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

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

Part I

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 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.

[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.

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 V of this book),
- for an in-depth account of the format and all the possibilities of modeling mathematical documents, the rigorous specification in Part XIV 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 XXXIV.
- 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

Abstract **3**

I Foreword **5**
Foreword 5

Preface **7**
Preface 7

Acknowledgments **9**

II Setting the Stage for Open Mathematical Documents **20**

III Markup for Mathematical Knowledge **23**

1 Mathematical Objects and Formulae **25**
1.1 MathML 26
1.2 OpenMath 28

2 Mathematical Texts and Statements **31**

3 Large-Scale Structure and Context in Mathematics **33**

IV Open Mathematical Documents **35**

4 A Brief History of the OMDoc Format **37**
4.1 The Design Problem 37
4.2 Design Principles 38
4.3 Development History 38

5 Three Levels of Markup **41**

6 Situating the OMDoc Format **43**

7 The Future: An Active Web of (Mathematical) Knowledge **45**

V An OMDoc Primer 47

Preface 49

VI Introduction 51

VII Textbooks and Articles 53

8 Minimal OMDoc Markup 55

9 Structure and Statements 59

10 Marking up the Formulae 61

11 Full Formalization 65

VIII OpenMath Content Dictionaries 69

IX Documented Ontologies 75

X Structured and Parametrized Theories 77

XI A Development Graph for Elementary Algebra 81

XII Courseware and the Narrative/Content Distinction 85

12 A Knowledge-Centered View 87

13 A Narrative-Structured View 91

14 Choreographing Narrative and Content OMDoc 93

15 Summary 95

XIII Communication between Systems 97

XIV The OMDoc Document Format 101

Preface 103

XV The OMDoc Format 105

16 Dimensions of Representation in OMDoc 107

17 OMDoc as a Modular Format 111

18 The OMDoc Namespaces **113**

19 Common Attributes in OMDoc **115**

XVI Mathematical Objects 117

20 OpenMath **119**
20.1 The Representational Core of OpenMath 119
20.2 Programming Extensions of OpenMath Objects 121
20.3 Structure Sharing in OpenMath 122

21 Content MathML **125**
21.1 The Representational Core of Content-MathML 125
21.2 OpenMath vs. Content MathML 126

22 Representing Types in Content-MathML and OpenMath **129**

23 Semantics of Variables **131**

24 Legacy Representation for Migration **133**

XVII Strict OMDoc 135

XVIII Metadata 137

25 Pragmatic to Strict Translation **139**

XIX Mathematical Statements 141

26 Types of Statements in Mathematics **143**

27 Theory-Constitutive Statements in OMDoc **145**
27.1 Symbol Declarations 145
27.2 Axioms 147
27.3 Type Declarations 147
27.4 Definitions 148

28 The Unassuming Rest **149**
28.1 Assertions 149
28.2 Type Assertions 151
28.3 Alternative Definitions 152
28.4 Assertional Statements 152

29 Mathematical Examples in OMDoc **155**

30 Inline Statements **157**

31 Theories as Structured Contexts **159**
31.1 Simple Inheritance 159
31.2 OMDoc Theories as Content Dictionaries 162

32 Strict Translations **163**

XX Mathematical Text 165

33 Mathematical Vernacular 167

34 Rhetoric/Mathematical Roles of Text Fragments 169

35 Phrase-Level Markup of Mathematical Vernacular 171

36 Technical Terms 175

37 Index and Bibliography 177

XXI Document Infrastructure 179

38 The Document Root 181

39 Metadata 183

40 Document Comments 185

41 Document Structure 187

42 Sharing Document Parts 189

43 Abstract Documents 191

44 Frontmatter and Backmatter 193

XXII Dublin Core Metadata in OMDoc 195

45 Dublin Core Ontology 197

46 Pragmatic Dublin Core Elements 199
46.1 Roles in Dublin Core Elements 201

XXIII Managing Rights 205

XXIV Derived Statements 209

47 Derived Definitions 211
47.1 Implicit Definitions 212
47.2 Inductive Definitions 212

48 Abstract Data Types 215

49 Strict Translations 219

XXV Representing Proofs 221

50 Proof Structure 223

51 Proof Step Justifications 227

52 Scoping and Context in a Proof **231**

53 Formal Proofs as Mathematical Objects **233**

XXVI Complex Theories 235

54 Inheritance via Translations **237**

55 Postulated Theory Inclusions **241**

56 Local/Required Theory Inclusions **243**

57 Induced Assertions **245**

58 Development Graphs **247**
58.1 Introduction 247
58.2 OMDoc Development Graphs 249

XXVII Notation and Presentation 251

59 Defining Notations **253**
59.1 Syntax of Notation Definitions 253
59.2 Semantics of Notation Definitions 256

XXVIII Auxiliary Elements 261

60 Non-XML Data and Program Code in OMDoc **263**

61 Applets and External Objects in OMDoc **267**

XXIX Exercises 271

XXX Document Models for OMDoc 275

62 XML Document Models **277**

63 The OMDoc Document Model **279**

64 OMDoc Sub-Languages **281**
64.1 Basic OMDoc 282
64.2 OMDoc Content Dictionaries 282
64.3 Specification OMDoc 282
64.4 MathWeb OMDoc 283
64.5 Educational OMDoc 283
64.6 Reusing OMDoc modules in other formats 283

XXXI Processing OMDoc 284

65 OMDoc resources **286**
65.1 The OMDoc Web Site, Wiki, and Mailing List 286

65.2 The OMDoc distribution . . . 286
65.3 The OMDoc bug tracker . . . 287
65.4 An XML catalog for OMDoc . . . 287
65.5 External Resources . . . 288

XXXII Validating OMDoc Documents 291
65.6 Validation with Document Type Definitions . . . 292
65.7 Validation with RelaxNG Schemata . . . 294
65.8 Validation with XML Schema . . . 295

XXXIII Transforming OMDoc 297
65.9 Extracting and Linking XSLT Templates . . . 298
65.10Interfaces for Systems . . . 299
65.11Presenting OMDoc to Humans . . . 301

XXXIV Applications and Projects 302

66 Introduction 304
66.1 Overview . . . 304
66.2 Application Roles of the OMDoc Format . . . 305

XXXV QMath Parser 307

XXXVI Sentido Integrated Environment 311

67 The User Interface 313

68 Formula Editing 315

69 Future Work and Availability 317

XXXVII MBase 319

XXXVIII A Search Engine for Mathematical Formulae 321

70 A Running Example: The Power of a Signal 323

71 Indexing Mathematical Formulae 325

72 A Query Language for Content Mathematics 327

73 Input Processing 329

74 Result reporting 331

75 Case Studies and Results 333

XXXIX Semantic Interrelation and Change Management 335

76 Semantic Interrelation Via Ontologies 337

77 Change Management 339

78 Variants 341

79 Relations to OMDoc 343

XL MathDox 345

80 Introduction 347

81 The Language 349

82 The MathDox System 351

83 Conclusion 353

XLI ActiveMath 355

84 The ActiveMath System 357
84.1 ActiveMath's Service-Approach 357

85 OMDoc Extensions for ActiveMath 359

86 Usage of Semantic Representation in ActiveMath 361
86.1 Computer Algebra Services 361
86.2 Presentation Component 361
86.3 Copy and Paste 362
86.4 Interactive Concept Map Tool icmap 362
86.5 Semantic Search 362
86.6 OMDoc-Related Components and Tools of ActiveMath 362

XLII Authoring Tools for ActiveMath 363

87 Validation Tools 365

88 Further Authoring Tools for ActiveMath 367

XLIII SWiM – An OMDoc-based Semantic Wiki 369

89 Semantic Wikis 371

90 Design of SWiM 373
90.1 Concepts and Relations 373
90.2 User Interface and Interaction Model 374

91 Further work 375

XLIV Maya 377

92 Overview 379

93 From Textual to Logical Representation 381

94 Verification In-the-large 383

95 Verification In-the-small 385

96 Evolution of Developments 387

97 Conclusion and System Availability 389

XLV HETS 391

98 Motivation 393

99 Institutions, Entailment Systems and Logics 395

100 The Architecture of the Hets System 397

XLVI CPoint 399

101 The CPoint Approach 401

102 The CPoint Application 403

103 Future Work 407

XLVII sTeX: A LaTeX-Based Workflow for OMDoc 409

104 Recap of the TeX/LaTeX System 411

105 A LaTeX-based Workflow for XML-based Mathematical Documents 413

106 Content Markup of Mathematical Formulae in TeX/LaTeX 415

107 Theories and Inheritance of Semantic Macros 417

XLVIII Standardizing Context in System Interoperability 419

108 Context Interoperability via Theory Morphisms 421

109 A Hierarchy of Logical Languages 423

110 Logic Interoperability via Logic Morphisms 425

XLIX Proof Assistants in Scientific Editors **427**

L VeriFun **431**

111 Encoding **433**

112 Decoding **435**

LI Appendix **436**

LII Changes to the specification **437**

A Changes from OMDoc1.2 to OMDoc1.6 **439**

B Planned Changes for OMDoc2.0 **441**

LIII Quick-Reference **443**

LIV Table of Attributes **449**

LV The RelaxNG Schema for OMDoc **455**

C The Sub-Language Drivers **457**

D Module MOBJ: Mathematical Objects and Text **459**

E Module MTXT: Mathematical Text **461**

F Module DOC: Metadata **463**

G Dublin Core Metadata **465**

H MARC Relators for Bibliographic Roles **467**

I Creative Commons Licenses **469**

J Module DOC: Document Infrastructure **471**

K Module ST: Mathematical Statements **473**

L Module ADT: Abstract Data Types **477**

M Module PF: Proofs and Proof objects **479**

N Module EXT: Applets and non-XML data **481**

O Module PRES: Adding Presentation Information **483**

P Module QUIZ: Infrastructure for Assessments **487**

LVI The RelaxNG Schemata for Mathematical Objects 489

Q The RelaxNG Schema for OpenMath 491

R The RelaxNG Schema for MathML 493

S Presentation MathML 495

T Strict Content MathML 503

U Pragmatic Content MathML 505

Part II

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 ?markup-types?), existing solutions (see ?markup:www?), and the current XML-based framework for markup languages on the web (see ?xml?). Then we will elaborate the special requirements for marking up the content of mathematics (see Part II).

[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.

Part III

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.

Chapter 1

Mathematical Objects and Formulae

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

The two best-known open markup formats for representing mathematical formulae for the Web are MathML [Aus+03b] and OpenMath [Bus+04]. There are various other formats that are proprietary or based on specific mathematical software packages like Wolfram Research's Mathematica® [Wol02]. We will not concern ourselves with them, since we are only interested in open formats. Furthermore, we will only give a general overview for the open formats here to survey the state of the art, since content MathML and OpenMath are used for formula representation in the OMDoc format and thus the technical details of the two markup schemes are covered in more detail in the OMDoc specification in Part XV. Figure 1.1 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/`

Figure 1.1: The Status of Markup Standardization for Mathematical Formulae

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

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

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).

1.1 MathML

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

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 1.2. 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 1.2 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. `&sum;` for Σ.

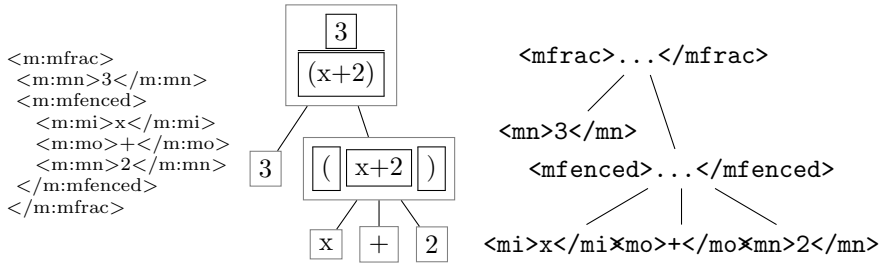


Figure 1.2: The Layout Tree for the Formula $\frac{3}{x+2}$

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 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

standardized 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.

[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.

resource considerations. Presentation-MathML is supported by current versions of the browsers Amaya [Vat], MS Internet Explorer [Cor] (via the MathPlayer plug-in []), and Mozilla [Org].

MathML also offers content markup for mathematical formulae, a sub-language 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 1.3, 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).

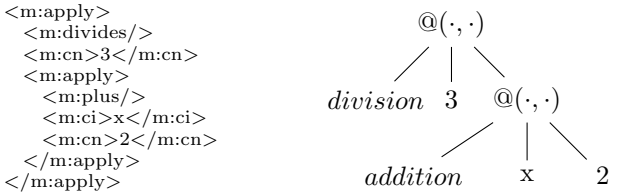


Figure 1.3: The functional Structure of $\frac{3}{x+2}$

Content-MathML offers around 80 specialized elements for the most common K-14 functions and individuals. In Figure 1.3 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`).

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 1.4 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-schemataschema 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.

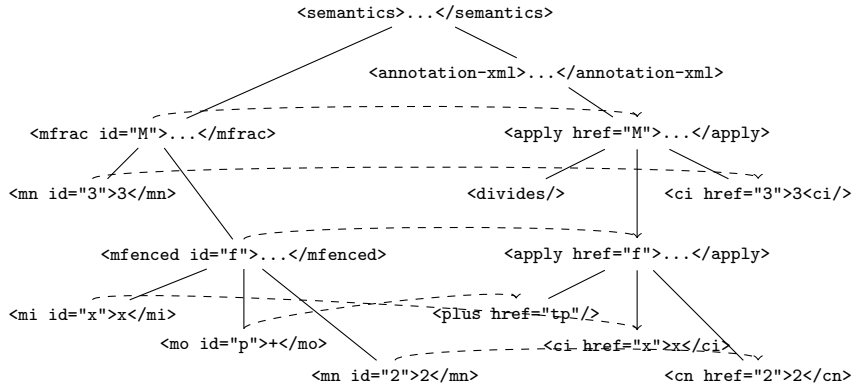


Figure 1.4: Mixing Presentation and Content-MathML

1.2 OpenMath

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

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

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

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

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. As variables do not carry a meaning independent of their local content, `om:OMV` only carries a `name` attribute. See Listing 1.1 for an example that uses most of the elements.

