element, 223, 224, 341, 342
Exercises
RNC Module QUIZ, 384
rnc Module QUIZ, 370

spec Module QUIZ, 93, 223, 232, 234, 353–356, 358
exhaustivity, 185
exhaustivity
attribute
on definition, 149
on morphism, 185, 342
exincl.xsl, 250
existence
attribute
on definition, 148, 331
on morphism, 185
expansion, 174
tree, 231
experimental
attribute value
for cdstatus on theory, 162
explicit
namespace
prefix, 121
explosion
DAG, 118
export
symbol, 145
expres.xsl, 250
EXT (Extensions)
RNC Module, 382
rnc Module, 370
spec Module, 93, 130, 215, 232, 234, 354, 355, 357
extension
concept, 73
conservative, 143, 154
Extensions
RNC Module EXT, 382
rnc Module EXT, 370
spec Module EXT, 93, 130, 215, 232, 234, 354, 355, 357
external
mathematical
service, 278
object, 219
subset, 243
external
attribute value
for original on data, 217, 340

factual
knowledge, 25
false
conjecture, 157
false
attribute value
for verdict on answer, 224
false-conjecture
attribute value
for type on assertion, 151
for type on omtext, 134, 343
family
font, 91
FAQ, 237
feature, 116
symbol, 123
field
element, 283
figure
didactic, 79, 80
rhetoric/didactic, 79, 80
file
style, 4, 16
finite-state
cascades, 263
FireFox, 83, 261, 264, 284
first-order
classical (), 176
logic, 29, 49, 132, 293
theorem
prover, 83, 293
fixity
attribute
on presentation, use, 207
on use, 209
flatten, 101
FMP
element, 50, 76, 85, 99, 131–135, 146, 148, 153, 173, 174, 176, 218, 223, 224, 231, 256, 257, 339, 340, 348
omcd:FMP
element, 19, 20
font
family, 91
variant, 91
footnote, 139
footnote
attribute value
for type on note, 139
for
attribute value

for type on example, 47, 155
for
attribute, 364, 365
on alternative, 154
on assertion, 338
on axiom, 74, 146, 339
on decomposition, 195, 340, 348, 351
on definition, 45, 148, 331
on example, 47, 75, 155
on hint, 341
on insort, 166
on mc, 342
on note, 139
on omstyle, presentation, 200
on omtext, 133, 349
on path-just, 196, 343
on presentation, 205–207
on private, code, 216
on private, 350
on proof, 84, 172, 177, 332, 344
on solution, 223
on type, 147, 351
foreign
namespace, 90
form, 3
definitional, 23
formal
analysis, 327
calculus, 30
mathematical
document, 327
property, 19
mathematics, 30
parameter
theory, 62
representation, 327
semantics, 327
software
development, 273, 296, 297
system, 30
formalism
attribute
on legacy, 127
formalization, 301
format
content dictionary, 162
migration, 31
dc:format
element, 106, 110
format
attribute
on data, 77, 217, 347, 348, 350
on legacy, 127
on use, xslt, style, 201, 209
on use, 252
old attribute on CMP (deprecated in OMDoc 1.1), 347
formula, 113
mathematical, 14, 21
formula
attribute value
for type on assertion, 151
for type on omtext, 134, 343
formulae as
types, 21
forward
reasoning, 178
Four-Colour
Theorem, IX
fr, 106, 130
fragment, 243
code, 77, 130
document, 103
identifier, 10
frame, 91
framework
logical, 21, 293
free, 165
free
attribute value
for type on adt, 166
frequently
asked
questions, 237
from
attribute
on axiom-inclusion, 191
on imports, 45, 159
on omtext, 133, 349
on theory-inclusion, 189
frontend
human-oriented (), 256
frozen
attribute value
for action on dc:date, 105
function, 3, 14
partial, 167

predecessor, 60, 165, 166
recursive, 149, 165
successor, 165, 167
total, 167
Fundamental Theorem of Algebra, 14
Fundamentals of Computer Science, 71
future-proof, X

Gödel's Incompleteness Theorem, IX
gap
steps, 173
gap
attribute value
for type on derive, 173
generated, 165
generated
attribute value
for type on adt, 166
generated-by
old attribute on alternative (deprecated in OMDoc 1.1), 347
old attribute on assertion (deprecated in OMDoc 1.2), 347
old attribute on axiom (deprecated in OMDoc 1.2), 347
old attribute on definition (deprecated in OMDoc 1.2), 348
old attribute on symbol (deprecated in OMDoc 1.2), 350
generated-from
attribute, 191, 364
on alternative, 338
on assertion, 151
on axiom-inclusion, 339
on axiom, 146, 168
on decomposition, 340
on example, 341
on exercise, 341
on hint, 341
on mc, 342
on omdoc, 342
on omgroup, 342
on omstyle, 342
on omtext, 343
on private, 340, 344
on proofobject, 344
on proof, 344
on solution, 344
on theory-inclusion, 346
on type, 346
generated-via
attribute, 191
on adt, 339
on alternative, 338
on assertion, 338–340
on axiom-inclusion, 339
on decomposition, 340
on example, 341
on exercise, 341
on hint, 341
on mc, 342
on omdoc, 342
on omgroup, 342
on omstyle, 342
on omtext, 343
on private, 340, 344
on proofobject, 344
on proof, 344
on solution, 344
on symbol, 345
on theory-inclusion, 346
on type, 346
generator
model, 81
mq:generic
attribute, 269–271
global, 159, 190
link, 28, 297
theorem
link, 298
global
attribute value
for scope on symbol, 145, 167
for type on imports, 159, 191
globals
attribute
on axiom-inclusion, 195
on path-just, 195
glyph, 145
mathematical, 308
goal
learning, 281
GoTo, 307
grammar
context-free, 8, 241

normative (), 242
graph, 188
dependency, 307
development, 62, 192, 233, 296, 297, 299
theory, 188
graphics
computer, IX
greatest
common
divisor, IX
grounded, 176
group, 14
multi-logic, 131, 146, 223
multi-system, 146
multilingual, 41, 42, 130, 131, 135, 146, 155, 173, 218, 223, 230, 254
group, 144, 160, 161
group representation, 29

hash
character, 10
head
template, 204
header
table, 138
height
old attribute on omlet (deprecated in OMDoc 1.2), 347, 349, 350
HetCasl, 303
heterogeneous
specification, 301
Hets, 276, 300, 302–304
hiding
attribute
on morphism, 68, 186, 342
hierarchy
algebraic, 183
higher-level
evidence, 174
structure, 42
higher-order
logic, 132, 297
hint, 223
hint
element, 223, 224, 341
Hol, 23, 38
home
theory, 142
href
attribute
on data, 217, 340, 348, 365
on link, 139
on om:OMR, 118, 119
on OMR, 50
on ref, 368
old attribute on premise (deprecated in OMDoc 1.1), 349
xlink:href
attribute
on in MathML, 17
HTML
dynamic, 219
HTML, 5, 6, 10, 14, 15, 201–203, 206, 208, 209, 219, 254, 266, 284, 285, 287, 288
html
attribute value
for format on legacy, 127
for format on use, 201
HTTP, 81, 83
human-oriented
frontend
format, 256
hyperlink, 138
hyperref.sty, 208
hypertext
document, 5
Hypertext Markup Language, 5
hypothesis
inductive, 132, 173, 178
hypothesis
attribute value
for type on omtext, 133
hypothesis
element, 173, 177, 178, 341, 346, 348

icmap, 285
ICOM, 294, 295
ID
system, 238
type, 10, 27, 90, 91, 101, 337
ID, 9
dc:id
attribute
on xml:creator, 104
on xml:subject, 344
on xml:title, 345