Listing 1.1: 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>
```

[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 [] or the or its infoset [CT04] must be.

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

Listing 1.1 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.

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

For instance, the specification in Listing 1.2 is part of the standard OpenMath content dictionary `arith1.ocd` [] for the elementary arithmetic operations.[8]

Listing 1.2: 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?

[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.

[8] The content of the `omcd:FMP` element is actually the OpenMath object in the representation in Listing 1.1, 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.

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.

Chapter 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 mathematicaldocuments into explicit ones. For instance, mathematical statements like the ones in the document fragment in Figure 2.1 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. □

Figure 2.1: 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 1.1, but the information that this formula is a theorem that has a proof, cannot be directly expressed without extending the formalism. Even formalizations of mathematics like Russell and Whitehead's famous "Principia Mathematica" [WR10] treat this information on the meta-level. If we are willing to extend the mathematical formalism to include primitives for such information, we arrive at formalisms called *logical frameworks* (see [Pfe01] for an overview),

where they are treated as the primary objects of study. The most prevalent approach here uses the "formulae as types" idea that delegates mathematical formulae to the status of types. Logical frameworks capture mathematical statements in formulae and as such can be expressed in Content-MathML or OpenMath. However, this approach relies on full formalization of the mathematical content, and cannot be directly used to capture mathematical practice. In particular, the gap between formal mathematics and informal (but rigorous) treatments of mathematics that rely on natural language as we find them in textbooks and journal articles is wide. The formalization process is so tedious, that it is seldom executed in practice (the "Principia Mathematica" and the Mizar mathematical library [] are solitary examples).

[=math-context]

Chapter 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[1], and therefore only consistent theories make interesting statements about mathematical objects.

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*

[1] 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.

extension", which states that conservative extensions to theories (by theorems and definitions) are safe for mathematical theories, and that possible sources for inconsistencies can be narrowed down to small sets of axioms.

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

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

Even though there have been standardization efforts, the most recent one being the Casl standard (Common Algebraic Specification Language; see [Mos04b]) 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.

Part IV

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.

Chapter 4

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 [**OMEGA02**] 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.

4.1 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 Part XII 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.

4.2 Design Principles

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

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

1. be *Ontologically uncommitted* (like the OpenMath format), so that it could serve as a *integration format* for mathematical software systems.

2. provide a representation format for *mathematical documents* that combined *formal* and *informal* views of all the *mathematical knowledge* contained in them.

3. be based on *sound logic/representational principles* (as not to embarrass the author in front of his colleagues from automated reasoning)

4. be based on *structural/content markup* to guarantee both 1.) and 2.).

4.3 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.

OMDoc1.1 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.

OMDoc1.2 is the mature version in the OMDoc1 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

[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 [06] 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.

follow the XML ID specification [MVW05], and the Dublin Core elements follow the official syntax [DCM03a]). 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.

Chapter 5

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 5.1.

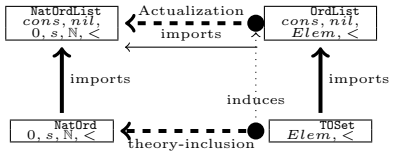
Level of Representation	OMDoc Example
Theory Level: Development Graph • Inheritance via symbol-mapping • Theory inclusion via proof-obligations • Local (one-step) vs. global links	
Statement Level: • Axiom, definition, theorem, proof, example,... • Structure explicit in statement forms and references	<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>

Figure 5.1: OMDoc in a Nutshell (the Three Levels of Modeling)

Building on the discussion in Part II we distinguish three levels of representation in OMDoc

***Mathematical Theories* (see Chapter 0)** 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 Chapter 1)** 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 5.1 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 Chapter 2)** At the level of mathematical formulae, OMDoc uses the established standards OpenMath [Bus+04] and Content-MathML [Aus+03b]. These provide content markup for the structure of mathematical formulae and context markup in the form of URI references in the symbol representations (see Part XV 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.
[=situating-OMDoc]

Chapter 6

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 6.1.

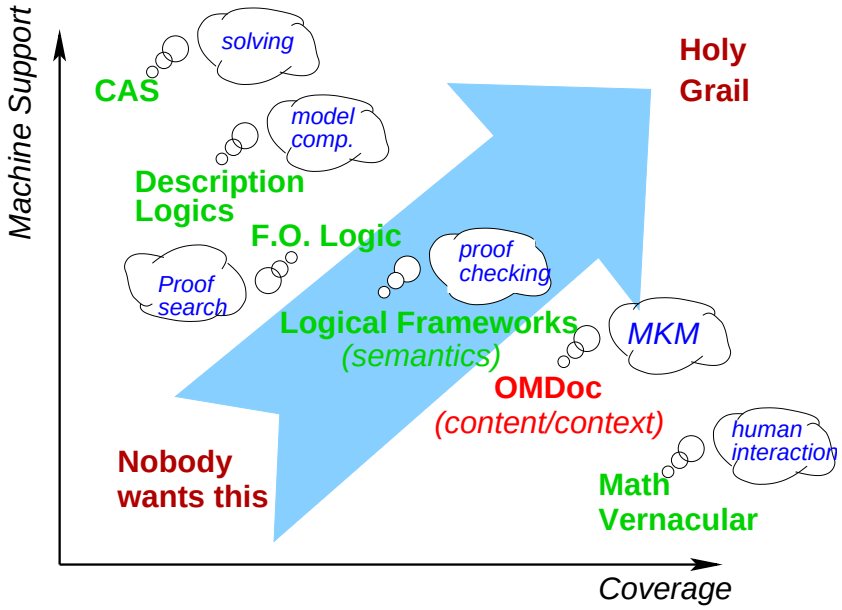


Figure 6.1: Situating Content Markup: Math. Knowledge Management

On the left hand side we see CAS like MATHEMATICA®[Wol02] or MAPLE™ [Cha+92] that are relatively restricted in the mathematical objects — they can deal with polynomials, group representations, differential equations only, but in this domain they can offer sophisticated services like equation solving, factorization, etc. More to the right we see systems like automated theorem provers, whose language — usually first-order logic — covers much more of mathematics, but that cannot perform computational services[1] 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

[1]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.

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

In this sense, content markup offers a more generic markup service (for all formal systems; we do not have to commit ourselves) at the cost of being less precise (we for instance miss out on some symbol equalities). Thus, content markup is placed to the lower right of semantic markup in Figure 6.1. 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.

Chapter 7

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" [Ber98] 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 V

An OMDoc Primer

Preface

This documentprovides 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 ?specification?.

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.

Part VI

Introduction

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.

Part VI: 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.

Part VII 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.

Part IX 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.

Part X 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.

Part XI 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.

Part XII 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.org/repos/omdoc/branches/omdoc-1.6/examples/spec/`

Part VII

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 7.1. 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.

Chapter 7 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 Chapter 8 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 Part XXXVI), or the In2Math knowledge base [Dah01; BB01], or assemble the text slices into individualized books e.g. covering only a sub-topic 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 Part XL can re-assemble the text in different orders.

In Chapter 9, 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 Part XXXII), but also to copy and paste them to symbolic math software systems. Furthermore, an assembly into texts can now be guided by the semantic theory structure, not only by the mathematical text categories or meta-data.

Finally, in Chapter 10 we will fully formalize the mathematical knowledge. This involves a transformation of the mathematical vernacular in the statements into some logical formalism. The main benefit of this is that we can verify the mathematical contents in theorem proving environments like NuPrL [Con+86], Hol [GM93], Mizar [Rud92] and OMEGA [Ben+97].

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).

Figure 7.1: A fragment from Bourbaki's algebra [Bou74]

Chapter 8

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 8.1, we have done this and also added meta-data. Actually, since the `metadata` is 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 8.1: The outer part of the document

```
<?xml version="1.0" encoding="utf-8"?>

<omdoc xml:id="algebra1.omdoc" version="1.6" modules="@basic"
       xmlns:dc="http://purl.org/dc/elements/1.1/"
       xmlns:cc="http://creativecommons.org/ns"
       xmlns="http://omdoc.org/ns">
  <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">
    <h:p 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).
    </h:p>
  </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 Section 64.0), 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.	?validate-dtd?
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 (OMDoc1.6) 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 Section 64.0).	?? p. ??
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 Chapter 9, but spurious namespace prefix declarations are not a problem in the XML world.	Part XIV
8	the namespace declaration for the document; if not prefixed, all elements live in the OMDoc namespace.	Chapter 17
9–29	The metadata for the whole document in Dublin Core format	?? p. ??
10	The title of the document	?? p. ??
11	The document creator, here in the role of a translator	Section 46.0
12	The date and time of first creation of the document in ISO 8601 norm format.	?? p. ??
13	The date and time of the last update to the document in ISO 8601 norm format.	?? p. ??
14–16	A short description of the contents of the document	?? p. ??
17–19	Here we acknowledge that the OMDoc document is just a translation from an earlier work.	?? p. ??
20	The type of the document, this can be `Dataset` (un-ordered mathematical knowledge) or `Text` (arranged for human consumption).	?? p. ??
21	The format/MIME type [FB96] of the document, for OMDoc, this is `application/omdoc+xml`.	?? p. ??
22	The copyright resides with the creator of the OMDoc document	?? p. ??
23–28	The creator licenses the document to the world under certain conditions as specified in the Creative Commons license specified in this element.	?? p. ??
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.	?? p. ??

26	The `cc:prohibitions` can be used to prohibit certain uses of the document, but this one is unencumbered.	?? p. ??
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).	?? p. ??
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".	?? p. ??
32-36	The `h:p` 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 `h:p` elements.	?? p. ??
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.	?? p. ??

Chapter 9

Marking up the text structure and statements

In the next step, we analyze and mark up the structure of the text of the further, and embed the paragraphs into markup for mathematical statements or text segments. Instead of lines 32–36 in Listing 8.1, we will now have the representation in Listing 9.1.

Listing 9.1: The segmented text

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

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

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

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

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

line	Description	ref.

1	The `omtext` element classifies the text fragment as a `definition`, other types for mathematical statements include `axiom`, `example`, `theorem`, and `lemma`. Note that the numbering of the original text is lost, but can be re-created in the text presentation process. The optional `xml:id` attribute specifies a document-unique identifier that can be used for reference later.	?? p. ??
2	A multilingual group of `h:p` 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.	?? p. ??
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.	?? p. ??
15	The three examples in the original in Figure 7.1 are grouped into an enumeration. We use the `omdoc` 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.	?? p. ??
16	We can use the `metadata` of the `omdoc` element to accommodate the title "Examples" in the original. We could enter more metadata at this level.	?? p. ??
18	The `type` attribute of this `omtext` element classifies this text fragment as an example.	?? p. ??

Chapter 10

Marking up the Formulae

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

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

Listing 10.1: The definition of a magma with OpenMath objects

```
<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">
    <h:p>
      Let <om:OMV name="E"/> be a set. A mapping of
     <om:OMA><om:OMS cd="products" name="Cartesian-product"/>
        <om:OMV name="E"/><om:OMV name="E"/>
     </om:OMA> is called a
      <term cd="magmas" name="magma" role="definiendum">law of composition</term>
      on <om:OMV name="E"/>. The value
     <om:OMA><om:OMV name="f"/>
        <om:OMV name="x"/><om:OMV name="y"/>
     </om:OMA>
     of <om:OMV name="f"/> for an ordered pair
     <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"/>
          <om:OMV name="E"/><om:OMV name="E"/>
        </om:OMA>
     </om:OMA> is called the
     <term cd="magmas" name="law_of_composition"
                       role="definiens-applied">composition</term>
      of <om:OMV name="x"/> and <om:OMV name="y"/> under this law.
     A set with a law of composition is called a
     <term cd="magmas" name="magma" role="definiendum">magma</term>.
    </h:p>
  </definition>
...
</theory>
...
```

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 Part VII), 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.	?sub-languagescd?
5	The start tag of a theory. We need this, since symbols and definitions can only appear inside `theory` elements.	?theories?
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`.	?? p. ??
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	?? p. ??
12	Upon closer inspection it turns out that the definition in Listing 10.1 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".	?? p. ??
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.	?? p.??
16	We use an OpenMath object for the set E. It is 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.	???????
17–21	This OpenMath object 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.	?? p. ??

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

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.	?? p. ??
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.	?? p. ??
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 10.1 is analogous. Thus we have treated the first definition in Figure 7.1. 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 10.1 above.

Finally, we come to the examples at the end of our fragment. In the markup shown in Listing 10.2 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 10.1, 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 10.2: 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"/>

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

    <example xml:id="e1.magma" for="#law_of_composition" type="for">
      <h:p>The mappings
        <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> and
        <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="intersection"/>
              <om:OMV name="X"/>
              <om:OMV name="Y"/>
            </om:OMA>
          </om:OMA>
        </om:OMBIND>
       are <term cd="magmas" name="law_of_composition">laws of composition</term>
        on the set of subsets of a set <om:OMS cd="magmas" name="E"/>.
      </h:p>
    </example>

    <example xml:id="e2.magma" for="#law_of_composition" type="for">
      <h:p>
        On the set <om:OMS cd="nat" name="Nat"/> of
        <term cd="nats" name="nats">natural numbers</term>,
        <term cd="nats" name="plus">addition</term>,
        <term cd="nats" name="times">multiplication</term>, and
        <term cd="nats" name="power">exponentiation</term> are ...
      </h:p>
    </example>
  </omdoc>
</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 `h:p` 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 10.3: Examples embedded into a theory

```
<theory xml:id="magmas">
  <imports xml:id="imp3" from="background.omdoc#products"/>
  <imports from="http://mbc.mathweb.org/omstd/relation1.omdoc#relation1"/>
  ...
  <theory xml:id="magmas-examples"
    <imports xml:id="imp4">
       from="http://www.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.

Chapter 11

Full Formalization

The final step in the migration of the text fragment involves a transformation of the mathematical vernacular in the statements into some logical formalism. The main benefit of this is that we can verify the mathematical contents in theorem proving environments. We will start out by dividing the first definition into two parts. The first one defines the symbol `law_of_composition` (see Listing 11.1), and the second one `magma` (see Listing 11.2).

Listing 11.1: 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">
  <h:p>
    Let <om:OMV name="E"/> be a set. A mapping of <om:OMR href="#comp.1"/>
    is called a <term cd="magmas" name="law_of_composition"
                      role="definiens">law of composition</term>
    on <om:OMV name="E"/>.
  </h:p>
  <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>
</definition>
```

The main difference of this definition to the one in the section above is the OpenMath object, which now accompanies the `h:p` element. It contains a formal definition of the property of being a law of composition in the form of a λ-term $\lambda E, F.set(E) \wedge F : E \times E \to E$[1]. The value `simple` of the `type` attribute in the `definition` element signifies that the OpenMath object 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[2]

[1] 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.

[2] 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

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 11.2.

Listing 11.2: The formal definition of a magma

```
<definition xml:id="magma.def" for="#magma" type="implicit">
 <h:p> A set with a law of composition is called a
   <term cd="magmas" name="magma" role="definiens">magma</term>.
 </h:p>
 <FMP>
  <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>
  </om:OMBIND>
 </FMP>
</definition>
```

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

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

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

Finally, the examples get a formal part as well. This mainly consists of formally representing the object that serves as the example, and making the way it does explicit. The first is done simply by adding the object to the example as a sibling node to the h:p. 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 Chapter 19), 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 OpenMath objects, 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 11.3: A formalized magma example

```
<example xml:id="e11.magma" for="#law_of_composition"
         type="for" assertion="e11.magma.ass">
  <h:p> The mapping <om:OMR href="#e11.magma.1"/> is a law of composition
  on the set of subsets of a set <om:OMS cd="magmas" name="E"/>.
  </h:p>
```

the value of A, where all the occurrences of X have been replaced by B. See [Bar80; And02] for an introduction.

```
    <om:OMA id="e11.magma.2"><om:OMS cd="sets" name="subset"/>
      <om:OMV name="E"/>
    </om:OMA>
    <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>
  </example>

  <assertion xml:id="e11.magma.ass">
    <FMP>
      <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>
   </FMP>
  </assertion>
```

Part VIII

OpenMath Content Dictionaries

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

Concretely, we will look at the content dictionary `arith1.ocd` which defines the OpenMath symbols `abs`, `divide`, `gcd`, `lcm`, `minus`, `plus`, `power`, `product`, `root`, `sum`, `times`, `unary_minus` (see [] 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 11.4: 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>
  <Description>
    The symbol to represent the n-ary function to return the least common
    multiple of its arguments.
  </Description>

  <h:p> lcm(a,b) = a*b/gcd(a,b) </h:p>
  <FMP>... </FMP>

  <h:p>
    for all integers a,b |
    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.
  </h:p>
  <FMP>...</FMP>
  ...
</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 11.4 is the OMDoc document in Listing 11.5.

The first 25 lines in Listing 11.4 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 Section 64.1 since it suffices in this case, this is indicated by the `modules` attribute on the `omdoc` element.

Listing 11.5: 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/">

<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>
  <dc:source>
    Derived from the OpenMath CD http://www.openmath.org/cd/arith1.ocd.
  </dc:source>
  <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">
     <OMA><OMS name="mapsto" cd="sts"/>
        <OMA><OMS name="nassoc" cd="sts"/><OMV name="SemiGroup"/></OMA>
        <OMV name="SemiGroup"/>
      </OMA>
    </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">
    <h:p>We define <OMR href="#lcm-def.O"/> as <OMR href="#lcm-def.1"/></h:p>
    <h:p xml:lang="de">
      Wir definieren <OMR href="#lcm-def.O"/> als <OMR href="#lcm-def.1"/>.
    </h:p>
    <requation>
     <OMA id="lcm-def.O">
        <OMS cd="arith1" name="lcm"/>
        <OMV name="a"/><OMV name="b"/>
     </OMA>
     <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>
   </requation>
  </definition>

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

    <assertion xml:id="lcm-prop-3" type="lemma">
      <h:p>For all integers <OMV name="a"/>, <OMV name="b"/> there is no
        <OMR href="#lcm-prop-3.1"/> such that <OMR href="#lcm-prop-3.2"/> and
        <OMR href="#lcm-prop-3.3"/> and <OMR href="#lcm-prop-3.4"/>.
      </h:p>
      <h:p xml:lang="de">Für alle ganzen Zahlen
        <OMV name="a"/>, <OMV name="b"/>
        gibt es kein <OMR href="#lcm-prop-3.1"/> mit
        <OMR href="#lcm-prop-3.2"/> und
        <OMR href="#lcm-prop-3.3"/> und
        <OMR href="#lcm-prop-3.4"/>.
      </h:p>
      <FMP>
        <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>
        </FMP>
      </assertion>
      ...
    </theory>
    ...
</theory>
```

One important difference between the original and the OMDoc version of the OpenMath content dictionary is that the latter is intended for machine manipulation, and we can transform it into other formats. For instance, the human-oriented presentation of the OMDoc version might look something like the following[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 $c < lcm(a,b)$.*

Figure 11.1: A human-oriented presentation of the OMDoc CD

[3]These presentation was produced by the style sheets discussed in ?omdoc2pres?.

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* $c < kgV(a,b)$.

Figure 11.2: A human-oriented presentation in German

Part IX

Documented Ontologies

OMDoc can be used for marking up ontologies. In particular, we use this for marking up the metadata ontologies, e.g. Part XXI. Until we put a revised version here, please see [LK09].

Part X

Structured and Parametrized Theories

In Part VII 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 [Mos04b]) standard to the Pvs theorem prover [ORS92], since it uses most language elements without becoming too unwieldy to present.

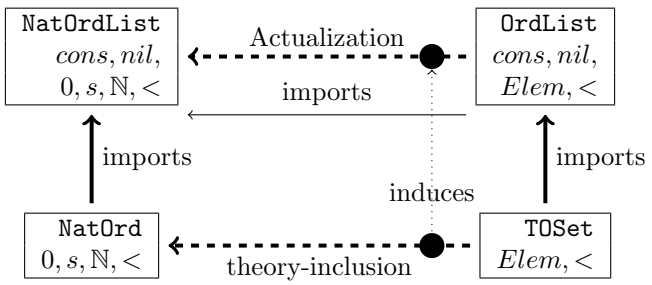


Figure 11.3: A Structured Specification of Lists (of Natural Numbers)

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 begin the extended example with the theories in the lower half of Figure 11.3. 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 47 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 26.1). Finally, the `argument` elements also introduce one partial inverse to the constructor functions per argument; in our case the predecessor function.

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

<theory xml:id="nat">
  <adt>
    <sortdef name="Nat">
      <constructor name="zero"/>
      <constructor name="succ">
        <argument>
          <type><OMS name="Nat" cd="nat"/></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"><h:p>≤ exists.</h:p></assertion>
  <assertion xml:id="leq.unique"><h:p>≤ is unique</h:p></assertion>
  <assertion xml:id="leq.TO"><h:p>≤ is a total order on Nat.</h:p></assertion>
</theory>
```

Finally we have extended the natural numbers by an ordering function $\leq$ (symbol `leq`) which we show to be a total ordering function in assertion `leq.TO`. Note that to state the assertion, we had to import the notion of a total ordering from theory `TOSet`. We can directly use this result to establish a theory inclusion between `TOSet` as the source theory and `NatOrd` as the target theory. A theory inclusion is a formula mapping between two theories, such that the translations of all axioms in the source theory are provable in the target theory. In our case, the mapping is given by the recursive function given in the `morphism` element in Listing X 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 Section 27.3) element of the source theory (the axiom `toset`) has been proven in the target theory, as witnessed by the assertion `leq.TO`[4].

```
<theory-inclusion xml:id="elem-nat-incl" to="#NatOrd" from="#TOSet">
  <morphism xml:id="elem-nat" type="pattern">
    <requation>
      <OMS cd="TOSet" name="set"/>
      <OMS cd="NatOrd" name="Nat"/>
    </requation>
    <requation>
      <OMS cd="TOSet" name="ord"/>
```

[4]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.

```
      <OMS cd="NatOrd" name="leq"/>
    </requation>
  </morphism>
  <obligation induced-by="#toset" assertion="#leq.TO"/>
</theory-inclusion>
```

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

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

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

The theory `NatOrdList` of lists of natural numbers is built up by importing from the theories `NatOrd` and `OrdList`. Note that the attribute `type` of the `imports` element `nat-list.im-elt` is set to `local`, since we only want to import the local axioms of the theory `OrdList` and not the whole theory `OrdList` (which would include the axioms from `TOSet`; see Chapter 55 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 Chapter 57 to obtain a theory inclusion from `OrdList` to `NatOrdList`: We first establish an axiom inclusion from theory `TOSet` to `NatOrdList` by observing that this is induced by composing the theory inclusion from `TOSet` to `NatOrd` with the theory inclusion given by the `imports` from `NatOrd` to `NatOrdList`. This gives us a decomposition situation: every theory that the source theory `OrdList` inherits from has an axiom inclusion to the target theory `NatOrdList`, so the local axioms of those theories are provable in the target theory. Since we have covered all of the inherited ones, we actually have a theory inclusion from the source- to the target theory.

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

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

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

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

Part XI

A Development Graph for Elementary Algebra

We will now use the technique presented in the last chapter for the elementary algebraic hierarchy. Figure 11.4 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.

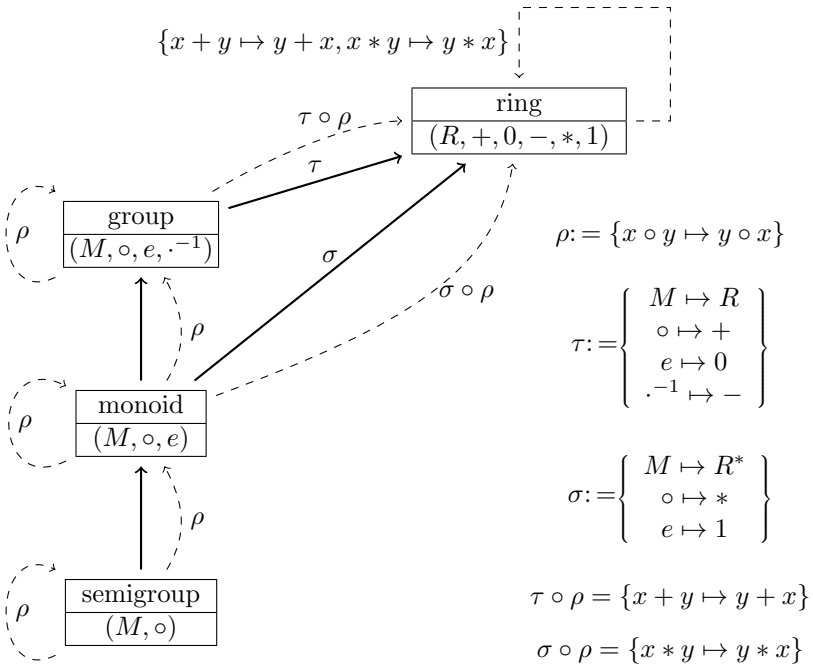


Figure 11.4: A Development Graph for Elementary Algebra

We start off with the theory for *semigroups*. It introduces two symbols, the base set M and the operation $\circ$ on M together with two axioms that state that M is closed under $\circ$ and that $\circ$

is associative on M. We have a structural theory inclusion from this theory to itself that uses the fact that M together with the converse $\sigma(\circ)$ of $\circ$ is also a semigroup: the obligation for the axioms can be justified by themselves (for the closure axiom we have $\sigma(\forall x, y \in M.x \circ y \in M) = \forall y, x \in M.x \circ y \in M$, which is logically equivalent to the axiom.)

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

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

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

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

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

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

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

Note that all of these axiom inclusions have the same morphism (denoted by ρ in Figure 11.4), in OMDoc we can share this structure using the `base` on the `morphism` element. This normally

points to a morphism that is the base for extension, but if the `morphism` element is empty, then this just means that the morphisms are identical.

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

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

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

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

Finally, we extend the whole setup to a theory of rings. Note that we have a dual import from `group` and `monoid` with different morphisms (they are represented by σ and τ in Figure 11.4). 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[5].

```
<theory xml:id="ring">
  <symbol name="R"/>
  <presentation for="#R"><use format="default">R</use></presentation>
  <symbol name="zero"/>
  <presentation for="#plus" role="applied">
    <use format="default">+</use>
  </presentation>
  <symbol name="one"/>
  <presentation for="#zero"><use format="default">0</use></presentation>
  <symbol name="times"/>
  <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">
```

[5] 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.

```
    <morphism>M ↦ R, x ∘ y ↦ x ∗ y, e ↦ 1, .^-1 ↦ −</morphism>
  </imports>
  <imports xml:id="mult.import" from="#monoid">
    <morphism hiding="setstar">M ↦ M*, x ∘ y ↦ x ∗ y, e ↦ 1</morphism>
  </imports>
  <axiom xml:id="dist.ax"><FMP>x ∗ (y + z) = (x ∗ y) + (x ∗ z)</FMP></axiom>
  <assertion xml:id="dist.conv"><FMP>(z + y) ∗ x = (z ∗ x) + (y ∗ x)</FMP></assertion>
</theory>
```

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

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

Part XII

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 Part XL) and can be produced e.g. by **C**Point (see Part XLV).

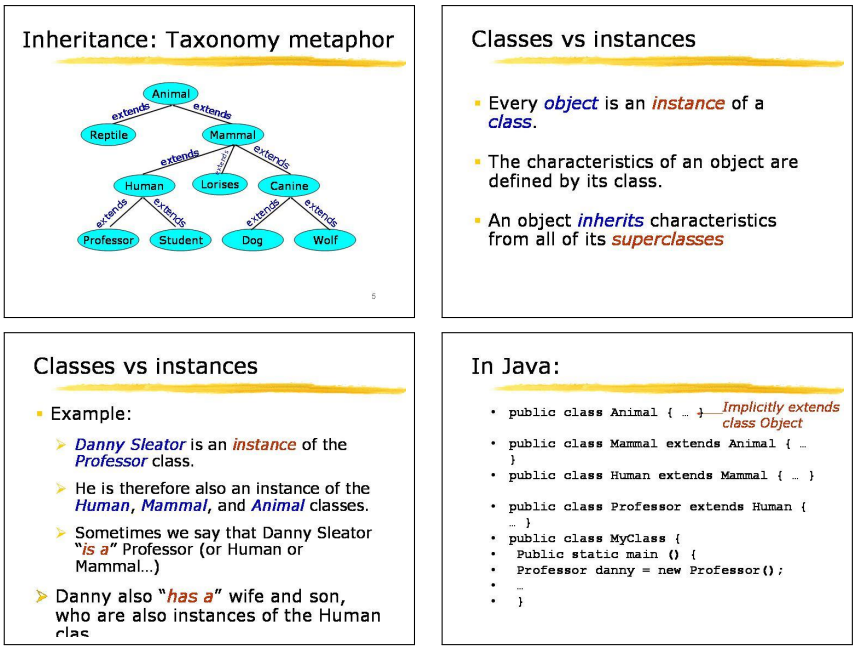


Figure 11.5: Three slides from 15-211

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 OMDocs and narrative-structured ones narrative OMDocs. 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[6] the content with a new slant.

Let us look at the four slides in Figure 11.5. 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 Chapter 12.

[6]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.

Chapter 12

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 11.5 and its structure explicit so that a knowledge management system like MBase (see Part XXXVI) or knowledge presentation system like ActiveMath (see Part XL) 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 12.1. It declares primitive symbols for all the concepts[1] (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 12.1: The OMDoc Representation for Slide 1 from Figure 11.5

```
<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"><OMS cd="kr" name="concept"/></type>
  </symbol>
  <symbol name="mammal">
    <type system="stlc"><OMS cd="kr" name="concept"/></type>
  </symbol>
  ...
  <axiom xml:id="mammal-ext-human">
    <h:p>Humans are Animals.</h:p>
    <FMP>
     <OMA><OMS cd="kr" name="extends"/>
        <OMS cd="animal-taxonomy" name="mammal"/>
        <OMS cd="animal-taxonomy" name="human"/>
     </OMA>
    </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 13.1.)

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

[1] 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".

any other. The three text blocks are marked up as axioms, using the attribute `for` to specify the symbols involved in these axioms. The value of the `for` attribute is a whitespace-separated list of URI references to `symbol` elements.

Listing 12.2: The OMDoc Representation for Slide 2 from Figure 11.5

```
<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="#cvi.object #cvi.instance #cvi.class">
    <h:p>Every <h:span style="font-style:italic;color:blue">object</h:span>
      is an <h:span style="font-style:italic; color:red">instance</h:span>
      of a <h:span style="font-style:italic; color:blue">class</h:span>.
    </h:p>
  </axiom>

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

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

For the third slide it is not entirely obvious which of the OMDoc elements we want to use for markup. The intention of the slide is obviously to give some examples for the concepts introduced in the second slide in terms of the taxonomy presented in the first slide in Figure 11.5. However, the OMDoc `example` element seems to be too specific to directly capture the contents (see p. **??**). 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[2]. 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 12.3 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, the "is a" relation[3]. So we arrive at the theory in Listing 12.3. 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 Chapter 12[4].

Listing 12.3: The OMDoc Representation for Slide 3 from Figure 11.5

```
<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="#cvi.class #cvi.instance #cvi-examples.danny">
    <h:p><h:span style="font-style:italic;color:blue">Danny Sleator</h:span>
      is an <h:span style="font-style:italic; color:red">instance</h:span>
      of the <h:span style="font-style:italic; color:blue">Professor</h:span>
      class.
    </h:p>
  </axiom>
```

[2] 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.

[3] 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.

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

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

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

An alternative, more semantic way to mark up the `assertion` element in the theory above would be to split it into multiple `assertion` and `example` elements, as in Listing 12.4, 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 12.4 — 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). The `type` specifies that this is not a counter-example, and the `assertion` points to the justifying assertion. In this particular case, the reasoning behind the example is pretty straightforward (therefore it has been omitted in the slides), but we will make it explicit to show the mechanisms involved. The `assertion` element just re-states the assertion implicit in the example, we refrain from giving the formal statement in an `FMP` child here to save space. The `just-by` can be used to point to set of proofs for this assertion, in this case only the one given in Listing 12.4. 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 11.5.

Listing 12.4: 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">
      <h:p>Danny Sleator is an instance of the
        <h:span style="font-style:italic; color:blue">Mammal</h:span> class.
6     </h:p>
      <OMS cd="cvi-examples" name="danny"/>
    </example>

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

    <proof xml:id="danny-human-pf" for="#dannys-mammal-thm">
      <derive xml:id="d1">
```

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

The last slide contains a set of Java code fragments that are related to the material before. We have marked them up in the `code` elements in Listing 12.5. 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 12.5: OMDoc Representation of Program Code

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

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

Chapter 13

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 `omdoc` elements with `type slide`.

The representation of the first slide in Figure 11.5 is rather straightforward: we use the `dc:title` element in `metadata` to represent the slide title. Its `class` attribute references a CSS `class` definitionn 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 `omdoc` 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 12.2). 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 13.1: The Narrative OMDoc for Figure 11.5

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

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

<omdoc xml:id="slide-848" type="slide">
  <metadata><dc:title class="15-211-title">Classes vs. instances</dc:title></metadata>
  <omdoc 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"/>
  </omdoc>
</omdoc>

<omdoc xml:id="slide-849" type="slide">
  <metadata><dc:title class="15-211-title">Classes vs. instances</dc:title></metadata>
  <omdoc type="itemize" style="list-style-type:url(square.gif)">
    <omtext style="position:30% 10%" xml:id="ex"><h:p>Example:</h:p></omtext>
    <omdoc 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"/>
    </omdoc>
    <omtext style="position:83% 10%" xml:id="has_a">
      <h:p>
        Danny also &#x201C;<h:span style="font-style:italic;color:red">has
        a</h:span>&#x201D; wife and son, who are also instances of the Human class
      </h:p>
    </omtext>
  </omdoc>
</omdoc>

<omdoc xml:id="slide-850" type="slide">
  <metadata><dc:title class="15-211-title">In Java</dc:title></metadata>
  <omdoc type="itemize">
    <omtext xml:id="slide-850.t1" style="position:80% 10%;color:red">
      <h:p>Implicitly extends class object</h:p>
    </omtext>
    <omtext xml:id="slide-850.t2">
      <h:p><omlet data="#cvic-code1" action="display" show="embed"/></h:p>
    </omtext>
    <omtext xml:id="slide-850.t3">
      <h:p><omlet data="#cvic-code2" action="display" show="embed"/></h:p>
    </omtext>
  </omdoc>
</omdoc>
...
```

Chapter 14

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 16.2. 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 ?knowledge-centered?. The arrows represent the theory inheritance structure, e.g. Theory $\mathcal{F}$ imports theory $\mathcal{N}$.

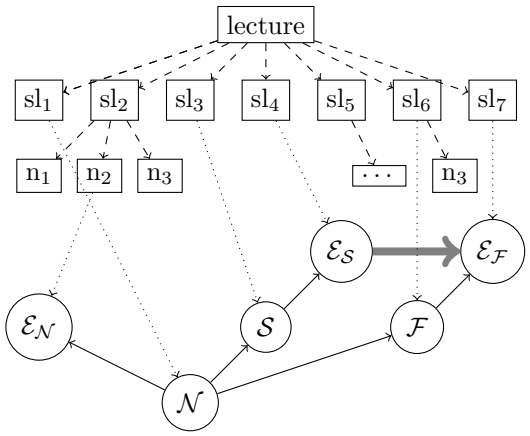


Figure 14.1: An Introduction of a Concept via a Straw-Man Theory

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 16.2, the second slide of "lecture" presents the first example: the text fragment n_1 links the content $\mathcal{E}_\mathcal{N}$, which is referenced from the content structure, to slide

1. The fragment n_2 might say something like "this did not work in the current situation, so we have to extend the conceptualization...".

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

Chapter 15

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 Part XL 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.

Part XIII

Communication with and between Mathematical Software Systems

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

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

On the protocol level, MathWeb-WS uses Soap remote procedure calls with the HTTP binding [Gud+03] (see [Mit03] for an introduction to Soap) interface that allows client applications to request service objects and to use their service methods. For instance, a client can simply request a service object for the automated theorem prover Spass [Wei97] via the HTTP GET request in Listing 15.1 to a MathWeb-WS broker node.

Listing 15.1: Discovering Automated Theorem Provers (Request)

```
GET /ws.mathweb.org/broker/getService?name=SPASS HTTP/1.1
Host: ws.mathweb.org
Accept: application/soap+xml
```

As a result, the client receives a Soap message like the one in Listing 15.2 containing information about various instances of services embodying the Spass prover known to the broker service.

[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.

Listing 15.2: 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 15.3[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 15.2).

Listing 15.3: 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"
          xmlns:omdoc="http://omdoc.org/ns"
          theory="http://mbase.mathweb.org:8080/RPC2#lovelife">
        <h:p>Peter hates somebody</h:p>
        <omdoc:FMP>
         <om:OMBIND xmlns:om="http://www.openmath.org/OpenMath">
           <om:OMS cd="quant1" name="exists"/>
           <om:OMBVAR><om:OMV name="X"/></om:OMBVAR>
           <om:OMA>
             <om:OMS cd="lovelife" name="hate"/>
              <om:OMS cd="lovelife" name="peter"/>
              <om:OMV name="X"/>
            </om:OMA>
          </om:OMBIND>
        </omdoc:FMP>
      </omdoc:assertion>
      <ws:replyWith><ws:state>proof</ws:state></ws:replyWith>
      <ws:timeout>20</ws:timeout>
    </ws:prove>
  </env:Body>
</env:Envelope>
```

This Soap remote procedure call uses a generic method "`prove`" that can be understood by the first-order theorem provers on MathWeb-SB, and in particular the Spass system. This method

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

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 15.3 depends on the information given in the theory `lovelife` referenced in the `theory` attribute in the `assertion` element (see ?theories? 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 Part XXXVI 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 15.4 to the MBASE service at `http://mbase.mathweb.org:8080`.

Listing 15.4: 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 15.5. Here, the SOAP envelope contains the OMDoc representation of the requested theory (irrespective of what the internal representation of MBASE was).

Listing 15.5: The Background Theory for Message 15.3

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

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

This information is sufficient to prove the theorem in Listing 15.3; and the SPASS service might reply to the request with the message in Listing 15.6 which contains an OMDoc representation of a proof (see Part XXIV for details). Note that the `for` attribute in the `proof` element points to the original assertion from Listing 15.3.

Listing 15.6: A proof that Peter hates someone

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

```
Content−Length: 588

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

The proof has two steps: The first one is represented in the `derive` element, which states that "Peter hates Peter". This fact is derived from the two axioms in the theory `lovelife` in Listing 15.5 (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 Part XXIV.