id
  attribute, 91, 338
    on CMP, 347
    on decomposition, 348
    on FMP, 348
    on m:bvar, 126
    on om:*, 343
    on type, 351
    on with, 351
  attribute (in MathML), 17
  attribute (in OpenMath objects), 118, 119
xml:id
  attribute, 7, 10, 90, 91, 132, 173, 200, 224, 338, 363
    on assertion, 151
    on code, 221
    on derive, 172, 177
    on description, 340
    on effect, 340
    on idx, 139
    on imports, 196, 341
    on inclusion, 341
    on input, 341, 343
    on legacy, 127
    on measure, 342
    on obligation, 342
    on omdoc, 40, 54, 96
    on omgroup, 43
    on omtext, 42, 102, 133
    on ordering, 343
    on phrase, 135
    on private, code, 216
    on proof, 172
    on ref, 100, 344
    on symbol, 343
    on term, 136
    on theory, 158, 160, 162, 345
ID-type, 10
ide
  element, 139
Identifier
  public, 9
identifier, 121
  fragment, 10
  public, 40, 239, 242
  system, 239
dc:identifier
  element, 106
idp
  element, 139
IDREF
  type, 347
IDREF, 9
idt
  element, 139
idx
  element, 130, 139
ignore
  element, 99, 101, 130
IkeWiki, 289, 290
image/gif
  attribute value
    for format on data, 217
image/jpeg
  attribute value
    for format on data, 217
implicit
  axiom, 147
  definition, 143
  knowledge, 305
implicit
  attribute value
    for type on definition, 50, 148, 149
import
  local, 159
imported
  attribute value
    for action on dc:date, 105
imports
  element, 45, 46, 49, 62, 68, 159–161, 185–188, 191, 196, 297, 322, 341
Imps, 188, 322
in scope of
  theory, 175
in-place
  error
    markup, 99, 117
in-the-large, 300
  structuring, 274
  verification, 296, 300
in-the-small, 300
  verification, 297, 300
INCLUDE, 245
include
  attribute value
    for type on ref, 101

included
structurally, 187
inclusion
axiom, 190, 194
theory, 61, 63, 183, 187, 190, 191, 193, 194
inclusion
element, 62, 190, 341
incomplete
proof, 173
Incompleteness
Gödel's (), IX
inconsistent, 23, 142
index
entry, 139
markup, 139
phrase, 139
text, 139
index
attribute
on ide, 139
on in module RT, 139
on phrase, 135
individualized
document, 256
induced-by
attribute, 361
on obligation, 189
induction
challenge problems, 293, 294
theorem prover, 293, 294
theorem provers, 293, 294
inductive
definition, 144
hypothesis, 132, 173, 178
proof, 178
step, 178
inductive
attribute value
for type on definition, 149, 340
inductive
attribute
on assumption, 132, 339
on hypothesis, 173
inductively defined
set, 60
inference
derived (), 180
rule, 174
infix
attribute value
for fixity on presentation, 207, 331
infixl
attribute value
for fixity on presentation, 207, 331
infixr
attribute value
for fixity on presentation, 207, 331
informal
mathematics, 30
informal
attribute value
for type on definition, 150
information
retrieval, IX
set, 243
style, 200
infoset, 18
inherit, 185
inheritance, 233
relation, 160, 183
semantic (), 313
theory, 79
inherited, 159, 161
inherits
attribute
on metadata, 348
Initiative
Creative Commons, 98, 109
ink-on-paper, 2, 3
InKa, 253, 293, 297, 321, 323, 324
input, 77
input
element, 77, 218, 221, 341
insertion
set, 165
insort
element, 166
instance
schema, 246
theory, 60
institution, 302
integer, 117
integration
protocol-based, 321

theory, 322
type, 299
integrity condition, 154
intellectual
property, 108
intelligence
artificial, 301
interactive
document, 256
mathematical
document, 277
interactivity, 277
interface, 255
theory, 323
interlingua, 251, 321
internal
subset, 44, 243
DTD, 243
international
mathematics
community, X
internationalization, 254
Internet
publication, 2
MS Internet Explorer , 249
Internet Explorer, 16, 83, 249, 440
interpretation
theory, 183
interrelation
semantic, 273
intersection
set, 132
`introduction`
attribute value
for `type` on `omtext`, 133
intuitive
knowledge, 25
`inv`, 161
invariant
under a document model $\mathcal{X}$, 228
invasive
editor, 256
inverse
left, 191
right, 191
"is a" relation, 75
ISABELLE, 23, 175
ISBN, 106
ISO
639, 103
norm, 106, 130, 254
8601, 103
norm, 40, 105, 354, 355
isomorphism
Curry-Howard, 325
ISSN, 106
`itemize`
attribute value
for `type` on `omgroup`, 78, 100

JAVA, 295, 330
Java, 77, 218
applet, 221
JavaScript, 219
JAXP
DOM, 295
JEDIT, 287
JEDITOQMATH, 287, 288
`jurisdiction`
attribute
on `cc:license`, 109
`just-by`
attribute
on `assertion`, 76, 152, 153, 338, 347
on `type`, 147, 346
justification, 174

K-14, 15, 18, 120
mathematics, 122
Kepler's
Conjecture, IX
key, 116
`key`
attribute value
for `role` on `presentation`, 207, 349
`kind`
old attribute on `ref` (deprecated in OMDoc 1.2), 350
old attribute on `symbol` (deprecated in OMDoc 1.1), 345
Kindergarten, 15, 120
Knowledge
Mathematical (), X, 255
knowledge
base, 173, 301
factual, 25
implicit, 305

intuitive, 25
management, XII, 73
mathematical, IX, XI, 1
mathematical (), XI, 192, 256, 326
presentation, 73
representation, 73
knowledge-based
proof
planning, 25
knowledge-centered
document, 72, 254
view, 95
knowledge-structured, 95

`labeled-dataset`
attribute value
for `type` on `omgroup` (deprecated in OMDoc 1.2), 100, 342
λClam, 253, 321, 323
`dc:lang`
attribute
on `xml:*`, 107
on `xml:contributor`, 105
on `xml:description`, 105
on `xml:subject`, 105
on `xml:title`, 104
`xml:lang`
attribute, 7, 130
on `CMP`, 41, 130
on `description`, 45
on `use, xslt, style`, 201, 209
language
content, 321
markup, 2
natural, 129
style sheet, 6
`dc:language`
element, 106, 110
language definition
theory, 252
languages
multiple, 130, 254
`larg-group`
old attribute on `use` (deprecated in OMDoc 1.2), 346
LaTeX, 4
LaTeXML
binding, 311
LaTeXML, 311, 312
Latin, 129
laymen, 129
layout
box, 91
schema, 17
`lbrack`
attribute value
for `crossref-symbol` on `presentation`, 208, 209
`lbrack`
attribute
on `map`, 203
on `presentation, use`, 208, 209
on `presentation`, 331
Learner
Model, 281
learning, 281
goal, 281
left
inverse, 191
unit, 191
legacy, 126
`legacy`
element, 42, 113, 127, 130, 132, 174, 180, 222, 341, 372
lemma, 151
`lemma`
attribute value
for `type` on `assertion`, 151, 331
for `type` on `omtext`, 42, 134, 343
level
assertion, 326
lexical
document
model, 228
`li`
element, 138
library
document, 262
`libxml2`, 239
license
CC, 108
Creative Commons, 41, 108
`cc:license`
element, 108, 109
lightweight
mechanism, X
line-feed, 8
lingua franca, IX

link
  global, 28, 297
  local, 28, 297
  simple, 139
link
  element, 130, 138
links
  attribute
    on decomposition, 195
    on idp,ide, 139
    on idp, 139
LISP, 207
lisp
  attribute value
    for bracket-style on presentation, 207
list
  bulleted, 138
  description, 138
  ordered, 138
  semicolon-separated, 91
  unordered, 138
lists of
  character, 63
local, 159, 161
  assumption, 173
  chain, 193, 195
  declaration, 178
  import, 159
  link, 28, 297
  name, 90, 244
  theory
    inclusion, 190, 191
local
  attribute value
    for original on data, 217, 340
    for scope on symbol, 145, 167
    for type on imports, 62, 159, 191
local
  attribute
    on path-just, 195
locale (XSLT parameter), 254
localization, 254
logic
  description, 301
  first-order, 29, 49, 132, 293
  higher-order, 132, 297
  morphism, 325
  propositional, 180
  translation, 302
logic
  attribute
    on FMP, 131
logical
  calculus, 169
  framework, 21, 293
  system, 131
logically
  redundant, 233
loose, 165
  definition, 144
loose
  attribute value
    for type on adt, 166

m:, 121
m:annotation
  element, 122
m:annotation-xml
  element, 17, 122, 211
m:apply
  element, 16, 18, 122, 145, 207, 210, 252
m:bvar
  element, 18, 122, 126, 145, 207, 210
m:ci
  element, 16, 121, 126, 210, 252
m:cn
  element, 16, 121
m:csymbol
  element, 121–123, 126, 134
m:divide
  element, 16
m:math
  element, 113, 121, 127, 130, 132, 174, 180, 341, 342
m:mathml
  element, 222
m:mfence
  element, 15
m:mfrac
  element, 15
m:mi
  element, 15
m:mn
  element, 15
m:mo
  element, 15