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 XIV

The OMDoc Document Format

Preface

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 is the specification of Version 1.6 of OMDoc of the OMDoc format, the first step towards OMDoc2. 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.

Part XV

The OMDoc Format

In this chapter we will discuss issues that pertain to the general setup of the OMDoc format, before we present the respective modules in later chapters. OMDoc1.6 is the first step towards a second version of the OMDoc format.

Chapter 16

Dimensions of Representation in OMDoc

Strict vs. Pragmatic The OMDoc format is divided into two sublanguages: "Strict" OMDoc (in the lower half of Figure 16.1) and "Pragmatic" OMDoc (in the upper half). The first subset uses a minimal set of elements representing the meaning of a mathematical expression in a uniform structure, while the second one tries to strike a pragmatic balance between verbosity and formality. Both forms of content expressions are legitimate and have their role in representing mathematics. The strict OMDoc format features a minimal set of conceptually orthogonal representational primitives, resulting in expressions with canonical structure, which simplifies the implementation of OMDoc processors as well as the comparison of content expressions. The pragmatic OMDoc format provides a large representational infrastructure that aims at being intuitive for humans to understand, read, and write. In particular, the simplicity and conceptual clarity of strict OMDoc make allow to express structural well-formedness constraints, whereas the vocabulary of pragmatic OMDocis much nearer to mathematical practice and is thus easier to learn. It is a crucial design choice of the OMDocformat that the meaning fo pragmatic representations is defined entirely interms of strict representations[1]. Note that there may be multiple "pragmatic vocabularies" defined in terms of the strict core catering to different communities and their tastes.

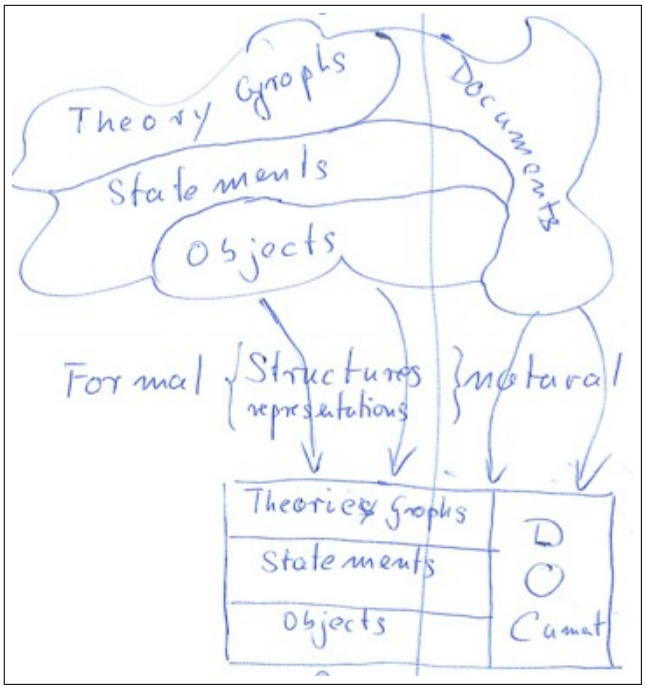


Figure 16.1: Dimensions of Representation in OMDoc

The introduction of strict OMDoc and the re-interpretation of pragmatic OMDoc in terms of it is radical redesign of the OMDoc format, which is new in OMDoc1.6. For this reason we consider OMDoc1.6 the first step into the direction of OMDoc2. With the development of strict OMDoc

[1]The strategy of dividing a markup format into a simple and structurally elegant core language and a larger set of pragmatic extensions which can be given a meaning by translating into the core was first pioneered by the author for content MathML3 [**CarlisleEd:MathML08**]

we aim to identify the representational primitives for representing mathematical documents, which can be given a simple and elegant semantics.

Formal vs. Informal One of the hallmarks of mathematical language is that is very rigorous in structure and usage in an attempt to fix the meaning of (mathematical) objects and statements about them. Indeed, the first decades of the last century established that mathematical language can in principle be expanded into logical form, where all objects and statements are fully identified by their syntactic form, and all reasoning steps are similarly justified by their form alone. we speak of "formal mathematics", when this is exercised and of "formal reasoning", when proofs are carried out in logical systems on this basis . In the last decades, significant parts of mathematical knowledge have been formalized and verified with the help of computers. But formalization and formal reasoning is still so costly and tedious that only a very small part of mathematics is formalized and verified in practice. Currently almost all mathematical documents consist of a mix of formal and informal (i.e. natural language) elements — certainly during the development of mathematical knowledge, but also in publications. Therefore are representation format for mathematical documents must allow this as well, consequently, OMDochas two sub-languages, "formal OMDoc" (on the left side of Figure 16.1) and "natural OMDoc" (on the right side).

OMDocoffers markup at three levels: objects, statements, and context.

objects are usually represented as *formulae* or *natural language phrases* in mathematical documents. In formal OMDocformulae are marked up according to their functional structure (as operator trees) and according to their layout in informal OMDoc(as layout trees). Note that any object can be represented in all three ways and all three ways of representation can be mixed at any level to account for mathematical practice, e.g. for mixed formulae like $\{n \in \mathbb{N} \big| n > 3 \text{ is prime}\}$.

statements are usually represented as *natural language sentences (with formulae)*[2] in informal settings and as (logical) formulae in formal ones. The discussion about the three ways of representation of objects applies analogously. Note that functional markup in formal OMDoconly addresses part of the requirements of formality, since their meaning depends on their context; we will explore this next.

theory graphs The context of objects (and the statements that contain them) is given by special statements (declarations). For conciseness and tractability, OMDocgroups declarations into "theories" and connects them by "theory morphisms" into "theory graphs". In a nutshell, every object (and thus every statement) has a "home theory", in which is meaningful. Theory morphisms make objects and statements available in their target theories.

As statements, theories and theory graphs are large objects, their informal representations (as mathematical text fragments and documents) usually carry linguistic cues to their discourse structure. We discuss the relation between the discourse structure of informal representations and the formal structure of statements and theory graphs next.

Discourse vs. Content Structure Mathematical Documents are very explicitly structured to help the reader grasp the complex objects, their relationships, and the flow of the argumentation in the proofs: Objects are often represented as formulae that reveal their structure, statements are labeled by indicators to their epistemic contribution to context (e.g. by labeling them as "definitions" or "theorems") and numbered for exact reference. The exposition of larger documents usually follows a topical structure with superimposed narrative structure driven by knowledge dependencies rather than e.g. a temporal dramaturgy driven by suspense. Even so, the structure of an informal document may be quite different from the formal structure of the knowledge it introduces. For instance, when we introduce a new concept in a course, 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 $\mathcal{E}_{\mathcal{F}}$.

[2] or even larger text fragments made up of sentences like paragraphs

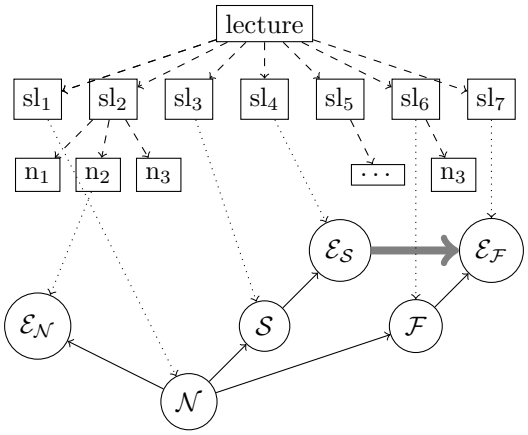


Figure 16.2: Content vs. Narrative Structures

The structure with the solid lines and boxes at the bottom of Figure 16.2 represents the content structure, where the circles $\mathcal{N}$, $\mathcal{E}_\mathcal{N}$, $\mathcal{S}$, $\mathcal{E}_\mathcal{S}$, $\mathcal{F}$, and $\mathcal{E}_\mathcal{F}$ signify theories for the content of the respective concepts and examples. The arrows represent the theory inheritance structure, e.g. Theory $\mathcal{F}$ imports theory $\mathcal{N}$. 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. In the example in Figure 16.2, the second slide of "lecture" presents the first example: the text fragment n_1 introduces it, and n_2 presents $\mathcal{E}_\mathcal{N}$ and n_2 might say something like "this did not work in the current situation, so we have to extend the conceptualization. . . ". In a conventional setting, the narrative structure on the top and the content structure would be represented in different documents: The lecture slides and the formalization, and the equivalences (e.g. that n_2 verbalizes $\mathcal{E}_\mathcal{N}$; we have visualized these relations as dotted arrows in Figure 16.2) could not be taken advantage of, since they are not explicitly represented.

But these equivalences can be utilized to render services to the reader, for instance the imports relation in the theory graph on the lower half of Figure 16.2 induces a dependency relation that can be used to generate a minimal explanation (without the motivation) of $\mathcal{E}_\mathcal{F}$. For an example at the object level, consider for instance the formula $a(x+y^2)$, whose layout is ambiguous in two places: a could be a factor in a product (presented as juxtaposition) or a function that is applied to an argument. Likewise y^2 could be the variable y raised to the second power or the second element in the sequence $y^1, y^2, \ldots, y^n$. Humans can usually disambiguate this from the context, but a screen reader service needs access to the operator tree to read this as "a times [pause] x plus y squared" or "a applied to [pause] x plus y two".

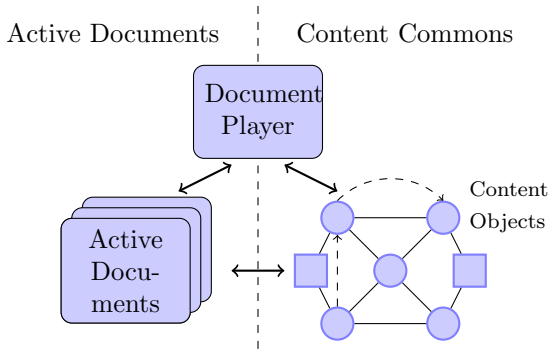


Figure 16.3: The Active Documents Paradigm

OMDocaims to reconcile the dichotomy between discourse structures (in informal mathematical documents which currently carry most of mathematical knowledge) and formal structures (that machines can operate upon) in one joint format. The central technique employed in OMDocis that of "parallel markup": The technique comes from MathML, where the `semantics` element is used to accomodate equivalent layout (presentation MathML) and operator trees (content MathML) and possibly foreign representations. Equivalence of nested sub-structures are represented by special cross-references. The MathML processor choses the one most adequate to its task — in the absence of distinguisthing information the first child.

OMDocextends this to the document level: The document contains elements whose children are alternative representations of the same object/statement/theory. The significance of this is

for that is Figure 16.3 shows the .

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

Coverage Currently our understanding of these primitives is largely limited to formal parts of mathematics, therefore strict OMDoc1.6 covers significantly less of informal mathematical documents than OMDoc1.2, so the meaning-giving translation from pragmatic OMDoc elements to strict OMDoc is partial. We plan to develop strict OMDoc into a system with greater coverage in the upcoming versions of OMDoc. OMDoc2.0 will be the first stable version where the coverage of strict OMDoc is complete.

Chapter 17

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 17.1 before we go into the specifics of the respective modules in Part XV to Part XXVIII. To avoid repetition, we will introduce some attributes already in this chapter that are shared by elements from all modules. In Part XXIX we will discuss the OMDoc document model and possible sub-languages of OMDoc that only make use of parts of the functionality (Chapter 63).

The first four modules in Figure 17.1 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 Part XX and Chapter 63).

Module	Title	Required?	Chapter
MOBJ	Mathematical Objects	yes	Part XV
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	Part XIX
Mathematical vernacular, i.e. natural language with embedded formulae			
DOC	Document Infrastructure	yes	Part XX
A basic infrastructure for assembling pieces of mathematical knowledge into functional documents and referencing their parts			
DC	Metadata	yes	Part XXI and Section 46.0
Contains bibliographical and licensing metadata ("data about data") which can be used to annotate many OMDoc elements by descriptive and administrative information that facilitates navigation and organization			
RT	Rich Text Structure	no	Chapter 36
Rich text structure in mathematical vernacular (lists, paragraphs, tables, ...)			
ST	Mathematical Statements	no	Part XVIII
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	Part XXIV
Structure of proofs and argumentations at various levels of details and formality			
ADT	Abstract Data Types	no	Chapter 47
Definition schemata for sets that are built up inductively from constructor symbols			
CTH	Complex Theories	no	Part XXV
Theory morphisms; they can be used to structure mathematical theories			
DG	Development Graphs	no	Chapter 57
Infrastructure for managing theory inclusions, change management			
EXT	Applets, Code, and Data	no	Part XXVII
Markup for applets, program code, and data (e.g. images, measurements, ...)			
PRES	Presentation Information	no	Part XXVI
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	Part XXVIII
Markup for exercises integrated into the OMDoc document model			

Figure 17.1: The OMDoc Modules

Chapter 18

The OMDoc Namespaces

The namespace for the OMDoc2 format is the URI `http://omdoc.org/ns`. Note that the OMDoc namespace does not reflect the versions[1], this is done in the `version` attribute on the document root element `omdoc` (see Part XX). As a consequence, the OMDoc vocabulary identified by this namespace is not static, it can change with each new OMDoc version. However, if it does, the changes will be documented in later versions of the specification: the latest released version can be found at [OMDoc].

In an OMDoc document, the OMDoc namespace must be specified either using a namespace declaration of the form `xmlns="http://omdoc.org/ns"` on the `omdoc` element or by prefixing the local names of the OMDoc elements by a namespace prefix (OMDoc customarily use the prefixes `omdoc:` or `o:`) that is declared by a namespace prefix declaration of the form `xmlns:o="http://omdoc.org/ns"` on some element dominating the OMDoc element in question (see ?xml? for an introduction). OMDoc also uses the following namespaces[2]:

Format	namespace URI	see
Dublin Core	`http://purl.org/dc/elements/1.1/`	Part XXI and Section 46.0
Creative Commons	`http://creativecommons.org/ns`	Part XXII
MathML	`http://www.w3.org/1998/Math/MathML`	Chapter 20
OpenMath	`http://www.openmath.org/OpenMath`	Chapter 19
XSLT	`http://www.w3.org/1999/XSL/Transform`	Part XXVI

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="1.6"
  xmlns="http://omdoc.org/ns"
  xmlns:cc="http://creativecommons.org/ns"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:om="http://www.openmath.org/OpenMath"
  xmlns:m="http://www.w3.org/1998/Math/MathML">
...
</omdoc>
```

[1] The namespace is different from the OMDoc1 formats (versions 1.0, 1.1, and 1.2), which was `http://www.mathweb.org/omdoc`, but the OMDoc2 namespace will stay constant over all versions of the OMDoc2 format.

[2] 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.

Chapter 19

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, not the type declaration in the DTD or XML schema (see ?xml.validation? 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) [Bos+98], which allow to specify presentational traits like text color, font variant, positioning, padding, or frames of layout boxes, and even aural aspects of the text.

To make use of CSS, most OMDoc elements (all that have `xml:id` attributes) have `style` attributes that can be used to specify CSS directives for them. In the OMDoc fragment in Listing 19.1 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 [Bos+98] and the emerging CSS 3 standard).

Listing 19.1: 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">
    <h:p>Here comes something
      <h:span style="font-weight:bold;color:green" class="emphasize">stylish</h:span>!
    </h:p>
  </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 styles heets, which can be referenced by the `class` attribute. In Listing 19.1 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[1] of the XML document (see [Cla99a] for details). Note that an OMDoc element can have both `class` and `style` attributes, in this case, precedence is determined by the rules for CSS style sheets as specified in [Bos+98]. In our example in Listing 19.1 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.

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

Part XVI

Mathematical Objects (Module MOBJ)

A distinguishing feature of mathematics is its ability to represent and manipulate ideas and objects in symbolic form as mathematical formulae. OMDoc uses the OpenMath and Content-MathML formats to represent mathematical formulae and objects. Therefore, the OpenMath standard [Bus+04] and the MathML 2.0 recommendation (second edition) [Aus+03b] are part of this specification.

We will review OpenMath objects in Chapter 19 and Content-MathML in Chapter 20, and specify an OMDoc element for entering mathematical formulae (element `legacy`) in Chapter 23.

Element	Attributes		Content
	Required	Optional	
`legacy`	`format`	`xml:id`, `formalism`	#PCDATA

Figure 19.1: Mathematical Objects in OMDoc

The recapitulation in the next two sections is not normative, please consult Chapter 0 for a general introduction and history and the OpenMath standard and the MathML 2.0 Recommendation for details and clarifications.

Chapter 20

OpenMath

[=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:`. In OMDoc we embed OpenMath expressions without the enclosing `om:OMOBJ` element, since this does not seem to add anything.

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

Figure 20.1: OpenMath Objects in OMDoc

20.1 The Representational Core of OpenMath

Definition 20.1.1 The central construct of the OpenMath is that of an OpenMath object, 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**).

om:OMA

om:OMV

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 Section 31.1 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).[1]

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 ?math-markup.openmath? 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 Chapter 22 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:

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

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 $\forall x.\sin(x) \leq \pi$.