`m:mover`
element, 15
`m:mroot`
element, 15
`m:mrow`
element, 15
`m:mstyle`
element, 15
`m:msub`
element, 15
`m:msubsup`
element, 15
`m:msup`
element, 15
`m:munder`
element, 15
`m:munderover`
element, 15
`m:plus`
element, 16
`m:semantics`
element, 17, 122, 123, 211
machine-readable, 18, 113
macro
semantic, 312
magma, 50
`Makefiles`, 250
management
change, 192, 233, 273, 275
configuration, 273
content, 72
correctness, 173
document, 6, 103
knowledge, XII, 73
rights, 98, 103
management of
change, 299
manager
document, 279
Mandelbrot
set, IX
`map`
element, 203, 341, 342, 344, 346
MAPLE™, 29, 263, 264, 321
markup
Content, 30
content, 4, 28, 30, 31
context, 22, 28
CSS, 219
document, 1
index, 139
language, 2
parallel, 17
presentation, 3
semantic, 30, 31, 305
semantic (), 30
`match`
attribute, 204
`math`
attribute value
for `bracket-style` on
`presentation`, 207
`m:math`
element, 113, 121, 127, 130, 132, 174, 180, 341, 342
`math`
element, 270
math-enabled
web
browser, 279
MATHDOX, 257, 277–280
Mathematica
notebook, 201
MATHEMATICA, 217
`mathematica`
attribute value
for `format` on `use`, 201
Mathematical
Knowledge
Management, X, 255
mathematical
assistant
system, 25
commented (), 20
concept, 144
context, 134
document, 2, 21, 326
dynamic (), 277
error, 99
external (), 278
formal (), 19, 327
formula, 14, 21
glyph, 308
interactive (), 277
knowledge, IX, XI, 1
base, 256
dissemination, XI
management, XI, 192

repository, 326
object, 14
practice, 326
proofs, 173
publication, 329
service, 81, 280, 321
software
system, 2, 216
software bus, 81
statement, 21, 22, 133, 134, 141, 191
symbol, 327
text, 105, 244, 327
text-editors, 326
theory, 22, 141, 297
verified (), 327
vernacular, 29, 38, 48, 104, 129, 133, 137, 138, 169
Mathematical Objects
RNC Module MOBJ, 372
spec Module MOBJ, 93, 96, 113, 130, 232, 234
Mathematical Statements
RNC Module ST, 375
rnc Module ST, 369
spec Module ST, 93, 141, 158, 232, 353–358
Mathematical Text
RNC Module MTXT, 372
spec Module MTXT, 93, 129, 130, 232, 234, 353–355, 357–359
MATHEMATICA®, 14, 29, 201, 256, 263, 264, 277, 314, 315, 318–321
mathematics, XII
formal, 30
informal, 30
international (), X
K-14, 122
MATHML
content, 10, 15–19, 21, 22, 27, 29, 31, 37, 42, 43, 113, 120–123, 125–127, 132, 134, 136, 162, 201, 207, 211, 249, 251, 252, 270, 312, 321, 323, 337, 342
presentation, 10, 15–17, 121, 122, 126, 201, 206, 209, 210, 241
MATHML, VIII, 1, 10, 14–19, 21, 28, 32, 90, 91, 93, 113, 116, 121–126, 162, 201, 206, 209, 210, 228, 241, 245, 254, 256, 258, 261–263, 268, 270, 271, 284, 285, 312, 318, 321, 372, 388, 390, 394, 444
MATHML, 209
`m:mathml`
element, 222
MATHPLAYER, 16
MATHWEB, XI, 81, 83, 237, 266, 295
MathWeb OMDOC, 234
MATHWEB-SB, 26, 81, 83
MATHWEB-WS, 81, 83, 85
MAXIMA, 263, 264, 284
MAYA, XIII, 256, 296–300, 303
MBASE, 37, 73, 83, 84, 256, 266, 267, 295, 298, 321
`mc`
element, 223, 224, 342
`measure`
element, 149, 150, 185, 342, 348
measure function, 149
mechanism
lightweight, X
membership
namespace, 7
mental
representation, 145
meta-data, 39
metadata, 40, 54, 98, 103
CC, 108
content dictionary, 162
`metadata`
element, 39, 43, 54, 77, 97, 98, 100, 103–106, 108, 110, 133, 146, 158, 217, 231, 305, 340, 341, 344, 347, 348, 350
method, 174
proof, 174, 188
`method`
element, 85, 173–175, 332, 348, 349
`m:mfence`
element, 15
`m:mfrac`
element, 15
`m:mi`
element, 15
Microsoft
Internet Explorer, 16, 83, 249, 440
Word, 3
migration
format, 31

MIME
type, 41, 106, 117, 217, 219, 220, 363
misconception
element, 283
MIZAR, 22, 23, 27, 38
MMiSS, 257, 273, 274, 276
m:mn
element, 15
mnemonic
entity, 10
m:mo
element, 15
MOBJ (Mathematical Objects)
RNC Module, 372
spec Module, 93, 96, 113, 130, 232, 234
mode
nXML, 246
Model
Learner, 281
model
document, 228
document object, 18
generator, 81
modularization, 244
module, 89
DTD, 244
element, 244
module
attribute
on omdoc, 44
module inclusion
entity, 244
modules
attribute, 89, 96, 97, 232
on omdoc, 40, 54, 97
on omgroup, 342
monograph, 22
monoid, 66, 146
monoid, 161
monomorphism
attribute value
for conservativity, 188
Moore's Law, IX
morphism, 184
base, 187
consequence, 297
logic, 325
theory, 183, 233
morphism
element, 61, 67, 68, 185, 187, 189, 342
motivation
attribute value
for type on omtext, 133
m:mover
element, 15
MoWGLI, 170
mOZART, 266
MOZILLA, 16, 249, 256, 261, 263–265, 284
MP3
recording, 108
mq:anyorder
attribute, 269–271
mq:generic
attribute, 269–271
m:mroot
element, 15
m:mrow
element, 15
MS
Internet Explorer, 16, 83, 249, 440
m:mstyle
element, 15
m:msub
element, 15
m:msubsup
element, 15
m:msup
element, 15
MTXT (Mathematical Text)
RNC Module, 372
spec Module, 93, 129, 130, 232, 234, 353–355, 357–359
multi-format
collection, 222
multi-logic
group, 131, 146, 223
multi-system
group, 146
multilingual, 135, 148
document, 254
documents, 130
group, 41, 42, 130, 131, 135, 146, 155, 173, 218, 223, 230, 254
parallel (), 135, 139
support, 130, 254
text, 233

multiple
languages, 130, 254
Multiple-choice exercise, 224
m:munder
element, 15
m:munderover
element, 15

omcd:Name
element, 115
name
local, 90, 244
qualified, 7, 8, 200, 202, 244
simple, 8, 9
name
attribute
on attribute, 203
on constructor, 167
on definition, 45
on element, 203
on om:OMS, 31, 114
on om:OMV, om:OMS, 125
on om:OMV, 115
on OMS, 18, 45
on OMV, 19, 45
on param, 222
on recognizer, 167
on selector, 167
on sortdef, 166, 350
on symbol, 45, 145, 146, 167, 331
on term, 136, 345
named
anchor, 10
namespace, 7, 130, 200, 202
abbreviation, 8, 245
Creative Commons, 40, 109
Creative Commons (), 109
declaration, 7, 8, 40, 90
default, 90
Dublin Core, 40, 104
Dublin Core (), 104
empty, 90
explicit (), 121
foreign, 90
membership, 7
OMDoc, 40, 89
OMDoc (), 89
OpenMath, 40, 114
OpenMath (), 114
prefix, 90, 104, 109, 114, 121
declaration, 40, 90
prefixed, 44
URI, 96, 244
XML, 7, 8, 244
namespace prefix
declaration, 44
namespace-aware, 121
narrative, 133, 230
OMDoc, 72
structure, 256
narrative
attribute value
for type on omgroup, 349
narrative
attribute value
for type on omgroup (deprecated in OMDoc 1.2), 101
narrative-centered, 101
view, 95
narrative-structured, 95, 230, 231
document, 72, 254
natural
deduction, 180, 348
calculus, 85
proof, 177
style, 177
language, 129
number, 165
positive (), 166
natural language
analysis, 327
navigation, 308
content, 307
NB2OMDOC, 318
neut, 161
new
symbol, 143
new
attribute value
for show on omlet, 220
nl, 106, 130
no
attribute value
for crossref-symbol on presentation, 208
for inductive on hypothesis, 173
for total on selector, 167, 350
no-consequence