```
<OMBIND xmlns="http://www.openmath.org/cd">
  <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>
```

`om:OMBIND` **Definition 20.1.2** The **om:OMBIND** element has exactly three children, the first one is a "binding operator"[2] — 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

`om:OMBVAR`

[1] Note that while the `cdbase` inheritance mechanism described here remains in effect for OpenMath objects embedded in to the OMDoc format, it is augmented by one in OMDoc. As a consequence, OpenMath objects in OMDoc documents will usually not contain `cdbase` attributes; see Section 31.1 for a discussion.

[2] The binding operator must be a symbol which either has the role `binder` assigned by the OpenMath content dictionary (see [Bus+04] for details) or the symbol declaration in the OMDoc content dictionary must have the value `binder` for the attribute `role` (see Section 27.0).

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.

OpenMath offers an element for annotating (parts of) formulae with external information (e.g. MathML or LaTeX presentation):

Definition 20.1.3 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)[3], and children at even positions are the **value** s of the keys specified by their immediately preceding siblings. In the OpenMath fragment in Listing 20.1 the expression $x + \pi$ is annotated with an alternative representation and a color. Listing 22.1 has a more complex one involving types.

`om:OMATTR` `om:OMATP`

Listing 20.1: Associating Alternate Representations with an OpenMath Object

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

A special application of the `om:OMATTR` element is associating non-OpenMath objects with OpenMath objects.

Definition 20.1.4 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.

`om:OMFOREIGN`

We recommend a MIME type [FB96] (see ?pres-bound? for an application).

20.2 Programming Extensions of OpenMath Objects

Definition 20.2.1 For representing objects in computer algebra systems OpenMath also provides other basic data types: **om:OMI** for integers, **om:OMB** for byte arrays, **om:OMSTR** for strings, and **om:OMF** for floating point numbers. These do not play a large role in the context of OMDoc, so we refer the reader to the OpenMath standard [Bus+04] for details.

`om:OMI` `om:OMB` `om:OMSTR` `om:OMF`

Definition 20.2.2 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

`om:OME`

[3]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.

one is an error operator[4], 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`).

```
<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.

20.3 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[5] in OpenMath objects. In Figure 20.2 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 20.2 is exponentially larger than the corresponding DAG on the right.

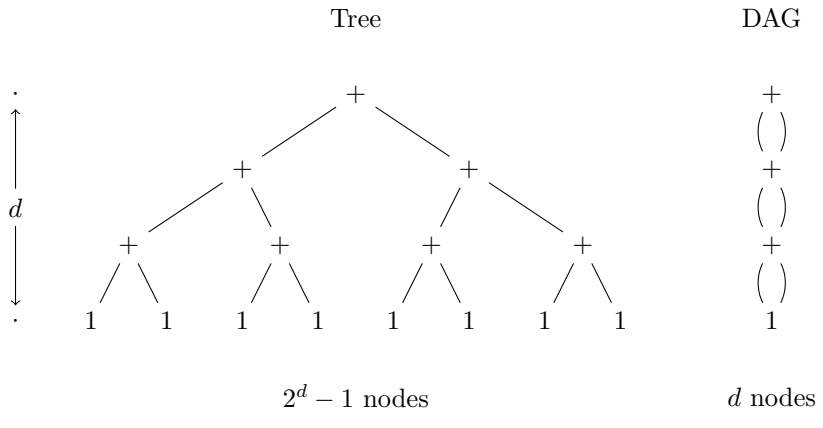


Figure 20.2: Structure Sharing by Directed Acyclic Graphs

[4] An error operator is like a binding operator, only the symbol has role `error`.

[5] 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.

Definition 20.3.1 To support DAG structures, OpenMath2 provides the (optional) attribute `id` on all OpenMath objects and an element **om:OMR** [6] 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.

`om:OMR`

Note that the representation of the `om:OMR` element is *structurally equal*, but not identical to the element it points to.

Using the `om:OMR` element, we can represent the OpenMath objects in Figure 20.2 as the XML representations in Figure 20.3.

Shared	Exploded
<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>	<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>

Figure 20.3: The OpenMath Objects from Figure 20.2 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** [7], i.e. the element that carries the `id` attribute pointed to by the `href` attribute. For instance, in the representation in Figure 20.3 the `om:OMA` element with `xml:id="t1"` and also the second `om:OMA` element dominate the `om:OMA` element with `xml:id="t11"`.

Axiom 20.3.2 (Acyclicity Constraint) An OpenMath element may not dominate itself.

Listing 20.2: A Simple Cycle

```
<OMA id="foo">
```

[6] OpenMath1 and OMDoc1.0 did now know structure sharing, OMDoc1.1 added `xref` attributes to the OpenMath elements `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.

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

```
  <OMS cd="nat" name="divide"/>
  <OMI>1</OMI>
  <OMA><OMS cd="nat" name="plus"/>
    <OMI>1</OMI>
    <OMR href="#foo"/>
  </OMA>
</OMA>
```

In Listing 20.2 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 20.3 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 20.3: A Cycle of Order Two

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

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

Chapter 21

Content MathML

[=cMathML]

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

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

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 Section 21.1 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`	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`			

Figure 21.1: Content-MathML in OMDoc

21.1 The Representational Core of Content-MathML

Definition 21.1.1 The top-level element of MathML is the **m:math** [1] element, see Figure 21.3 for an example. `m:math`

Like OpenMath, Content-MathML organizes the mathematical objects into a functional tree.

The basic objects (MathML calls them **token element** s) are

identifier s (element **m:ci**) corresponding to variables. The content of the `m:ci` element is `m:ci`

[1] 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).

arbitrary Presentation-MathML, used as the name of the identifier.

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

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` `m:bvar` **Definition 21.1.2** 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 21.3 below for a use case in a universal quantifier.

`m:semantics` `m:annotation-xml` `m:annotation` **Definition 21.1.3** 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 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.

21.2 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 contentdictionaries []. For the purposes of OMDoc, we will consider the various representations following four representations of a content symbol in Figure 21.2 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 21.3 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.

[=omml-types]

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

```
<m:plus definitionURL="http://www.openmath.org/cd/arith1#plus"/>a
```
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

Figure 21.2: Four equivalent Representations of a Content Symbol

OpenMath

```
<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>
```

MathML

```
<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>
```

Figure 21.3: OpenMath vs. C-MathML for Commutativity

Chapter 22

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 heavy-weight 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 22.1 and 22.2 show the realizations of a quantification over a variable of functional type in both formats.

Listing 22.1: A Complex Type in OpenMath

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

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 22.2: 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>
```

[=omml-semvar]

Chapter 23

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 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)., 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 renamedo without changing the object (α-conversion). If `<OMV name="x"/>` is not bound, then the scope of the variable cannot be reliably defined; so equality with other occurrences of the variable `<OMV name="x"/>` becomes an ill-defined problem. We therefore discourage the use of unbound variables in OMDoc; they are very simple to avoid by using symbols instead, introducing suitable theories if necessary (see Chapter 30).

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.[1] 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[2] 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 [BFM98]).) 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>

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

[2] 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 ?pres-bound? for a discussion of the OMDoc solution to this problem).

```
  <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 ?pres-bound? for a discussion.

[=legacy]

Chapter 24

Legacy Representation for Migration

Sometimes, OMDoc is used as a migration format from legacy texts (see Part VI for an example). In such documents it can be too much effort to convert all mathematical objects and formulae into OpenMath or Content-MathML form.

Definition 24.0.1 For this situation OMDoc provides the **legacy** element, which can contain arbitrary math markup[1]. The `legacy` element can occur wherever an OpenMath object or Content-MathML expression 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">
  \lambda{x}{x}
</legacy>
```

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

Part XVII

Strict OMDoc

In this chapter we will *strict OMDoc*, a subset of the language that uses a minimal set of elements representing the meaning of a mathematical expression in a uniform structure. Eventually all other parts of OMDoc we call them pragmatic OMDoc will be defined in terms of strict OMDoc (see Chapter 15 for details).

Element	Attributes		M	Content
	Required	Optional	D	
theory	`name, meta`	`xml:id,`	+	`theory* & object* & imports* & instance*`
object	`name, semrole`	`xml:id, synrole`	+	`type*,definition?`
imports	`from`	`xml:id`	+	`--`
instance	`name, from`	`xml:id`	+	`metamorphism?,(maps*&hides*)`
view	`name, from, to`	`xml:id`	+	`metamorphism?,(maps*&hides*)`
maps	`flatname`	`xml:id`	–	⟪*mobj*⟫
hides	`flatname`	`xml:id`	–	
metamorphism		`xml:id`	–	⟪*mobj*⟫
where ⟪*mobj*⟫ is `(OMOBJ |m:math |legacy)`				

Figure 24.1: Strict OMDoc Elements

[=genmeta]

Part XVIII

Metadata (Module DC)

Metadata is "data about data" — in the case of OMDoc data about documents fragments, such as titles, authorship, language usage, or administrative aspects like modification dates, distribution rights, identifiers, or version information. OMDoc documents also contain data about realations between mathematical objects, statements, and theories, that other applications would consider as metadata. To ensure interoperability with such applications and the Semantic Web, OMDoc supports the extraction of OMDoc-specific metadata to the RDF format and the annotation of many OMDoc elements with web-ontology metadata.

In this section we will introduce the metadata framework for strict omdoc, which provides a generic, extensible infrastructure for adding metadata based on the recently stabilized RDFa [Adi+08] a standard for flexibly embedding metadata into X(HT)ML documents. This design decision allows us to separate the *syntax* (which is standardized in RDFa) from the *semantics*, which we externalize in metadata ontologies, which can be encoded in OMDoc; see Part VIII.

The OMDoc format will incorporate various concrete metadata vocabularies as part of the document format. These are given as documented ontologies encoded as OMDoc theories and are normative parts of the format specification. Note that these metadata theories can be thought of as the minimal specifications of the metadata: Refinements are admissible, if they are formulated as OMDoc theories that entertain views into the normative theories.

Element	Attributes		Content
	Req.	Optional	
metadata			(meta|link)*
meta	property,content	datatype	ANY
link	rel	href	(resource|mlink|meta)*
resource		typeof, about	(meta|link)*

Figure 24.2: Metadata in OMDoc

Definition 24.0.2 The **metadta** element contains content encodings for RDF triples and resources. metadta

Chapter 25

Pragmatic to Strict Translation

Every OMDoc element that admits a `metadata` child can serve as a metadata subject, so we can offer the following pragmatic (abbreviative) syntax:

pragmatic	strict equivalent
<⟪*elt*⟫ rel="⟪*r*⟫" href="⟪*h*⟫"> ⟪*body*⟫ </⟪*elt*⟫>	<⟪*elt*⟫> <metadata> <link rel="⟪*r*⟫" href="⟪*h*⟫"/> </metadata> ⟪*body*⟫ </⟪*elt*⟫>
<⟪*elt*⟫ rel="⟪*r*⟫" href="⟪*h*⟫"> <metadata> ⟪*meta*⟫ </metadata> ⟪*body*⟫ </⟪*elt*⟫>	<⟪*elt*⟫> <metadata> <link rel="⟪*r*⟫" href="⟪*h*⟫"/> ⟪*meta*⟫ </metadata> ⟪*body*⟫ </⟪*elt*⟫>

OMDoc1.2 had some descriptions of metadata inhertance for Dublin Core Metadata. In the new metadata framework we do not need to explicitly present this, since it is part of the metadata ontology: If a metadatum can be inferred it is defined to be present.

Part XIX

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.

Chapter 26

Types of Statements in Mathematics

[=statementtypes]

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 Chapter 1). 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.

To support this notion of context OMDoc supports an infrastructure for theories using special `theory` elements, which we will introduce in Chapter 30 and extend in Part XXV. Theory-constitutive elements must be contained as children in a `theory` element; we will discuss them in Section 27.3, non-constitutive statements will be defined in Chapter 27. 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

[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.

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.

Figure 26.1: The Peano Axioms

The Peano axioms in Figure 26.1 (implicitly) introduce three symbols: the number 0, the successor function s, and the set $\mathbb{N}$ of natural numbers. The five axioms in Figure 26.1 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.

Chapter 27

Theory-Constitutive Statements in OMDoc

[=constitutive-statements]

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		M	Content
	Required	Optional	C	
symbol	name	xml:id, role, scope, style, class	+	type*
type		xml:id, system, style, class	−	h:p*,⟪*mobj*⟫
axiom	name	xml:id, for, type, style, class	+	h:*,FMP*
definition	for	xml:id, type, style, class	+	h:p*, ⟪*mobj*⟫)?
where ⟪*mobj*⟫ is (OMOBJ |m:math |legacy)				

Figure 27.1: Theory-Constitutive Elements in OMDoc

27.1 Symbol Declarations

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 47.1 would be considered as "symbols" while the concept of a "group" in mathematics would be called a "concept".

`symbol`

Definition 27.1.1 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[1] as defined in XML 1.1 [Bra+04]. The optional attribute `scope`

[1]This limits the characters allowed in a name to a subset of the characters in Unicode 2.0; e.g. the colon

takes the values `global` and `local`, and allows a simple specification of visibility conditions: if the `scope` attribute of a `symbol` has value `local`, then it is not exported outside the theory; The `scope` attribute is deprecated, a formalization using the `hiding` attribute on the `imports` element should be used instead. Finally, the optional attribute `role` that can take the values[2]

`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 47.

If the `role` is not present, the value `object` is assumed.

The children of the `symbol` element consist of a multi-system group of `type` elements (see Section 27.2 for a discussion). For this group the order does not matter. In Listing 27.1 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 `h:p` elements, see the discussion in Part XIX). 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 27.1: 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">
    <OMS cd="arities" name="ternary-relation"/>
  </type>
</symbol>
```

: 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 Part XXVI.

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

27.2 Axioms

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:

Definition 27.2.1 The **axiom** element contains mathematical vernacular as a sequence of `h:p` elements a `FMP` 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 47) which subsumes it, since it is a more succinct representation of the same mathematical content. Finally the `axiom` element has an optional `for` attribute to specify salient semantic objects it uses as a whitespace-separated list of URI references to symbols declared in the same theory, see Listing 27.2 for an example. Finally, the `axiom` element can have an `type` attribute, whose values we leave unspecified for the moment. `axiom`

Listing 27.2: An OMDoc `axiom`

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

27.3 Type Declarations

Types (also called sorts in some contexts) are representations of certain simple sets that are treated specially in (human or mechanical) reasoning processes. A **type declaration** $e\colon t$ makes the information that a symbol or expression e is in a set represented by a type t available to a specified mathematical process. For instance, we might know that 7 is a natural number, or that expressions of the form $\sum_{i=1}^{n} a_i x^i$ are polynomials, if the a_i are real numbers, and exploit this information in mathematical processes like proving, pattern matching, or while choosing intuitive notations. If a type is declared for an expression that is not a symbol, we will speak of a **term declaration** .

Definition 27.3.1 OMDoc uses the **type** element for type declarations. The optional attribute `system` contains a URI reference that identifies the type system which interprets the content. There may be various sources of the set membership information conveyed by a type declaration, to justify it this source may be specified in the optional `just-by` attribute. The value of this attribute is a URI reference that points to an `assertion` or `axiom` element that asserts $\forall x_1, \ldots, x_n.e \in t$ for a type declaration $e\colon t$ with variables $x_1, \ldots, x_n$. If the `just-by` attribute is not present, then the type declaration is considered to be generated by an implicit axiom, which is considered theory-constitutive[3]. `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. A `type` element with two mathematical objects represents a term declaration $e\colon t$, where the first object represents the expression e and the second one the type t (see Listing 28.2 for an example). There 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).

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).

[3]It is considered good practice to make the axiom explicit in formal contexts, as this allows an extended automation of the knowledge management process.

27.4 Definitions

Definitions are a special class axioms that completely fix the meaning of symbols.

`definition` **Definition 27.4.1** 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. We call these symbols **defined** and **primitive** otherwise. A `definition` contains mathematical vernacular as a sequen ce of `h:p` elements to describe the meaning of the defined symbols.

A `definition` element contains a mathematical object that can be substituted for the symbol specified in the `for` attribute of the definition. The `type` is fixed to `simple`. Listing 27.3 gives an example of a simple definition of the number one from the successor function and zero. OMDoc treats the `type` attribute as an optional attribute. If it is not given explicitly, it defaults to `simple`.

Listing 27.3: A Simple OMDoc `definition`.

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

Chapter 28

The Unassuming Rest

[=non-constitutive-statements]

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 Part XXIV.

Element	Attributes		M	Content
	Required	Optional	D	
`assertion`		`xml:id, type, theory, class, style, status, just-by`	+	`h:p*, FMP*`
`type`	`system`	`xml:id, for, just-by, theory, class, style`	−	`h:p*,` ⟪*mobj*⟫`,`⟪*mobj*⟫
`example`	`for`	`xml:id, type, assertion, theory, class, style`	+	`h:p* |` ⟪*mobj*⟫`*`
`alternative`	`for, theory, entailed-by, entails, entailed-by-thm, entails-thm`	`xml:id, type, theory, class, style`	+	`h:p*, (FMP* | requation* |` ⟪*mobj*⟫`)`
where ⟪*mobj*⟫ is `(OMOBJ |m:math |legacy)`				

Figure 28.1: Assertions, Examples, and Alternatives in OMDoc

28.1 Assertions

Definition 28.1.1 OMDoc uses the **assertion** element for all statements (proven or not) about mathematical objects (see Listing 28.1) 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. `assertion`

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 28.2.