attribute value
for status on assertion, 153
node
attribute, 7
text, 7
non-applied
occurrence, 212
normal form
ref, 101
normalization
DTD, 243
URI, 160
normalized, 241
normative
grammar
formalism, 242
precedence, 9
normed
attribute value
for action on dc:date, 105
notation, 2
Backus Naur form, 97
Polish, 114
prefix, 114
note, 139
editorial, 308
note
attribute value
for type on omtext, 133, 402
note
element, 130, 139
notebook
Mathematica, 201
NotebookML, 318
notice
attribute
on cc:requirements, 110
NqThm, 293, 294
ns
attribute
on attribute, 203, 339
on element, 203, 341
NS.prefixed, 245
number, 121
natural, 165
NuPrL, 23, 38, 175
nXML
mode, 246
o:, 245
object
OpenMath, 18, 114
binding, 145
external, 219
mathematical, 14
proof, 180
symbol, 252
object
attribute value
for role on selector, 344
for role on symbol, 145, 167
for type on constructor, 340
for type on recognizer, 344
for valuetype on param, 222
xhtml:object
element, 219, 220, 342
object model
document, 219, 227
object-oriented
programming, 72
obligation
proof, 188, 296–298
obligation
attribute value
for type on assertion, 151
for type on omtext, 134, 343
obligation
element, 61, 189, 191, 193, 196, 231, 342, 346–348
obsolete
attribute value
for cdstatus on theory, 162
occurrence
bound, 126
defining, 125, 210
non-applied, 212
official
attribute value
for cdstatus on theory, 162
ol
element, 130, 138
om:, 40, 114
om:*
element, 343
om:OMA
element, 18, 45, 114, 115, 118–120, 207, 252
om:OMATP

element, 116, 117
om:OMATTR
element, 116–118, 122, 123, 125, 145, 207, 210
om:OMB
element, 117
om:OMBIND
element, 18, 114–116, 118, 125, 145, 204, 207, 210, 252
om:OMBVAR
element, 18, 114, 116, 125, 203, 204, 210
om:OME
element, 117
om:OMF
element, 117
om:OMFOREIGN
element, 117, 211, 343
om:OMI
element, 117
om:OMOBJ
element, 7, 18, 45, 49, 50, 113, 114, 118, 127, 130, 132, 174, 180, 222, 230, 295, 341, 348, 349
om:OMR
element, 50, 118–120, 231, 343
om:OMS
element, 7, 18, 45, 114–116, 125, 136, 158, 162, 232
om:OMSTR
element, 117, 348
om:OMV
element, 18, 19, 45, 114, 115, 125, 252
om:OMA
element, 18, 45, 114, 115, 118–120, 207, 252
om:OMATP
element, 116, 117
om:OMATTR
element, 116–118, 122, 123, 125, 145, 207, 210
om:OMB
element, 117
om:OMBIND
element, 18, 114–116, 118, 125, 145, 204, 207, 210, 252
om:OMBVAR
element, 18, 114, 116, 125, 203, 204, 210
omcd:CDDefinition
element, 115
omcd:CDName
element, 54
omcd:CDURL
element, 54
omcd:CMP
element, 20
omcd:FMP
element, 19, 20
omcd:Name
element, 115
OMDoc
content, 72
content dictionary, 44, 162, 232
namespace, 40, 89
URI, 89
narrative, 72
OMDoc
version 1.0, 118
version 1.1, 27, 91, 100, 101, 118, 170, 249, 337, 338
version 1.2, 27, 28, 40, 100, 104, 118, 244, 337, 338, 342
version 1, VIII, 27, 337
OMDoc, V, VIII, X–XIII, 1–3, 6, 7, 9, 10, 14, 18, 20, 23–35, 37, 39–45, 47, 50, 53, 54, 56, 60–62, 67, 71–81, 83–85, 87, 89–93, 96–110, 113, 114, 116, 117, 121–123, 126, 127, 129, 130, 132–139, 141, 142, 144–148, 150–152, 154–156, 158–162, 165–167, 170–176, 179–181, 183–187, 189–192, 195, 196, 199–207, 210, 212, 215–220, 222–225, 227–235, 237–239, 241–246, 249–263, 265, 266, 276, 278, 281–298, 301, 303–305, 307, 308, 310, 311, 313–316, 318, 320, 321, 323–325, 329–333, 337, 338, 340, 343, 344, 346, 348–350, 353, 361, 365, 367–373, 375, 376, 378, 383, 387
omdoc
element, 9, 41, 54, 89, 90, 96–98, 100, 101, 105, 110, 232, 246, 291, 342
omdoc-basic, 44
omdoc.cat, 239
omdoc.prefix, 245

omdoc1.1adapt1.2.xsl, 249
omdoc2html.xsl, 254
omdoc2pvs.xsl, 252
omdoc2share.xsl, 254
omdoc2sys.xsl, 252
omdoc2tex.xsl, 254
omdoc:, 245
OMDOM, 227
om:OME
  element, 117
ΩMEGA, 326
om:OMF
  element, 117
om:OMFOREIGN
  element, 117, 211, 343
omgroup
  element, 43, 77, 78, 96, 97, 100, 101, 158, 232, 305, 342, 349
om:OMI
  element, 117
omlet
  element, 77, 130, 219–222, 244, 342, 347, 349, 350, 361, 363
om:OMOBJ
  element, 7, 18, 45, 49, 50, 113, 114, 118, 127, 130, 132, 174, 180, 222, 230, 295, 341, 348, 349
om:OMR
  element, 50, 118–120, 231, 343
om:OMS
  element, 7, 18, 45, 114–116, 125, 136, 158, 162, 232
om:OMSTR
  element, 117, 348
omstyle
  element, 200–203, 205, 206, 231, 250, 342, 349–351, 363
omtext
  element, 7, 41–43, 46, 47, 77, 91, 99, 133, 134, 157, 172, 230, 232, 281, 342, 343, 349
om:OMV
  element, 18, 19, 45, 114, 115, 125, 252
onLoad
  attribute value
    for action on omlet, 220
onPresent
  attribute value
    for action on omlet, 220
onRequest
  attribute value
    for action on omlet, 220
ontologically
  promiscuous, 325
ontology, 273
op, 161, 230
Open eBook, 107
opening
  tag, 7
OpenMath
  content dictionaries
    a, 294
  namespace, 40, 114
    URI, 114
OpenMath
  object, 18
OpenMath, VIII, 1, 7, 8, 10, 14, 15, 17–22, 26–31, 34, 37, 40, 42–46, 50, 53, 54, 56, 59, 90, 91, 93, 95, 98, 99, 113–127, 130–132, 136, 145, 162, 163, 204, 209–212, 228, 232, 233, 245, 249, 251–253, 258, 263, 264, 268, 270, 277, 278, 280, 284–287, 294, 295, 315, 319–321, 323, 324, 331, 332, 342, 343, 345, 349, 361, 362, 372, 387, 415, 416
OpenMath elements
  extra attributes id and xref, 118
OpenOffice, 265
operator
  binding, 116, 117
  error, 117
OQMath, 258, 287, 288
ordered
  list, 138
ordering, 144, 149
ordering
  element, 149, 185, 342, 343, 348, 349
original
  attribute
    on data, 217, 340
other
  attribute value
    for action on omlet, 220
    for show on omlet, 220
outfix
  attribute value
    for fixity on presentation, 331

output, 77
`output`
  element, 77, 218, 343
Owl, 291
Oz, 266

`p`
  element, 130, 138
packing
  sphere, IX
padding, 91
`swim:page`
  element, 291
pair, 46
  attribute-value, 203
paragraph, 100, 138
parallel
  markup, 17
  multilingual
    markup, 135, 139
`param`
  element, 219, 221, 222
parameter, 60, 174, 254
  XSLT, 254
  actual (), 62
  entity, 44, 242, 243
  formal (), 62
`parameter`
  element, 348
parameter entity, 245
  reference, 243
`parameters`
  attribute
    on `adt`, 166, 332, 339
parametric
  theory, 60
`parent`
  old attribute on `presentation`
    (deprecated in OMDoc 1.2), 349
parsed
  character data, 97
parser, 77
  validating, 8, 241
  XML, 8, 98, 252
partial
  function, 167
`partial`
  attribute value, 350
`path-just`
  element, 195, 196, 339, 343
pattern, 149
`pattern`
  attribute value
    for `type` on `definition`, 150, 340
    for `type` on `morphism`, 185
`pattern`
  element, 349
PDF, 4, 284
Peano
  axioms, 142, 143, 165
permission, 41
permissions, 109
`cc:permissions`
  element, 41, 109
`permitted`
  attribute value, 109
persistent
  comment, 99
PF (Proofs and Arguments)
  DTD Module, 244
  RNC Module, 378
  rnc Module, 370
  spec Module, 93, 169, 170, 232,
    354–357
phrase, 134, 135
  index, 139
`phrase`
  element, 92, 130, 132, 134, 135, 157,
    201, 203, 339, 340, 343, 346
physical
  representation, 145
pickling, 266
picture, 108
pixel-on-screen, 3
plug-in, 219, 220
`plus`, 29
`m:plus`
  element, 16
`pmml`
  attribute value
    for `format` on `legacy`, 127
    for `format` on `use`, 201
pointer, 125, 126
pointing
  semantics by, 31
Polish
  notation, 114
polynomial, 29