Definition 28.1.2 The `assertion` element also takes an optional `xml:id` element that allows to reference it in a document, an optional `theory` attribute to specify the theory that provides

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 Chapter 28).	
`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.	

Figure 28.2: Types of Mathematical Assertions

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 47).

Listing 28.1: An OMDoc Assertion About Semigroups

```
<assertion xml:id="ida.c6s1p4.l1" type="lemma">
  <h:p> A semigroup has at most one unit.</h:p>
  <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 28.3). Note that the states in Figure 28.3 are not mutually exclusive, but have the inclusions depicted in Figure 28.4.

`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 $\mathcal{C}_i$.

Figure 28.3: Proof Status for Assertions in a Theory $\mathcal{T}$

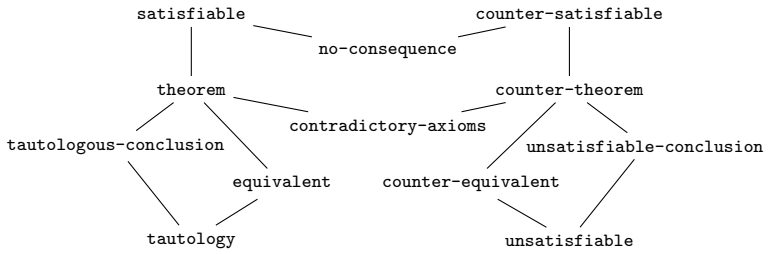


Figure 28.4: Relations of Assertion States

28.2 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 Section 27.2 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 28.2 contains a type assertion $x + x : evens$, which makes the information that doubling an integer number results in an even number available to the reasoning process.

Listing 28.2: 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://cds.omdoc.org/integers/evens"/>
  </m:math>
</type>

<Assertion xml:id="double-even" type="lemma" theory="adv.int">
  <FMP>
    <m:math>
      <m:apply><m:forall/>
        <m:bvar><m:ci xml:id="x13" type="integer">X</m:ci></m:bvar>
        <m:apply><m:in/>
          <m:apply><m:plus/>
            <m:ci definitionURL="x13" type="integer">X</m:ci>
            <m:ci definitionURL="x13" type="integer">X</m:ci>
          </m:apply>
          <m:csymbol definitionURL="http://cds.omdoc.org/nat/evens"/>
        </m:apply>
      </m:apply>
    </m:math>
  </FMP>
</assertion>
```

The body of a type assertion contains two mathematical objects, first the type of the object and the second one is the object that is asserted to have this type.

28.3 Alternative Definitions

In contrast to what we have said about conservative extensions at the end of Section 27.3, 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.

`alternative`

Definition 28.3.1 The `for` on the **alternative** points to the primary definition or assertion. 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.

28.4 Assertional Statements

There is another distinction for statements that we will need in the following. Some kinds of mathematical statements add information about the mathematical objects in question, whereas other statements do not. For instance, a symbol declaration only declares an unambiguous name for an object. We will call the following OMDoc elements **assertional** : `axiom` (it asserts central properties about an object), `type` (it asserts type properties about an object), `definition` (this asserts properties of a new object), and of course `assertion`.

The following elements are considered non-assertional: `symbol` (only a name is declared for an object), `alternative` (here the assertional content is carried by the `assertion` elements referenced in the structure-carrying attributes of `alternative`). For the elements introduced below we will discuss whether they are assertional or not in their context. In a nutshell, only statements introduced by the module ADT (see Chapter 47) will be assertional.

Chapter 29

Mathematical Examples in OMDoc

[=examples]

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 $\mathbf{A} = \neg\mathcal{Q}'\mathbf{D}'$.

`example`

Definition 29.0.1 OMDoc specifies this intuition in an **example** element that contains mathematical vernacular as a `h:p` elements for the description and n mathematical objects (the witnesses). It has the attributes

`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`, or `assertion` elements.

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

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

`example` elements are considered non-assertional in OMDoc, since the assertional part is carried by the `assertion` element referenced in the `assertion` attribute.

Note that the list of mathematical objects in an `example` element does not represent multiple examples, but corresponds to the argument list of the symbol, they exemplify. In the example below, the symbol for monoid is a three-place relation (see the type declaration in Listing 27.1), so we have three witnesses.

Listing 29.1: An OMDoc representation of a mathematical example

```
1 <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">
  <h:p>(A*, ::, ϵ) is a monoid.</h:p>
  <FMP>mon(A*, ::, ϵ)</FMP>
</assertion>
...
<example xml:id="mon.ex1" for="monoid" type="for"
         assertion="string.struct.monoid">
  <h:p>The set of strings with concatenation is a monoid.</h:p>
  <OMA id="nat-strings">
    <OMS cd="strings" name="strings"/>
    <OMS cd="setname1" name="N"/>
  </OMA>
  <OMS cd="strings" name="concat"/>
  <OMS cd="strings" name="empty-string"/>
</example>

<assertion xml:id="monoid.are.groups" type="false-conjecture">
 <h:p>Monoids are groups.</h:p>
 <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">
  <h:p>The set of strings with concatenation is not a group.</h:p>
  <OMR href="#nat-strings"/>
  <OMS cd="strings" name="strings"/>
  <OMS cd="strings" name="concat"/>
  <OMS cd="strings" name="empty-string"/>
</example>

<assertion xml:id="strings.isnt.group" type="theorem">
  <h:p>(A*, ::, ϵ) is a monoid, but there is no inverse function for it.</h:p>
</assertion>
```

In Listing 29.1 we show an example of the usage of an `example` element in OMDoc: We declare constructor symbols `strings-over`, that takes an alphabet A as an argument and returns the set A^* of stringss over A, `concat` for strings concatenation (which we will denote by ::), and `empty-string` for the empty string ϵ. Then we state that $\mathcal{W} = (A^*, ::, \epsilon)$ is a monoid in an `assertion` with `xml:id="string.struct.monoid"`. The `example` element with `xml:id="mon.ex1"` in Listing 29.1 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 $\mathbf{A}'$.

Chapter 30

Inline Statements

[=inline-statements]

Note that the infrastructure for statements introduced so far does its best to mark up the interplay of formal and informal elements in mathematical documents, and make explicit the influence of the context and their contribution to it. However, not all statements in mathematical documents can be adequately captured directly. Consider for instance the following situation, which we might find in a typical mathematical textbook.

> **Theorem 3.12***:* In a monoid M the left unit and the right unit coincide, we call it the **unit** of M.

The overt role of this text fragment is that of a mathematical theorem — as indicated by the cue word "**Theorem**", therefore we would be tempted represent it as an `omtext` element with the value `theorem` for the `type` attribute. But the relative clause is clearly a definition (the definiens is even marked in boldface). What we have here is an aggregated verbalization of two mathematical statements. In a simple case like this one, we could represent this as follows:

Listing 30.1: A Simple-Minded Representation of **Theorem 3.12**

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

But this representation remains unsatisfactory: the definition is not part of the theorem, which would really make a difference if the theorem continued after the inline definition. The real problem is that the inline definition is linguistically a phrase-level construct, while the `omtext` element is a discourse-level construct. However, as a phrase-level construct, the inline definition cannot really be 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 `h:span` element and its `verbalizes`, we can do the following:

Listing 30.2: An Inline Definition

```
<assertion xml:id='unit-unique' type="theorem" >
  <h:p>In a monoid M, the left unit and the right unit coincide,
    <h:span verbalizes="#unit-def">we call it the unit of M</h:span>.</h:p>
</assertion>
<symbol name="unit"/>
<definition xml:id="unit-def" for="unit" just-by='#unit-unique'>
  <h:p>We call the (unique) element of a monoid M that acts as a left
    and right unit the <term role="definiendum" name="unit">unit</term> of M.</h:p>
</definition>
```

thus we would have the phrase-level markup in the proper place, and we would have an explicit version of the definition which is standalone[1], and we would have the explicit relation that states that the inline definition is an "abbreviation" of the standalone definition.

[1]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

Chapter 31

Theories as Structured Contexts

[=theories] 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 Part XXV.

Definition 31.0.1 Theories are specified by the **theory** element in OMDoc, which has a required `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 Chapter 19 for a discussion). `theory`

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 Chapter 25 and Section 27.3, even `theory` elements themselves. Note that theory-constitutive elements may *only* occur in `theory` elements.

Definition 31.0.2 Theories can be structured like documents e.g. into sections and the like (see Chapter 40 for a discussion) via the **tgroup** element, which behaves exactly like the `omdoc` element introduced in Chapter 40 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.[1] `tgroup`

Element	Attributes		M	Content
	Req.	Optional	D	
`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 and theory-constitutive elements				

Figure 31.1: Theories in OMDoc

31.1 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

[1] This element has been introduced to keep OMDoc validation manageable: We cannot directly use the `omdoc` element,since there is no simple, context-free way to determine whether an `omdoc` is dominated by a `theory` element.

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 Chapter 50 for details on availability of assertional statements in proofs and justifications.

`imports`

Definition 31.1.1 The meaning of the **imports** element is determined by two attributes:

`from` The value of this attribute is a URI reference that specifies the **source theory** , i.e. the theory we import from. The current theory (the one specified in the parent of the `imports` element, we will call it the **target theory**) inherits the constitutive elements from the source theory.

`type` This optional attribute can have the values `global` and `local` (the former is assumed, if the attribute is absent): We call constitutive elements **local** to the current theory, if they are explicitly defined as children, and else **inherited** . A **local import** (an `imports` element with `type="local"`) only imports the local elements of the source theory, a global import also the inherited ones.

The meaning of nested `theory` elements is given in terms of an implicit imports relation: The inner theory imports from the outer one. Thus

```
<theory xml:id="a.thy">
  <symbol name="aa"/>
  <theory xml:id="b.thy">
    <symbol name="cc"/>
    <definition xml:id="cc.def" for="cc" type="simple">
      <OMS cd="a.thy" name="af"/>
    </definition>
  </theory>
</theory>
```

is equivalent to

```
<theory xml:id="a.thy"><symbol name="aa"/></theory>
<theory xml:id="b.thy">
  <imports from="#a.thy" type="global"/>
  <symbol name="cc"/>
  <definition xml:id="cc.def" for="cc" type="simple">
    <OMS cd="a.thy" name="af"/>
  </definition>
</theory>
```

In particular, the symbol `cc` is visible only in theory `b.thy`, not in the rest of theory `a.thy` in the first representation.

Note that the inherited elements of the current theory can themselves be inherited in the source theory. For instance, in the Listing 31.2 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 31.1: 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 31.1: 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[2]. 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.

In Listing 31.2 we have specified three algebraic theories that gradually build up a theory of groups importing theory-constitutive statements (symbols, axioms, and definitions) from earlier theories and adding their own content. The theory `semigroup` provides symbols for an operation `op` on a base set `set` and has the axioms for closure and associativity of `op`. The theory of monoids imports these without modification and uses them to state the `left-unit` axiom. The theory `monoid` then proceeds to add a symbol `neut` and an axiom that states that it acts as a left unit with respect to `set` and `op`. The theory `group` continues this process by adding a symbol `inv` for the function that gives inverses and an axiom that states its meaning.

Listing 31.2: 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">
    <h:p>neut is a left unit for op.</h:p><FMP>∀x ∈ set.op(x, neut) = x</FMP>
  </axiom>
  <definition xml:id="setstar.def" for="setstar" type="implicit">
    <h:p>·* subtracts the unit from a set </h:p><FMP>∀S.S* = S\{unit}</FMP>
  </definition>
</theory>

<theory xml:id="group">
  <imports from="#monoid"/>
  <symbol name="inv"/>
  <axiom xml:id="left-inv">
    <h:p>For every X ∈ set there is an inverse inv(X) wrt. op.</h:p>
  </axiom>
</theory>
```

The example in Listing 31.2 shows that with the notion of theory inheritance it is possible to re-use parts of theories and add structure to specifications. For instance it would be very simple to define a theory of Abelian semigroups by adding a commutativity axiom.

The set of symbols, axioms, and definitions available for use in proofs in the importing theory consists of the ones directly specified as `symbol`, `axiom`, and `definition` elements in the target theory itself (we speak of **local** axioms and definitions in this case and the ones that are inherited from the source theories via `imports` elements. Note that these symbols, axioms, and definitions (we call them **inherited**) can consist of the local ones in the source theories and the ones that are inherited there.

The local and inherited symbols, definitions, and axioms are the only ones available to mathematical statements and proofs. If a symbol is not available in the home theory (the one given by the dominating `theory` element or the one specified in the `theory` attribute of the statement), then it cannot be used since its semantics is not defined.

[2]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`

31.2 OMDoc Theories as Content Dictionaries

In Part XV, 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 OpenMath standard, these documents are identified as OpenMath content dictionaries, the MathML recommendation is not specific. In the examples above, we have seen that OMDoc documents can contain definitions of mathematical concepts and symbols, thus they are also candidates for "defining documents" for symbols. By the OpenMath2 standard [Bus+04] suitable classes of OMDoc documents can act as OpenMath content dictionaries (we call them OMDoc **content dictionaries** ; see Section 64.1). The main distinguishing feature of OMDoc content dictionaries is that they include `theory` elements with symbol declarations (see Section 27.3) that act as the targets for the pointers in the symbol representations in OpenMath and Content-MathML. The theory name specified in the `xml:id` attribute of the `theory` element takes the place of the `CDname` defined in the OpenMath content dictionary.

Furthermore, the URI specified in the `cdbase` attribute is the one used for disambiguation on `om:OMS` elements (see Chapter 19 for a discussion).

For instance the symbol declaration in Listing 27.1 can be referenced as