positioning, 91
positive
  natural
    number, 166
postfix
  attribute value
    for fixity on presentation, 207, 331
PostScript, 4
postulate
  attribute value
    for type on assertion, 151
    for type on omtext, 134, 343
postulated
  theory
    inclusion, 188
PowerPoint
  Content in , 305
PowerPoint, 314
MS PowerPoint, 305–308
practice
  mathematical, 326
precedence
  normative, 9
precedence
  attribute
    on map, 203
    on presentation, 207
predecessor
  function, 60, 165, 166
predefined
  symbol, 252
predicate, 167
  recognizer, 167
prefix
  namespace, 90, 104, 109, 114, 121
  namespace (), 40, 90
  notation, 114
prefix
  attribute value
    for fixity on presentation, 207, 331
prefixed
  namespace, 44
preloading
  semantic, 312
premise
  element, 76, 85, 174–176, 349
preparation language
  document, 256
PRES (Presentation)
  RNC Module, 383
  rnc Module, 369
  spec Module, 93, 126, 199, 232, 233, 353, 355, 357–359
Presentation
  Component, 281
  RNC Module PRES, 383
  rnc Module PRES, 369
  spec Module PRES, 93, 126, 199, 232, 233, 353, 355, 357–359
presentation, 95, 254, 305
  knowledge, 73
  markup, 3
  proof, 173, 257
  proof (), 180, 216
  semantics-induced, 307
  slides, 72
presentation
  element, 200, 205, 206, 208–210, 231, 249, 250, 252, 253, 331, 333, 343, 349–351, 367
Presentation MathML, 10, 15–17, 121, 122, 126, 201, 206, 209, 210, 241
P.Rex, 326
PRIMITIVE
  SYMBOL, 73, 74, 148
PRINCIPAL
  TYPE, 147
PRINCIPLE OF
  CONSERVATIVE
    EXTENSION, 23
PRIORITY-UNION, 110
PRIVATE
  ATTRIBUTE VALUE
    FOR CDSTATUS ON THEORY, 162
PRIVATE
  ELEMENT, 73, 174, 216–218, 220, 222, 231, 340, 344, 346, 347, 349, 350, 363, 366
PROBLEM, 151
PROCEDURE
  DECISION, 25
PROCESS
  CALCULUS, 301
  EVOLUTIONARY, 297
  REASONING, 123, 147

processing instruction
  style sheet, 92
processor
  Xml, 6
  XML, 239
product
  Cartesian, 45
program, 216
programming
  C (), 18
  construct, 278
  object-oriented, 72
PROHIBITED
  attribute value, 109
prohibitions, 109
CC:PROHIBITIONS
  element, 41, 109
Prolog, 207
promiscuous
  ontologically, 325
proof, 141, 169, 188, 321
  assistant, 326
  automated (), 81
  bottom-up (), 178
  checking, 30
  development
    environment, 25
  incomplete, 173
  inductive, 178
  knowledge-based (), 25
  method, 174, 188
  object, 180
  obligation, 188, 296–298
  presentation, 173, 257
    system, 180, 216
  sequent, 176
  top-down (), 178
  verification, 30
PROOF
  attribute value
    for TYPE on OMTEXT, 133
PROOF
  element, 76, 84, 171, 173–179, 181, 195, 332, 333, 341, 344, 350
proof assistance
  system, 326
proof-theoretic, 324
PROOFOBJECT
  element, 174, 175, 180, 181, 344, 350
proofs
  mathematical, 173
Proofs and Arguments
  DTD Module PF, 244
  RNC Module PF, 378
  rnc Module PF, 370
  spec Module PF, 93, 169, 170, 232, 354–357
property
  constitutive, 20
  CSS, 102
  intellectual, 108
PROPOSITION
  attribute value
    for TYPE on ASSERTION, 151
    for TYPE on OMTEXT, 134, 343
propositional
  logic, 180
protocol-based
  integration, 321
WS:PROVE
  element, 83
prover
  theorem, 216, 217
PTO
  attribute
    on DATA, 217, 340, 344
PTO-VERSION
  attribute
    on DATA, 217, 340, 344
public
  Identifier, 9
  identifier, 40, 239, 242
publication, 326
  Internet, 2
  mathematical, 329
DC:PUBLISHER
  element, 105, 340
Pvs, 59, 252, 253, 321–324

QMath, 256, 258–261, 263–265, 287, 292
QMATH
  attribute value
    for FORMAT on LEGACY, 127
qualified
  name, 7, 8, 200, 202, 244

QUIZ (Exercises)
RNC Module, 384
rnc Module, 370
spec Module, 93, 223, 232, 234, 353–356, 358

RANK
ATTRIBUTE
ON PREMISE, 349
RARG-GROUP
OLD ATTRIBUTE ON USE (DEPRECATED IN OMDoc 1.2), 346
RBRACK
ATTRIBUTE VALUE
FOR CROSSREF-SYMBOL ON PRESENTATION, 208, 209
RBRACK
ATTRIBUTE
ON MAP, 203
ON PRESENTATION, USE, 208, 209
ON PRESENTATION, 331
RDF, 80, 98, 110, 262, 289, 291
REASONING
BACKWARD, 178
FORWARD, 178
PROCESS, 123, 147
SYSTEM, 81
RECOGNIZER
PREDICATE, 167
RECOGNIZER
ELEMENT, 167, 344, 347, 350
RECOMMENDATION, 6, 14
Record
Bibliographic (), 262
RECORDING
MP3, 108
RECURSE
ELEMENT, 202, 203, 346, 350
RECURSIVE
EQUATION, 149
FUNCTION, 149, 165
RECURSIVE
ATTRIBUTE VALUE
FOR TYPE ON MORPHISM, 185
REDUCIBLE
REF, 101
REDUCTION
REF, 101
REDUNDANT
LOGICALLY, 233
REF
ATTRIBUTE VALUE
FOR VALUETYPE ON PARAM, 222
REF
ELEMENT, 78, 80, 100–102, 130, 132, 231, 250, 344, 348, 350, 368
REF-NORMAL FORM, 101
REF-REDUCIBLE, 101
REF-REDUCTION, 101
REF-TARGET
REF-TARGET, 101
REF-TARGET, 101
REF-VALID, 101
REFERENCE
PARAMETER ENTITY, 243
URI, 10, 45, 74, 96, 100, 121, 127, 139, 148, 152, 159, 187, 195, 200, 201, 216, 217, 220, 222
REFERENCING, 90
REFINEMENT
CONTROLLED, 256
REFORMULATES
ATTRIBUTE
ON PRIVATE, 73, 217, 344
RELATION, 45
"IS A", 75
CONSEQUENCE, 30
INHERITANCE, 160, 183
DC:RELATION
ELEMENT, 106
RELATIVE
URI, 160
RelaxNG, 8, 9, 238, 241, 242, 245, 246, 369–372, 387, 388
RENAMING
VARIABLE, 116, 125
RENUMBERING, 4
REPLACE
ATTRIBUTE VALUE
FOR SHOW ON OMLET, 220
REPLACES
OLD ATTRIBUTE ON PRIVATE (DEPRECATED IN OMDoc 1.2), 344, 350
REPORT-ERRORS, 250
REPRESENTATION

formal, 327
knowledge, 73
mental, 145
physical, 145
static, 328
theorem, 22
representational
rigor, XI
reproduction, 41
REPRODUCTION
attribute
on CC:PERMISSIONS, 109
REQUATION
element, 149, 150, 185, 231, 342
request
computation, 321
requirement
specification, 274, 296
requirements, 109
CC:REQUIREMENTS
element, 41, 109
REQUIRES
attribute
on CODE, 77
on PRIVATE, CODE, 216
on USE, XSLT, STYLE, 201, 209
resource
uniform (), 5, 7
resource description format, 98
Resource Description Framework, 98
retrieval
document, 37
information, IX
reuse
component, 296
document, 273
theory, 23
review
date, 163
REVIEW-ON
attribute value
for ACTION on DC:DATE, 105
revision, 163
rhetoric
role, 133
rhetoric/didactic
figure, 79, 80
Rich Text Structure
RNC Module RT, 381
spec Module RT, 93, 100, 129, 130, 137–139, 232, 234, 342, 354–359, 364, 368
right
inverse, 191
unit, 191
rights
Digital (), 108
management, 98, 103
DC:RIGHTS
element, 106, 108, 110
rigor
representational, XI
rigorous, 129
RNC, 379, 380
RNC Module
ADT (Abstract Data Types), 378
CTH (Complex Theories), 379
DC (Dublin Core Metadata), 374
DG (Development Graphs), 380
DOC (Document Structure), 373
EXT (Extensions), 382
MOBJ (Mathematical Objects), 372
MTXT (Mathematical Text), 372
PF (Proofs and Arguments), 378
PRES (Presentation), 383
QUIZ (Exercises), 384
RT (Rich Text Structure), 381
ST (Mathematical Statements), 375
rnc Module
EXT (Extensions), 370
PF (Proofs and Arguments), 370
PRES (Presentation), 369
QUIZ (Exercises), 370
ST (Mathematical Statements), 369
role, 116
rhetoric, 133
ROLE
attribute, 104, 348
on CONSTRUCTOR, 340
on DC:*, 107
on PRESENTATION, 206, 349
on RECOGNIZER, 344