```
<OMS cd="elAlg" name="monoid" cdbase="http://omdoc.org/algebra.omdoc"/>
```

if it occurs in a theory for elementary algebra whose `xml:id` attribute has the value `elAlg` and which occurs in a resource with the URI `http://omdoc.org/algebra.omdoc` or if the `cdbase` attribute of the `theory` element has the value `http://omdoc.org/algebra.omdoc`.

To be able to act as an OpenMath2 content dictionary format, OMDoc must be able to express content dictionary metadata (see Listing 11.4 for an example). For this, the `theory` element carries some optional attributes that allow to specify the administrative metadata of OpenMath content dictionaries.

The `cdstatus` attribute specifies the **content dictionary status** , which can take one of the following values: `official` (i.e. approved by the OpenMath Society), `experimental` (i.e. under development and thus liable to change), `private` (i.e. used by a private group of OpenMath users) or `obsolete` (i.e. only for archival purposes). The attributes `cdversion` and `cdrevision` jointly specify the **content dictionary version number** , which consists of two parts, a major **version** and a **revision** , both of which are non-negative integers. For details between the relation between content dictionary status and versions consult the OpenMath standard [Bus+04].

Furthermore, the `theory` element can have the following attributes:

`cdbase` for the content dictionary base which, when combined with the content dictionary name, forms a unique identifier for the content dictionary. It may or may not refer to an actual location from which it can be retrieved.

`cdurl` for a valid URL where the source file for the content dictionary encoding can be found.

`cdreviewdate` for the **review date** of the content dictionary, i.e. the date until which the content dictionary is guaranteed to remain unchanged.

Chapter 32

Strict Translations

We will now give the a formal semantics of the ST elements in terms of strict OMDoc (see Part XVI).

pragmatic	strict
<axiom name="⟪*n*⟫" xml:id="⟪*i*⟫"> ⟪*body*⟫ </axiom>	<object name="⟪*n*⟫" xml:id="⟪*i*⟫"> ⟪*body*⟫ </object>
<symbol name="⟪*n*⟫"> <type system="⟪*s*⟫">⟪*t*⟫</type> </symbol> <definition type="simple" xml:id="⟪*i*⟫" for="⟪*n*⟫"> ⟪*body*⟫ </definition>	<object name="⟪*n*⟫" xml:id="⟪*i*⟫"> <type system="⟪*s*⟫">⟪*t*⟫</type> <definition>⟪*body*⟫</definition> </object>

[=mtext]

Part XX

Mathematical Text (Module MTXT)

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

Element	Attributes		M	Content
	Required	Optional	D	
omtext		xml:id, type, for, class, style, verbalizes	+	h:p*
h:p		xml:id, style, class, index, verbalizes	+	⟪*math vernacular*⟫
h:span		type, for, relation, verbalizes	+	⟪*math vernacular*⟫
term	name	cd, cdbase, role, xml:id, class, style	−	⟪*math vernacular*⟫

Figure 32.1: The OMDoc Elements for Specifying Mathematical Properties

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

Chapter 33

Mathematical Vernacular

OMDoc models mathematical vernacular as parsed text interspersed with content-carrying elements. Most prominently, the OpenMath objects, Content-MathML expressions, and `legacy` elements are used for mathematical objects, see Part XV. The text structure is marked up with the inline fragment of XHTML 1.0 [The02].

In Figure 33.1 we have given an overview over the ones described in this book. The last two modules in Figure 33.1 are optional (see Chapter 63). 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 Chapter 59 and [Koha]), chemistry vernacular about chemical formulae (e.g. represented in Chemical Markup Language [Mur+]).

Module	Elements	Comment	see
XHTML	`h:p` and inline Elements	extended by MTXT	[The02]
MOBJ	`om:OM*`, `m:*`, `legacy`	mathematical Objects	Part XV
MTXT	`h:span`, `term`, `note`, `idx`, `citation`	phrase-level markup	below
DOC	`ref`, `ignore`	document structure	Part XX
EXT	`omlet`	for applets, images, ...	Definition ????

Figure 33.1: OMDoc Modules Contributing to Mathematical Vernacular

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.

Definition 33.0.1 `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`.

Chapter 34

Rhetoric/Mathematical Roles of Text Fragments

This can take e.g. the values `abstract`, `introduction`, `conclusion`, `comment`, `thesis`, `antithesis`, `elaboration`, `motivation`, `evidence`, `transition` with the obvious meanings. In the last five cases `omtext` also has the extra attribute `for`, and in the last one, also an attribute `from`, since these are in reference to other OMDoc elements. The content of an `omtext` element is mathematical vernacular contained in a sequence of `h:p` elements. This can be preceded by a `metadata` element that can be used to specify authorship, give the passage a title, etc. (see Part XXI).

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 Part XVIII for a more elaborated markup infrastructure). These are usually classified as definitions, axioms, etc. Moreover, they are of a form that can (in principle) be formalized up to the level of logical formula; in fact, mathematical vernacular is seen by mathematicians as a more convenient form of communication for mathematical statements that can ultimately be translated into a foundational logical system like axiomatic set theory [Ber91]. For such text fragments, OMDoc reserves the following values for the `type` attribute:

`axiom` (fixes or restricts the meaning of certain symbols or concepts.) An axiom is a piece of mathematical knowledge that cannot be derived from anything else we know.

`definition` (introduces new concepts or symbols.) A definition is an axiom that introduces a new symbol or construct, without restricting the meaning of others.

`example` (for or against a mathematical property).

`proof` (a proof), i.e. a rigorous — but maybe informal — argument that a mathematical statement holds.

`hypothesis` (a local assumption in a proof that will be discharged later) for text fragments that come from (parts of) proofs.

`derive` (a step in a proof), we will specify the exact meanings of this and the two above in Part XXIV and present more structured counterparts.

For the first four values, `omtext` also provides the attribute `for`, as they point to other mathematical aspects such as symbols, assertions, definitions, axioms or alternatives.

Finally, OMDoc also reserves the values `theorem`, `proposition`, `lemma`, `corollary`, `postulate`, `conjecture`, `false-conjecture`, `formula`, `obligation`, `assumption`, `rule` and `assertion` for

statements that assert properties of mathematical objects (see Figure 28.2 in Section 28.0 for explanations). Note that the differences between these values are largely pragmatic or proof-theoretic (conjectures become theorems once there is a proof). 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 Part XVIII. 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 Part XVIII. 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 Chapter 29 for further discussion).

Chapter 35

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 use the `h:span` element from XHTML a general wrapper for sentence-level phrases that allows to mark their specific properties with special attributes and a `metadata` child. The `term` element is naturally restricted to phrases by construction.

`h:span`

Definition 35.0.1 The **h:span** element 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 Chapter 18 and ?omstyle?).

The semantics of the `h:span` element is defined by mapping to the SALT Rhetorical Ontology [Gro+07] i.e. for example we define a `nucleus` phrase to be an instance of `http://salt.semanticauthoring.org/onto/rhetorical-ontology#nucleus`. The `type` attribute serves a linguistic purpose. A `h:span` denoting a part of a sentence that plays an important role in the understanding of the entire text or is simply basic information essential to the author's purpose takes the value of a `nucleus`. A `h:span` that plays a secondary role in the text, i.e. that serves primarily to further explain or support the `nucleus` with additional information takes the value of a `satellite`. The main difference between these two concepts is that a `nucleus` can be comprehended in a context of a text by iself, while on the other hand a `satellite` is incomprehensible without its corresponding `nucleus` phrase. In order to further clarify and annotate this dependence, if a `h:span` element has a value `satellite` for the `type` attribute, it also has the optional attributes `for` and `relation`, which are explained below.

The `relation` attribute gives the type of dependency relation connecting the `satellite` phrase with its corresponding `nucleus` phrase. It can take one of the following values: `antithesis`, `circumstance`, `concession`, `condition`, `evidence`, `means`, `preparation`, `purpose`, `cause`, `consequence`, `elaboration`, `restatement` and `solutionhood`. We go through each of these terms separately to further clarify their role and meaning.

`antithesis` is a relation where the author has a positive regard for the `nucleus`. The `nucleus` and the `satellite` are in contrast i.e. both can not be true, and the intention of the author is to increase the reader's positive regard towards the `nucleus`.

`circumstance` is a relation where the situation presented in the `satellite` is unrealized. It simply provides a framework in the subject matter within which the reader is to interpret the `nucleus`.

`concession` is a relation where the author yet again wants to increase the reader's positive regard for the `nucleus`. This time, by acknowledging a potential or apparent incompatibility between the `nucleus` and the `satellite`.

`condition` is a relation in which the `satellite` represents a hypothetical future, i.e. unrealized situation and the realization of the statement given in the `nucleus` phrase depends on the realization of the situation described in the `satellite` phrase.

`evidence` is a relation where the author wants to increase the reader's belief in the `nucleus` by providing the `satellite`, which is something that the reader believes in or will find credible.

`means` is a relation where the `satellite` represents a method or an instrument which tends to make the realization of the situation presented in the `nucleus` phrase more likely. For instance: **The Gaussian algorithm solves a linear system of equations**, by first reducing the given system to a triangular or echelon form using elementary row operations and then using a back substitution to find the solution.

`preparation` is a relation where the `satellite` precedes the `nucleus` in the text, and tends to make the reader more ready, interested or oriented for reading what is to be stated in the `nucleus` phrase.

`purpose` is a relation where the author wants the reader to recognize that the activity described in the `nucleus` phrase is initiated in order to realize what is described in the `satellite` phrase.

`cause` is a relation where the author wants the reader to recognize that the `satellite` is the cause for the action described in the `nucleus` phrase.

`consequence` is a relation where the author wants the reader to recognize that the action described in the `nucleus` is to have a result or consequences, as described in the `satellite` phrase.

`elaboration` is a relation where the `satellite` phrase provides additional detail for the `nucleus`. For instance: **In elementary number theory, integers are studied without the use of techniques from other mathematical fields.** Questions of divisibility, factorization into prime numbers, investigation of perfect numbers, use of the Euclidean algorithm to compute the GCD and congruences belong here.

`restatement` is a relation where the `satellite` simply restates what is said in the `nucleus` phrase. However the `nucleus` is more central to the authors purposes than the `satellite` is. For instance: The somewhat older term arithmetic is also used to refer to number theory, but is no longer as popular as it once was..... **Number theory used to be called "the higher arithmetic", but this is dropping out of use.**

`solutionhood` is a relation in which the `nucleus` phrase represents a solution to the problem(s) presented in the `satellite` phrase.

Listing 35.1: Phrases and their attribute usage

```
<omtext>
  <h:span id="sat1.2" type="satellite" relation="concession" for="#nuc1.1">Although it
   still shows up in the names of mathematical fields such as arithmetic functions,
  arithmetic of eliptic curves, fundamental theorem of arithmetic
  </h:span>
  <h:span id="nuc1.1" type="nucleus"> the word arithmetic is no longer as popular as it once was
  </h:span>
</omtext>
```

The `for` attribute, available when a `h:span` is denoted to be a `satellite`, is there to link the `h:span` to its corresponding `nucleus` phrase i.e. serves only for referential purposes, holding the value of the URI of the `nucleus` phrase. Thus having the phrases uniquely identified by the

`xml:id` attribute is highly encouraged due to its great relevance for weaving the semantics of a text at this granularity level.

Furthermore, the `h:span` element allows the attribute `index` for parallel multilingual markup: Recall that sibling `omtext` elements form multilingual groups of text fragments. We can use the `h:span` element to make the correspondence relation on text fragments more fine-grained: `h:span` elements in sibling `omtext`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 `omtext` element (but not beyond). Thus the `index` attributes simplify manipulation of multilingual texts, see Listing 37.2 for an example at the discourse level.

Finally, the `h:span` 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 35.2: Parallel Markup between Formal and Informal

```
<h:p>
  If <h:span verbalizes="#isaG">⟨G, ∘⟩ is a group</h:span>, then of course
    <h:span verbalizes="#isaM">it is a monoid</h:span> by construction.
</h:p>
<FMP>
  <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>
</FMP>
```

Another important application of the `verbalizes` is the case of inline mathematical statements, which we will discuss in Chapter 29.

Chapter 36

Technical Terms

In OMDoc we can give the notion of a **technical term** a very precise meaning: it is a span representing a concept for which a declaration exists in a content dictionary (see Section 27.0). In this respect it is the natural language equivalent for an OpenMath symbol or a Content-MathML token[1]. 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:

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

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 Part XXXII), 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`

Definition 36.0.1 The **term** element has one required attribute: `name` and two optional ones: `cd` and `cdbase`. Together they determine the meaning of the phrase just like they do for `om:OMS` elements (see the discussion in Chapter 19 and Section 31.1). 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[2] 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 7.1 in Part VI.

> **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.

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

[1] and is subject to the same visibility and scoping conditions as those; see Chapter 30 for details

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

Listing 36.1: Marking up the Technical Terms

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

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

Chapter 37

Index and Bibliography

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

Figure 37.1: Rich Text Format OMDoc

Definition 37.0.1 (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 `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 idp

Listing 37.1: An Example of Rich Text Structure

```
<omtext>
  <h: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 <em>unfounded</em> statement).
  In reality only some animals are, for instance:</h:p>
  <h:ul id="l1">
    <h:li>sharks (they bite) and </h:li>
    <h:li>bees (they sting).</h:li>
  </h:ul>
  <h:p>If we measure danger by the number of deaths, we obtain</h:p>
  <table xmlns="http://www.w3.org/1999/xhtml">
    <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>
  <h: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
  <h:a href="http://www.yellowpress.com/killerbee.jpg">here</h:a>.</h:p>

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

note

Definition 37.0.2 (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.

All elements in the RT module carry an optional `xml:id` attribute for identification and an `index` attribute for parallel multilingual markup (e.g. Chapter 34 for an explanation and Listing 37.2 for a translation example).

Listing 37.2: Multilingual Parallel Markup

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

Part XXI

Document Infrastructure (Module DOC)

[=document-types] 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 reordered 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 Chapter 0, but now it is on the level of document structure. Note that mathematical documents are often in both categories: a mathematical textbook can be read from front to end, but it can also be used as a reference, accessing it by the index and the table of contents. The way humans work with knowledge also involves a change of state. When we are taught or explore a mathematical domain, we have a linear/narrative path through the material, from which we abstract more and more, finally settling for a semantic representation that is relatively independent from the path we acquired it by. Systems like ActiveMath (see Part XL) 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 Part XI. We will look at markup elements for knowledge-structured documents in Chapter 30.

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 Chapter 18.

Chapter 38

The Document Root

[=omdoc-root]

omdoc

Definition 38.0.1 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 has an optional attribute `xml:id` that can be used to reference the whole document. The optional attribute `version` is used to specify the version of the OMDoc format the contens of the element conforms to. It is fixed to the string `1.6` 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 element. The value of this attribute is a whitespace-separated list of module identifiers (e.g. MOBJ the left column in Figure 17.1), OMDoc sub-language identifiers (see Figure 64.1), or URI references for externally given OMDoc modules or sub-language identifiers.[1] 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.

Here and in the following we will use tables as the one in Figure 38.1 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 "MD" to indicate whether an OMDoc element can have a `metadata` child (see Definition ????), which will be described in the next section. Finally the fifth column describes the content model — i.e. the allowable children — of the element. For this, we will use a form of Backus Naur form notation also used in the DTD: `#PCDATA` stands for "parsed character data", i.e. text intermixed with legal OMDoc elements.) A synopsis of all elements is provided in Part LII.

Element	Attributes		M	Content
	Required	Optional	D	
`omdoc`		`xml:id, type, class, style, version, modules`	+	(⟪*top-level*⟫)*
`ref`	`xref`	`type, class, style`	–	
`ignore`		`type, comment`	–	`ANY`
where ⟪*top-level*⟫ stands for top-level OMDoc elements				

Figure 38.1: OMDoc Elements for Specifying Document Structure.

[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.

Chapter 39

Metadata

[=metadata]

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

In general, metadata can either be embedded in the respective document, or be stated in a separate one. The first facilitates maintenance and control (metadata is always at your fingertips, and it can only be manipulated by the document's authors), the second one enables inference and distribution. OMDoc allows to embed metadata into the document, from where it can be harvested for external metadata formats, such as the XML resource description format (RDF [LS99]). We use one of the best-known metadata schemata for documents – the *Dublin Core* (cf. Part XXI and Section 46.0). The purpose of annotating metadata in OMDoc is to facilitate the administration of documents, e.g. digital rights management, and to generate input for metadata-based tools, e.g. RDF-based navigation and indexing of document collections. Unlike most other document formats OMDoc allows to add metadata at many levels, also making use of the metadata for document-internal markup purposes to ensure consistency.

`metadata`

Definition 39.0.1 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 Part XXII), 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.

Chapter 40

Document Comments

[=comments]

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

Therefore, anything that would normally go into comments should be modeled with an `omtext` element (`type comment`, if it is a text-level comment; see Chapter 33) or with the `ignore` element for persistent comments, i.e. comments that survive processing.

Definition 40.0.1 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 Part XIX). `ignore`

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 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">
    <h:p>This does not make sense yet, in particular, the equation
      [...] cannot be true, think of [...]
    </h:p>
  </ignore>
  <assertion xml:id="heureka">...</assertion>
</ignore>
```

Example 40.0.2 Another good use of the `ignore` element is to use it as an analogon to the in-place error markup in OpenMath objects (see Section 20.1). 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 XML valid. 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 40.1. XML-well-formedness and validity errors will have to be handled by the XML tools involved.

Listing 40.1: 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 42.1 in Chapter 41.

Chapter 41

Document Structure

[=sectioning]

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 `omdoc` element with an optional attribute `type`. It can take the values

`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 `omdoc` 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 Chapter 18).

`enumeration` for ordered lists. The children of this type of `omdoc` 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 `omdoc` 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 Part XXI 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 `omdoc`, in order to allow plugging together the content of different OMDoc documents as `omdoc`s in a larger document. Therefore, all the attributes of the `omdoc` element also appear on `omdoc` elements and behave exactly like those.

[=sharing]

Chapter 42

Sharing and Referring to Document Parts

Definition 42.0.1 As the document structure need not be a tree in hypertext documents, `omdoc` elements also allow empty **ref** elements whose `xref` attribute can be used to reference OMDoc elements defined elsewhere. The optional `xml:id` (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.

`ref`

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 `omdoc` element which has the same children as the original `omdoc` element[1]. 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 ?coursewmare.narrative-structured? 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 42.1 for an example). It also makes it possible to have more than one view on a document using `omdoc` structures that reference a shared set of OMDoc elements. Note that we have embedded the `ref`-targets of the top-level `omdoc` element into an `ignore` comment, so that an OMDoc transformation (e.g. to text form) does not encounter the same content twice.

[1] 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 `omdoc` element, and thus behaves like one.

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

```
<omdoc xml:id="text" type="sequence">
  <ref xref="#t1"/>
  <ref xref="#enum"/>
  <ref xref="#t4"/>
</omdoc>