ON SELECTOR, 344
ON SORTDEF, 344
ON SYMBOL, 116, 145, 167, 345
ON TERM, 45, 136
ROOT
DOCUMENT, 7, 9, 89, 105, 232
ROW
TABLE, 138
RRL, 293
RT (RICH TEXT STRUCTURE)
RNC MODULE, 381
SPEC MODULE, 93, 100, 129, 130, 137–139, 232, 234, 342, 354–359, 364, 368
RULE
INFERENCE, 174
RULE
ATTRIBUTE VALUE
FOR TYPE ON OMTEXT, 134

SATISFIABLE
ATTRIBUTE VALUE
FOR STATUS ON ASSERTION, 153
SAXON, 249
SCHEMA, 96, 238, 241, 246
INSTANCE, 246
LAYOUT, 17
XML, 8, 41, 91, 227, 241, 369
XSI:SCHEMALOCATION
ATTRIBUTE
ON OMDOC, 246
SCHEME, 326
SCHEME
ATTRIBUTE
ON DC:IDENTIFIER, 106
SCIENCE, XII
COMPUTER, 71
SCIENTIFIC
ANTECEDENT, 107
BACKGROUND, 2
EDITOR, 326
SCOPE, 175
SCOPE
ATTRIBUTE
ON SYMBOL, 145, 167
SEARCH
CONTENT, 307
SEARCHING, 261
SECTION, 22, 100
CDATA, 8
SECTIONING
ATTRIBUTE VALUE
FOR TYPE ON OMGROUP, 100
SEE
ATTRIBUTE
ON IDP,IDE, 139
SEEALSO
ATTRIBUTE
ON IDP,IDE, 139
SELECT
ATTRIBUTE
ON ATTRIBUTE, 203, 339
ON MAP, 203
ON RECURSE, 203
ON VALUE-OF, 203
SELECTOR
SYMBOL, 165–167
SELECTOR
ELEMENT, 121, 167, 344, 345, 350
SELFINCLUSION
THEORY, 66, 68
SEMANTIC
WEB, 301
SEMANTIC
ANNOTATION, 328
AUTHORING
TOOL, 306
CONTENT, 327
DOCUMENT
FORMAT, 328
ERROR, 117
INHERITANCE
RELATION, 313
INTERRELATION, 273
MACRO, 312
MARKUP, 30, 31, 305
FORMAT, 30
PRELOADING, 312
SEMANTIC MEDIAWIKI, 290
SEMANTIC-ATTRIBUTION
ATTRIBUTE VALUE
FOR ROLE ON SYMBOL, 145, 345
SEMANTICALLY
ANNOTATED
DOCUMENTS, 328
SEMANTICS
BY POINTING, 19
EVALUATION, 328

formal, 327
M:SEMANTICS
element, 17, 122, 123, 211
semantics by
pointing, 31
semantics-induced
presentation, 307
semicolon-separated
list, 91
semigroup, 66
Abelian, 161
SEMIGROUP, 161
Sentido, 256, 260–265
SEPARATOR
attribute value
for CROSSREF-SYMBOL on PRESENTATION, 208, 209
SEPARATOR
element, 203, 341
SEPARATOR
attribute
on PRESENTATION, USE, 208
SEQUENCE
attribute value
for TYPE on OMGROUP, 100, 101, 349
sequent, 132, 348
proof, 176
style, 176
proof, 177
server
document, 279
service
description, 321
mathematical, 81, 280, 321
web, 6
set
inductively defined, 60
information, 243
insertion, 165
intersection, 132
Mandelbrot, IX
theory, 321
SET, 161
SETNAME1, 123
SGML, 6, 242
shared
component, 296
sharing
document, 273
structure, 118
shorthand
xpointer, 10
SHOW
attribute, 220
on OMLET, 220
side-effect, 77
Siette, 283
simple
definition, 143
link, 139
name, 8, 9
SIMPLE
attribute value
for TYPE on DEFINITION, 49, 148, 150
Simple Generalized Markup Language, 6
SIZE
attribute
on DATA, 217, 348
slicing, 42
slide, 72
SLIDE
attribute value
for TYPE on OMGROUP, 77
slide presentation, 72
Soap, 81–84
software
formal (), 273, 296, 297
mathematical (), 2, 216
software bus
mathematical, 81
solution, 223
SOLUTION
element, 223, 224, 231, 344
solver
constraint, 81
sort, 60, 147, 165, 166
symbol, 166
SORT
attribute value
for ROLE on SELECTOR, 344
for ROLE on SYMBOL, 145, 167
SORT
attribute
on ARGUMENT, 347

old attribute on ARGUMENT (deprecated in OMDoc 1.2), 338
SORT-BY
attribute
on IDP, 139
SORTDEF
element, 60, 166, 167, 231, 339, 344, 346, 350
source, 4
comment, 98
document, 4
theory, 61, 159, 185, 187, 191
DC:SOURCE
element, 54, 106, 110
Spass, 81–85
spec Module
ADT (Abstract Data Types), 93, 155, 165, 166, 232, 233, 353–355, 358
CC (Creative Commons Metadata), 93, 103, 108, 355, 357, 358
CTH (Complex Theories), 93, 183, 185, 187, 191, 192, 232, 233, 296, 303, 353, 355, 356, 358
DC (Dublin Core Metadata), 93, 103, 104, 232, 234, 354–359
DG (Development Graphs), 93, 183, 192, 195, 232, 303, 354, 357
DOC (Document Structure), 89, 93, 95, 130, 232, 234, 355–358
EXT (Extensions), 93, 130, 215, 232, 234, 354, 355, 357
MOBJ (Mathematical Objects), 93, 96, 113, 130, 232, 234
MTXT (Mathematical Text), 93, 129, 130, 232, 234, 353–355, 357–359
PF (Proofs and Arguments), 93, 169, 170, 232, 354–357
PRES (Presentation), 93, 126, 199, 232, 233, 353, 355, 357–359
QUIZ (Exercises), 93, 223, 232, 234, 353–356, 358
RT (Rich Text Structure), 93, 100, 129, 130, 137–139, 232, 234, 342, 354–359, 364, 368
ST (Mathematical Statements), 93, 141, 158, 232, 353–358
specification, 59, 233
algebraic, 23, 27, 60, 337
heterogeneous, 301
requirement, 274, 296
system, 296
specifications
structured, 296
sped Module
DG (Development Graphs), 296
sphere
packing, IX
ST (Mathematical Statements)
RNC Module, 375
rnc Module, 369
spec Module, 93, 141, 158, 232, 353–358
STANDALONE
attribute
on ?XML, 243
standard
communication, 256
start
tag, 40, 44
state, 278
statement, 137
mathematical, 21, 22, 133, 134, 141, 191
static
context, 278, 279
manager, 280
representation, 328
status
content dictionary, 162
STATUS
attribute
on ASSERTION, 152, 153, 338, 366
STEM, XII
step
inductive, 178
steps
gap, 173
strict, 165
string, 117
empty, 156

strings, 156
  concatenation, 156
structural
  convention, 2
  theory
    inclusion, 188
structurally
  included, 187
structure
  abstract, 43
  conceptual, 256
  content, 134
  discourse, 256
  document, 233, 275
  higher-level, 42
  narrative, 256
  sharing, 118
  XML (), 228
structured
  document, 276
  specifications, 296
structuring
  in-the-large, 274
sts, 123, 124
style
  file, 4, 16
  information, 200
  sequent, 176
  sequent (), 177
style
  element, 201, 202, 204, 205, 209, 210, 344, 350, 351
style
  attribute, 91, 92, 96, 100, 139, 224, 338
    on definition, 158
    on omlet, 219
    on omtext, 91
    on phrase, 135
    on ref, 78, 102
    on with, 351
  attribute (meaning changed in 1.2), 338
  attribute (new meaning 1.2), 338
Style Sheet
  Cascading, 91
style sheet, 4, 6, 199, 238, 249
  cascading, 338
  CSS, 91, 238
  language, 6
  processing instruction, 92
  XSL, 238
Subversion, 237
dc:subject
  element, 105, 146, 339, 340, 344, 345
subset
  external, 243
  internal, 44, 243
  internal (), 243
successor
  function, 165, 167
support
  multilingual, 130, 254
svg, 284
svn, 237
SWiM, 289, 290, 292
swim:page
  element, 291
symbol, 18, 114, 121, 142, 144, 232
  bullet, 100
  constructor, 165, 166
  declaration, 22, 147, 162
  defined, 148
  export, 145
  feature, 123
  mathematical, 327
  new, 143
  object, 252
  predefined, 252
  primitive, 73, 74, 148
  selector, 165–167
  sort, 166
symbol
  element, 45, 74, 116, 144–147, 152, 155, 161, 168, 173, 174, 204, 231, 252, 253, 331, 333, 339, 341, 342, 345, 350, 351
syntax
  abstract, 43
  XML (), 228
system
  algebra, 29
  axiom, 143
  clipboard, 261
  computer algebra, 19, 25, 29, 81, 117, 216, 217, 252, 321
  entailment, 302

FORMAL, 30
ID, 238
IDENTIFIER, 239
LOGICAL, 131
PROOF ASSISTANCE, 326
REASONING, 81
SPECIFICATION, 296
TYPE, 147
VISUALIZATION, 321
SYSTEM
ATTRIBUTE
ON TYPE, 147, 346

TABLE, 138
DATA, 138
HEADER, 138
ROW, 138
TABLE
ELEMENT, 130, 138
TAG
CLOSING, 7, 41
OPENING, 7
START, 40, 44
TARGET, 4, 101, 119
REF, 101
DOCUMENT, 4
THEORY, 61, 159, 185, 187, 191
TARGETLANGUAGE, 254
TAUTOLOGOUS-CONCLUSION
ATTRIBUTE VALUE
FOR STATUS ON ASSERTION, 153
TAUTOLOGY
ATTRIBUTE VALUE
FOR STATUS ON ASSERTION, 153
TAXONOMY, 72, 77
TD
ELEMENT, 138
TEACHING, X
TECHNICAL
TERM, 135
TECHNOLOGY, XII
TEMPLATE, 199
XSLT, 250
HEAD, 204
TERM
CONSTRUCTOR, 165
DECLARATION, 147, 346
TECHNICAL, 135
TERM
ELEMENT, 45, 130, 134, 136, 345
TERMINATING
ATTRIBUTE
ON MEASURE, 149
ON ORDERING, 343
TERMINATION, 144
TEX
ATTRIBUTE VALUE
FOR FORMAT ON LEGACY, 127
FOR FORMAT ON USE, 201
TEX, 4, 14
TEXMACS, 326
TEXPOINT, 308
TEXT AS DUBLIN CORE TYPE, 105
TEXT, 40, 105, 106, 231
TEXT
COLOR, 91
INDEX, 139
MATHEMATICAL, 105, 244, 327
MULTILINGUAL, 233
NODE, 7
TEXT
ELEMENT, 202, 203, 346, 350
TEXT-EDITORS
MATHEMATICAL, 326
TEXT/PLAIN
ATTRIBUTE VALUE
FOR FORMAT ON DATA, 217
TEXTBOOK, 22
TGROUP
ELEMENT, 158, 345
TH
ELEMENT, 138
THEOREM
FOUR-COLOUR, IX
THEOREM, 23, 93, 137, 141, 151, 187, 188
AUTOMATED (), 25, 81, 321
FIRST-ORDER (), 83, 293
GLOBAL (), 298
PROVER, 216, 217
REPRESENTATION, 22
THEOREM
ATTRIBUTE VALUE
FOR STATUS ON ASSERTION, 153
FOR TYPE ON ASSERTION, 151
FOR TYPE ON ATTRIBUTE, 157
FOR TYPE ON OMTEXT, 42, 134, 343
THEOREM PROVER, 293

THEOREM LINK, 297
THEOREM PROVER, 29
  INDUCTION, 293, 294
THEOREM PROVERS
  INDUCTION, 293, 294
THEOREMA, 265
THEORY, 59, 72, 151, 297
  DEFINITIONAL (), 188
  DEVELOPMENT
    SYSTEM, 216
  GRAPH, 188
  HOME, 142
  IN SCOPE OF, 175
  INCLUSION, 61, 63, 183, 187, 190, 191, 193, 194
  INHERITANCE, 79
  INSTANCE, 60
  INTEGRATION, 322
  INTERFACE, 323
  INTERPRETATION, 183
  LANGUAGE DEFINITION, 252
  LOCAL (), 190, 191
  MATHEMATICAL, 22, 141, 297
  MORPHISM, 183, 233
  PARAMETRIC, 60
  POSTULATED (), 188
  REUSE, 23
  SELFINCLUSION, 66, 68
  SET, 321
  SOURCE, 61, 159, 185, 187, 191
  STRUCTURAL (), 188
  TARGET, 61, 159, 185, 187, 191
THEORY
  ELEMENT, 44, 45, 54, 101, 142, 152, 154, 158–163, 166, 190, 252, 291, 313, 331, 345, 346, 349, 350, 364
THEORY
  ATTRIBUTE
    ON ALTERNATIVE, 154, 338
    ON ASSERTION, 83, 151
    ON AXIOM-INCLUSION, 339
    ON DECOMPOSITION, 340
    ON EXAMPLE, 341
    ON EXERCISE, 341
    ON HINT, 341
    ON MC, 342
    ON OMDOC, 342
    ON OMGROUP, 158, 342
    ON OMTEXT, 343
    ON PRESENTATION, 349
    ON PRIVATE, CODE, 216
    ON PRIVATE, 340, 344
    ON PROOF, 172, 350
    ON SOLUTION, 344
    ON STATEMENT, 142, 162
    ON THEORY-INCLUSION, 346
    ON TYPE, 152, 346
  OLD ATTRIBUTE ON PRESENTATION (DEPRECATED IN OMDOC 1.2), 343
THEORY ELEMENT
  CONSTITUTIVE, 142
THEORY-CONSTITUTIVE, 147, 152, 166
  ELEMENT, 142
THEORY-INCLUSION
  ELEMENT, 187, 189, 191, 195, 297, 298, 340, 346, 348, 351, 364, 365
THESIS
  ADVISOR, 107
THESIS
  ATTRIBUTE VALUE
    FOR TYPE ON OMTEXT, 133
THS
  ATTRIBUTE VALUE
    FOR ROLE ON DC:*, 107
TIM
  BERNERS-LEE, 32
TIME
  DEVELOPMENT, 296
TIMESTAMP
  OLD ATTRIBUTE ON AXIOM-INCLUSION (DEPRECATED IN OMDOC 1.2), 347
  OLD ATTRIBUTE ON THEORY-INCLUSION (DEPRECATED IN OMDOC 1.1), 351
DC:TITLE
  ELEMENT, 77, 100, 104, 231, 345
TO
  ATTRIBUTE
    ON AXIOM-INCLUSION, 191
    ON THEORY-INCLUSION, 189
TOKEN
  ELEMENT, 121
TOOLBAR, 306
TOP-DOWN

PROOF
STEP, 178
TOP-LEVEL, 158, 189, 195
DOMAIN, 109
ELEMENT, 96
TOTAL, 168
FUNCTION, 167
TOTAL
ATTRIBUTE
ON SELECTOR, 167, 350
ATTRIBUTE VALUE, 350
TPS, 253, 321–323
TPTP, 293
TR
ELEMENT, 138
TRAMP, 85
TRANG, 246
TRANSCRIBER, 107
TRANSITION
ATTRIBUTE VALUE
FOR TYPE ON OMTEXT, 133
TRANSLATION, 139
LOGIC, 302
TRANSLATION-EQUIVALENT, 20
TRANSLATOR, 40, 107
TRC
ATTRIBUTE VALUE
FOR ROLE ON DC:*, 107
TREE, 73, 169, 175
DOCUMENT, 6
EXPANSION, 231
TRL
ATTRIBUTE VALUE
FOR ROLE ON DC:*, 107
TRUE
ATTRIBUTE VALUE
FOR VERDICT ON ANSWER, 224
TRYBULEC
ANDRZEJ, 27
TURING, 16
TUTORIAL
COMPONENT, 281
TUTORING, 326
TWELF, 23, 293
TYPE, 123, 147, 166
ASSERTIONS, 152
ATTRIBUTE, 9
DECLARATION, 147
ID, 10, 27, 90, 91, 101, 337
IDREF, 347
INTEGRATION, 299
MIME, 41, 106, 117, 217, 219, 220, 363
PRINCIPAL, 147
SYSTEM, 147
TYPE, 123
ATTRIBUTE VALUE
FOR ROLE ON SYMBOL, 145, 167
DC:TYPE
ELEMENT, 105, 106, 110, 231
TYPE
ELEMENT, 146, 147, 152, 155, 167, 333, 338, 346, 351
TYPE
ATTRIBUTE, 100
ON ADT, 166, 339
ON ASSERTION, 151, 243, 331, 347
ON ATTRIBUTE, 157
ON AXIOM, 146, 339
ON DEFINITION, 29, 49, 50, 148
ON DERIVE, 172
ON EXAMPLE, 47, 76, 155
ON IGNORE, 99
ON IMPORTS, 62, 159, 191
ON M:CN, 121
ON MORPHISM, 185, 342
ON NOTE, 139
ON OMDOC, 97
ON OMGROUP, 43, 77, 100, 349
ON OMTEXT, 42, 43, 99, 133–135, 343, 349
ON PHRASE, 135
ON REF, 101, 350
ATTRIBUTE (ON MATHML OBJECTS), 123
OLD ATTRIBUTE ON CODE (DEPRECATED IN OMDOC 1.1), 340, 344
OLD ATTRIBUTE ON OMLET (DEPRECATED IN OMDOC 1.2), 349
OLD ATTRIBUTE ON SELECTOR (DEPRECATED IN OMDOC 1.1), 350
OLD ATTRIBUTE ON SORTDEF (DEPRECATED IN OMDOC 1.2), 344
TYPES

formulae as, 21

UL
element, 130, 138
underspecification
assertion-level proof, 329
Unicode, 6, 228, 229
Unicode, 7, 9, 10, 15, 228, 241, 258, 264, 331
uniform
resource
identifier, 5
locator, 5, 7
uniqueness, 90
UNIQUENESS
attribute
on **DEFINITION**, 148
on **MORPHISM**, 185
unit
left, 191
right, 191
universal
digital
mathematics library, X
Unix, 239, 250
unordered
list, 138
UNSATISFIABLE
attribute value
for **STATUS** on **ASSERTION**, 153
UNSATISFIABLE-CONCLUSION
attribute value
for **STATUS** on **ASSERTION**, 153
UPDATED
attribute value
for **ACTION** on **DC:DATE**, 105
URI, 5, 9, 31, 40, 105, 106, 115, 146, 147, 155, 160, 162, 242, 243
base, 160
effective, 160
namespace, 96, 244
normalization, 160
reference, 10, 45, 74, 96, 100, 121, 127, 139, 148, 152, 159, 187, 195, 200, 201, 216, 217, 220, 222
relative, 160
URL, 5, 7, 238, 239
USE
element, 206, 208–210, 252, 253, 331, 346, 351
UTF-8
encoding, 40

valid
REF, 101
validating
parser, 8, 241
XML
parser, 9, 246
validation, 241
validity, 9, 241
value, 116, 149
VALUE
attribute
on **PARAM**, 222
VALUE-OF
element, 203, 346, 351
VALUETYPE
attribute
on **PARAM**, 222
variable, 18, 114, 121
bound, 18, 114, 116, 125, 210
renaming, 116, 125
variant, 276
font, 91
VERBALIZES
attribute
on **OMTEXT**, 134, 343
on **PHRASE**, 135, 157
VERDICT
attribute
on **ANSWER**, 224
verification
in-the-large, 296, 300
in-the-small, 297, 300
proof, 30
verified
mathematical
document, 327
vernacular
chemistry, 130
computer science, 130
mathematical, 29, 38, 48, 104, 129, 133, 137, 138, 169
version, 163
control, 273
VERSION

ATTRIBUTE, 89
ON CC:LICENSE, 109
ON OMDOC, 40, 96, 243
VERSION NUMBER
CONTENT DICTIONARY, 162
VIA
ATTRIBUTE
ON INCLUSION, 190
VIA A MORPHISM, 185
VIEW
DOCUMENT, 262
KNOWLEDGE-CENTERED, 95
NARRATIVE-CENTERED, 95
VISUALIZATION
SYSTEM, 321
VSE-SL, 296, 297

W3C, 6, 14, 32, 81, 245
WEB
SEMANTIC, 301
WORLD WIDE, 5, 13, 97
WORLD WIDE (), 6
WEB
AGENT, 5
APPLICATION, 279
MATH-ENABLED (), 279
SERVICE, 6
WELL-DEFINED, 143
WELL-FORMED, 243
WHITESPACE, 8
WHITESPACE-SEPARATED LIST, 146, 155
WHO
ATTRIBUTE
ON DC:DATE, 105, 340, 348
WIDTH
OLD ATTRIBUTE ON OMLET (DEPRECATED IN OMDOC 1.2), 347, 349, 350
WINDOWS, 239
WIRIS, 284
WITH
ELEMENT, 351
WOLFRAM RESEARCH, 14
WORD
MICROSOFT, 3
WORKFLOW, 311
WORLD WIDE
WEB, 5, 13, 97
CONSORTIUM, 6
WS:PROVE
ELEMENT, 83
WYSIWYG, 3, 326

XALAN, 249
XHTML, 138, 220, 228, 234, 254, 256, 262, 263, 284
XHTML:OBJECT
ELEMENT, 219, 220, 342
XLINK, 220, 241
XLINK, 139
XLINK:HREF
ATTRIBUTE, 362
XML
APPLICATION, 3, 14, 90, 228, 234
CATALOG, 9, 40, 239, 242
COMMENT, 8, 98, 229
ENTITY, 8, 9, 241
ESCAPING, 8
NAMESPACE, 7, 8, 244
PARSER, 8, 98, 252
PROCESSOR, 239
SCHEMA, 8, 41, 91, 227, 241, 369
STRUCTURE
DOCUMENT MODEL, 228
SYNTAX
DOCUMENT MODEL, 228
VALIDATING (), 9, 246
XML, XII, 2, 3, 6–11, 14–16, 18–20, 26, 27, 34, 37, 39–41, 77, 81–83, 90–92, 96, 98, 99, 104, 105, 114, 115, 117, 119, 120, 122, 125, 127, 130, 142, 145, 160, 175, 199, 202, 203, 209, 215, 216, 227–230, 234, 238, 239, 241–246, 249, 252, 260, 261, 266, 269, 270, 277, 278, 287, 288, 293, 310, 311, 318, 331, 337, 338, 343, 346–348, 361, 363, 369, 382
XML-RPC, 81, 234, 282
XML:ID
ATTRIBUTE, 7, 10, 90, 91, 132, 173, 200, 224, 338, 363
ATTRIBUTE (IN MODULE RT), 139
ATTRIBUTE (ON DUBLIN CORE ELEMENTS), 340
XML:LANG
ATTRIBUTE, 7, 130

XML_CATALOG_FILES, 239
XMLLINT, 242, 246
XMLLINT, 239
XMLNS, 8, 243, 244
XPATH, 10, 203, 204, 206, 261, 339, 362, 366
XPOINTER, 10
XPOINTER
  SHORTHAND, 10
XREF
  ATTRIBUTE, 118, 364
    ON IDX, 139
    ON METHOD, 85, 174, 348
    ON OMSTYLE, PRESENTATION, USE, XSLT, STYLE, 200
    ON OMSTYLE, PRESENTATION, 200, 342, 349
    ON PREMISE, 174, 349
    ON PRESENTATION, 206
    ON REF, 78, 100, 101
XSL
  STYLE SHEET, 238
XSLT, 11, 16, 90, 199, 201–206, 209, 210, 228, 245, 249–252, 254, 262, 286, 338, 346, 347, 350, 351, 359, 442, 450
XSLT
  ELEMENT, 201, 202, 204, 205, 209, 210, 346, 351
XSLTPROC, 239, 249

YACAS, 263, 264, 284
YES
  ATTRIBUTE VALUE
    FOR CROSSREF-SYMBOL ON PRESENTATION, 208, 209
    FOR CR ON ELEMENT, 206
    FOR INDUCTIVE ON HYPOTHESIS, 173
    FOR TOTAL ON SELECTOR, 167, 350

ZERO, 165