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

  <omdoc xml:id="enum"
         type="enumeration">
    <ref xref="#t2"/>
    <ref xref="#t3"/>
  </omdoc>
</ignore>
```

Figure 42.1: Flattening a Tree Structure

While the OMDoc approach to specifying document structure is a much more flexible (database-like) approach to representing structured documents[2] 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 42.1. 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 42.1 is just the one on the left.

In Chapter 18 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.

[=docalt]

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

Chapter 43

Abstract Documents

Definition 43.0.1 To be able to support abstract documents that can be concretized, OMDoc supplies the **docalt** element that groups alternative document fragments so that the presentation process can chose among them.

docalt

Example 43.0.2 One very simple example is to group language variants[1] using the optional `xml:lang` attribute to specify 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.

Listing 43.1: A Multilingual Group of `CMP` Elements

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

Listing 43.1 shows an example of a multilingual group. Here, the OpenMath extension by DAG representation (see Chapter 19) 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.

[1] i.e. all the document fragments in this group are direct translations of each other.

Chapter 44

Frontmatter and Backmatter

[=dc-elements]

Part XXII

Dublin Core Metadata in OMDoc

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.

Chapter 45

Dublin Core Ontology

Chapter 46

Pragmatic Dublin Core Elements

In the following we will describe the variant of Dublin Core metadata elements used in OMDoc. 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 [03]) on the authorship elements and with a rights management system based on the Creative Commons Initiative.

The descriptions in this section are adapted from [DCM03a], 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:`.

Element	Attributes		Content
	Req.	Optional	
dc:creator		xml:id, class, style, role	ANY
dc:contributor		xml:id, class, style, role	ANY
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 Part XIX

Figure 46.1: Dublin Core Metadata in OMDoc

Definition 46.0.1 (Titles) 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 (see Part XIX). Multiple `dc:title` elements inside a `metadata` element are assumed to be translations of each other.

Definition 46.0.2 (Creators) 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 46.0.

Definition 46.0.3 (Contributors) A party whose contribution to the publication is secondary to those named in `dc:creator` elements. Apart from the significance of contribution, the semantics of the **dc:contributor** is identical to that of `dc:creator`, it has the same restriction content and carries the same attributes plus a `dc:lang` attribute that specifies the target language in case the contribution is a translation.

`dc:contributor`

Definition 46.0.4 (Subjects) 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`

Definition 46.0.5 (Descriptions) 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 Part XIX.

Definition 46.0.6 (Publishers) 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`

Definition 46.0.7 (Dates) The date and time a certain action was performed on the element that contains this. The content is in the format defined by XML Schema data type `dateTime` (see [BM04] for a discussion), which is based on the ISO 8601 norm for dates and times.

Concretely, the format is $\langle\!\langle YYYY\rangle\!\rangle$-$\langle\!\langle MM\rangle\!\rangle$-$\langle\!\langle DD\rangle\!\rangle$T$\langle\!\langle hh\rangle\!\rangle$:$\langle\!\langle mm\rangle\!\rangle$:$\langle\!\langle ss\rangle\!\rangle$ where $\langle\!\langle YYYY\rangle\!\rangle$ represents the year, $\langle\!\langle MM\rangle\!\rangle$ the month, and $\langle\!\langle DD\rangle\!\rangle$ the day, preceded by an optional leading "`-`" sign to indicate a negative number. If the sign is omitted, "`+`" is assumed. The letter "`T`" is the date/time separator and $\langle\!\langle hh\rangle\!\rangle$, $\langle\!\langle mm\rangle\!\rangle$, $\langle\!\langle ss\rangle\!\rangle$ represent hour, minutes, and seconds respectively. Additional digits can be used to increase the precision of fractional seconds if desired, i.e the format $\langle\!\langle ss\rangle\!\rangle.\langle\!\langle sss\ldots\rangle\!\rangle$ with any number of digits after the decimal point is supported. The **dc:date** element has the attributes `action` and `who` to specify who did what: The value of `who` is a reference to a `dc:creator` or `dc:contributor` element and `action` is a keyword for the action undertaken. Recommended values include the short forms `updated`, `created`, `imported`, `frozen`, `review-on`, `normed` with the obvious meanings. Other actions may be specified by URIs pointing to documents that explain the action.

`dc:date`

Definition 46.0.8 (Types) Dublin Core defines a vocabulary for the document types in [DCM03b]. 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` "*a resource whose content is primarily words for reading. For example – books, letters, dissertations, poems, newspapers, articles, archives of mailing lists. Note that facsimiles or images of texts are still of the genre text.*"

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

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

`dc:type`

Definition 46.0.9 (Formats) The physical or digital manifestation of the resource. Dublin Core suggests using MIME types [FB96]. Following [MSK01] we fix the content of the **dc:format** element to be the string `application/omdoc+xml` as the MIME type for OMDoc. `dc:format`

Definition 46.0.10 (Identifiers) 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`

Definition 46.0.11 (Sources) 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`

Definition 46.0.12 (Relations) Relation of this document to others. The content model of the **dc:relation** element is not specified in the OMDoc format. `dc:relation`

Definition 46.0.13 (Languages) 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`

Definition 46.0.14 (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 Part XXII.

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

46.1 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 [03].

`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` (bibliographic/scientific antecedent) Use for the author responsible for a work upon which the element is based.

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

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

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

`trc` (transcriber) Use for a person who prepares a handwritten or typewritten copy from original material, including from dictated or orally recorded material. This is also the role (on the `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 46.1 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 46.1: 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 46.2 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 46.2: A Formalization with Scientific Antecedent (`ant`)

```
<omdoc xml:id="NNat" version="1.6" 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>
```

[=cc]

Part XXIII

Managing Rights by Creative Commons Licenses

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.

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

Figure 46.2: The OMDoc Elements for Creative Commons Metadata

Definition 46.1.1 The Creative Commons Metadata Initiative [**URL:creativecommons**] 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. After these, a a natural language explanation of the license grant in a math text (see Part XIX). The `cc:license` element has two optional arguments:

`cc:license`

`jurisdiction` which allows to specify the country in whose jurisdiction the license will be enforced[1]. It's value is one of the top-level domain codes of the "Internet Assigned Names Authority (IANA)" []. 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 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:`. All three elements can contain a natural language explanation of their particular contribution to the license grant in a sequence of `h:p` elements.

`cc:permissions`

Definition 46.1.2 (Permissions) 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):

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`

Definition 46.1.3 (Prohibitions) cc:prohibitions are the things the license prohibits.

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

`cc:requirements`

Definition 46.1.4 (Requirements) cc:requirements are restrictions imposed by the license.

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 [] which defines the meaning, and provides an RDF [LS99] based implementation. As we have discussed in Chapter 38, 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 46.3 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).

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

Listing 46.3: 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>
```

Part XXIV

Derived Statements

[=derived-defs]

Chapter 47

Derived Definition Forms

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.

Definiens	Definiendum	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

Figure 47.1: Some Common Definitions

Definitions can have many forms, they can be

- equations where the left hand side is the defined symbol and the right hand side is a term that does not contain it, as in our discussion above or the first case in Figure 47.1. 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 47.1. 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 47.1, the differential equation only characterizes the exponential function up to additive real constants. In this case, the "definition" only restricts the meaning of the exponential function to a set of possible values. We call such a set of axioms a **loose** definition.
- given as a set of equations, as in the third case of Figure 47.1, 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** .

In Figure 47.1 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 Chapter 25):
[=implicit-defs]

Element	Attributes		M	Content
	Required	Optional	C	
definition	for	xml:id, type, style, class, uniqueness, existence	+	h:p* ⟪mobj⟫
definition	for	xml:id, type, style, class, consistency, exhaustivity	+	h:p*, requation+, 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)				

Figure 47.2: Theory-Constitutive Elements in OMDoc

47.1 Implicit Definitions

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` element with `type implicit`. 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 Section 28.3) that represent the respective properties. We give an example of an implicit definition in Listing 47.1.

Listing 47.1: 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">
  <h:p>
    There is at most one differentiable function that solves the
    differential equation in definition <ref type="cite" xref="#exp-def"/>.
  </h:p>
</assertion>
<assertion xml:id="exp-exists">
  <h:p>
    The differential equation in <ref type="cite" xref="#exp-def"/> is solvable.
  </h:p>
</assertion>
```

[=inductive-defs]

47.2 Inductive Definitions

`requation`

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 carries a `type` with value `inductive` and 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).

Listing 47.2 gives an example of a a recursive definition of the addition on the natural numbers.

Listing 47.2: 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>
  <h:p>Addition is defined by recursion on the second argument.</h:p>
  <requation>x + 0 ⇝ x</requation>
  <requation>x + s(y) ⇝ s(x + y)</requation>
```

</definition>

To guarantee termination of the recursive instantiation (necessary to ensure well-definedness), it is possible to specify a measure function and well-founded ordering by the optional `measure` and `ordering` elements which contain mathematical objects. The elements contain mathematical objects.

`measure`

`ordering`

Definition 47.2.1 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.

Definition 47.2.2 Pattern definitions are a special degenerate cases 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 of a `definition` element whose `type` has value `pattern`.

[=adt]

Chapter 48

Abstract Data Types (Module ADT)

[=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 types. 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 [Mos04b]) 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 26.1) 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.

Definition 48.0.1 In OMDoc, we use the **adt** element to specify abstract data types possibly consisting of multiple sorts. It is a theory-constitutive statement and can only occur as a child of a `theory` element (see Chapter 25 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 Part XLIX for an example. `adt`

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 48.1

Element	Attributes		M	Content
	Req.	Optional	D	
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	+	

Figure 48.1: Abstract data types in OMDoc

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

Definition 48.0.2 A **sortdef** element is a highly condensed piece of syntax that declares a sort symbol together with the constructor symbols and their selector symbols of the corresponding sort. It has a required `name` attribute that specifies the symbol name, an optional `type` attribute that can have the values `free`, `generated`, and `loose` with the meaning discussed above. A `sortdef` element contains a set of **constructor** and **insort** elements. The latter are empty elements which refer to a sort declared elsewhere in a `sortdef` with their `for` attribute: An `insort` element with `for="`⟪*URI*⟫`#`⟪*name*⟫`"` specifies that all the constructors of the sort ⟪*name*⟫ are also constructors for the one defined in the parent `sortdef`. Furthermore, the type of a sort given by a `sortdef` element can only be as strict as the types of any sorts included by its `insort` children.

constructor

insort

Listing 48.1 introduces the sort symbols `pos-nats` (positive natural numbers) and `nats` (natural numbers) , the symbol names are given by the required `name` attribute. Since a constructor is in general an n-ary function, a `constructor` element contains n `argument` children that specify the argument sorts of this function along with possible selector functions.

argument

Definition 48.0.3 The argument sort is given as the first child of the **argument** element: a `type` element as described in Section 27.2.

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 48.1). 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`).

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`.

selector

Definition 48.0.4 The **selector** element has a 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 48.1 the `selector` element in the first `sortdef` introduces a selector symbol for the successor function `succ`. As `succ` is a function from `nats` to `pos-nats`, `pred` is a total function from `pos-nats` to `nats`.

recognizer

Definition 48.0.5 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 48.1 introduces such a **recognizer predicate** as the last child of the `sortdef` element for the sort `pos-nats`.

Note that the `sortdef`, `constructor`, `selector`, and `recognizer` elements define symbols of the name specified by their `name` element in the theory that contains the `adt` element. To govern the visibility, they carry the attribute `scope` (with values `global` and `local`) and the attribute `role` (with values `type`, `sort`, `object`).

Listing 48.1: 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><OMS cd='Nat' name="nats"/></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 48.1: 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`.

Chapter 49

Strict Translations

We will now give the a formal semantics of the ST elements in terms of strict OMDoc (see Part XVI).

Implicit definitions are licensed by a description operator in the meta-theory. In a theory $\langle\!\langle t \rangle\!\rangle$, whose meta-theory $\langle\!\langle m \rangle\!\rangle$ contains a description operator $\langle\!\langle that \rangle\!\rangle$ we have

pragmatic	strict

<symbol name="⟪n⟫">
  <type system="⟪s⟫">⟪t⟫</type>
</symbol>
<definition type="implicit"
  xml:id="⟪i⟫" for="⟪f⟫"
  uniqueness="⟪u⟫" existence="⟪e⟫">
  <FMP>⟪Φ[n]⟫</FMP>
</definition>
	<object name="⟪n⟫" xml:id="⟪i⟫">
	  <type system="⟪s⟫">⟪t⟫</type>
	  <definition>
	  <OMBIND>
	  <OMS cd="⟪m⟫" name="⟪that⟫"/>
	  <OMBVAR><OMV name="⟪x⟫"/></OMBVAR>
	  $\boxed{\Phi[x]}$
	  </OMBIND>
	 </definition>
	</object>

where $\Phi[x]$ is obtained from $\Phi[n]$ by replacing all `<OMS cd="⟪t⟫'' name="⟪n⟫"/>` by `<OMV name="⟪n⟫''/>`.

Similarly inductive definitions are licensed by a recursion operator $\langle\!\langle rec \rangle\!\rangle$:

pragmatic	strict

<symbol name="⟪n⟫">
  <type system="⟪s⟫">⟪t⟫</type>
</symbol>
<definition type="inductive"
  xml:id="⟪i⟫" for="⟪f⟫"
  consistency="⟪c⟫" exhaustivity="⟪e⟫">
  <requation>⟪Φ1[n]⟫⟪Ψ1[n]⟫</requation>
  ...
  <requation>⟪Φm[n]⟫⟪Ψm[n]⟫</requation>
</definition>
	<object name="⟪n⟫" xml:id="⟪i⟫">
	  <type system="⟪s⟫">⟪t⟫</type>
	  <definition>
	  <OMBIND>
	  <OMS cd="⟪m⟫" name="⟪rec⟫"/>
	  <OMBVAR><OMV name="⟪x⟫"/></OMBVAR>
	  $\boxed{\Phi_1[x]}\ \boxed{\Psi_2[x]}\ \ldots\ \boxed{\Phi_m[x]}\ \boxed{\Psi_m[x]}$
	  </OMBIND>
	  </definition>
	</object>

Part XXV

Representing Proofs (Module PF)

[=proofs-intro] 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 Chapter 52.

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 49.1) as it could be found in a typical elementary math. textbook, only that we have numbered the proof steps for referencing convenience. Figure 49.1 will be used as a running example throughout this chapter.

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 49.1. 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

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 $q := p_1 \cdots p_n + 1$.
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. □

Figure 49.1: A Theorem with a Proof.

not to be expressive enough to represent the proofs in the HELM library [Asp+01]. As a consequence, the PF module has been redesigned [AKS03] 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 [Sac06] proof representation calculus.

[=proof-structure]

Chapter 50

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 49.1 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 49.1 is given in Listing 50.1.

Listing 50.1: An OMDoc Representation of Figure 49.1.

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

proof

Definition 50.0.1 Proofs are specified by **proof** elements in OMDoc that have the optional 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).

Note that there can be more than one proof for a given assertion.

Element	Attributes		M	Content
	Req.	Optional	D	
proof	for	theory, xml:id, class, style	+	(omtext | derive | hypothesis | symbol | definition)*
proofobject		xml:id, for, class, style, theory	+	(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		−	EMPTY

Figure 50.1: The OMDoc Proof Elements

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

`omtext` OMDoc allows this element to allow for intermediate text in proofs that does not have to have a logical correspondence to a proof step, but e.g. guides the reader through the proof. Examples for this are remarks by the proof author, e.g. an explanation why some other proof method will not work. We can see another example in Listing 50.1 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 50.1 for examples of `derive` proof steps that only state the local assertion. We will consider the specification of justifications in detail in Chapter 50 below. The **derive** element carries an optional `xml:id` attribute for identification and an optional `type` to single out special cases of proofs steps.

derive

The value `conclusion` is reserved for the concluding step of a proof[1], i.e. the one that derives the assertion made in the corresponding theorem.

The value `gap` is used for proof steps that are not justified (yet): we call them **gap steps** . Note that the presence of gap steps allows OMDoc to specify incomplete proofs as proofs with gap steps.

`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 50.1 the classification of step 2 from Figure 49.1 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`.

[1] As the argumentative structure of the proof is encoded in the justification structure to be detailed in Chapter 50, the concluding step of a proof need not be the last child of a proof element.

`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 27.3, only that the role of the `axiom` element described there is taken by the `hypothesis` element. In Listing 50.1 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 51.2 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.

[=justifications]

Chapter 51

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[1] (see Listing 51.1 for an example):

`method`

Definition 51.0.1 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 51.1 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.[2] A method may have OpenMath objects, Content-MathML expressions, `legacy`, `premise`, `proof`, and `proofobject`[3] children. These act as parameters to the method, e.g. for the repeated universal instantiation method in Listing 51.1 the parameters are the terms to instantiate the bound variables.

`premise`

Definition 51.0.2 The **premise** elements are used to refer to already established assertions: other proof steps or statements — e.g. ones 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.

[1] The structural and formal justification elements discussed in this section are derived from hierarchical data structures developed for semi-automated theorem proving (satisfying the logical side). They allow natural language representations at every level (allowing for natural representation of mathematical vernacular at multiple levels of abstraction). This proof representation (see [Ben+97] for a discussion and pointers) is a DAG of nodes which represent the proof steps.

[2] 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 51.1), and the `FMP` holds a content representation for the inference rule, e.g. using the content dictionary [Koh05]. 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.

[3] This object is an alternative representation of certain proofs, see Chapter 52.

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 50.1 to make the dependencies of the argumentation explicit.

If a `derive` step is a logically (or even mathematically) complex step, an expansion into substeps 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 [Con+86]. Thus, proof nodes may have justifications at multiple levels of abstraction in an hierarchical proof data structure. Thus the `method` elements allow to augment the linear structure of the proof by a tree/DAG-like secondary structure given by the `premise` links. Due to the complex hierarchical structure of proofs, we cannot directly utilize the tree-like structure provided by XML, but use cross-referencing. The `derive` step in Listing 51.1 represents an inner node of the proof tree/DAG with three children (the elements with identifiers `A2`, `A4`, and `A5`).

Listing 51.1: A `derive` Proof Step

```
<proof xml:id="proof.2.1.2.proof.D2.1" for="#assertion.2.1.2">
  ...
  <derive xml:id="D2.1">
    <h:p>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).</h:p>
    <FMP>z + (a + (−a)) = (z + a) + (−a)</FMP>
    <method xref="nk−sorts.omdoc#NK−Sorts.forallistar">
      <OMV name="z"/>
      <OMV name="a"/>
      −a
      <premise xref="#A2"/><premise xref="#A4"/><premise xref="#A5"/>
    </method>
  </derive>
  ...
</proof>
...
<theory xml:id="NK−Sorts">
  <metadata>
    <dc:title>Natural Deduction for Sorted Logic</dc:title>
  </metadata>

  <symbol name="forallistar">
    <metadata>
      <dc:description>Repeated Universal Instantiation></dc:description>
    </metadata>
  </symbol>
  <definition xml:id="forallistar.def" for="forallistar" type="informal">
    <CMP>Given n parameters, the inference rule ∀I* instantiates
      the first n universal quantifications in the antecedent with them.</CMP>
  </definition>
  ...
</theory>
```

In OMDoc the `premise` elements must reference proof steps in the current proof or statements (`assertion` or `axiom` elements) in the scope of the current theory: A statement is in **scope** of the current theory, if its home theory is the current theory or imported (directly or indirectly) by the current theory.

Furthermore note that a proof containing a `premise` element is not self-contained evidence for the validity of the `assertion` it proves. Of course it is only evidence for the validity at all (we call such a proof grounded), if all the statements that are targets of `premise` references have grounded proofs themselves[4] 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 51.2) to fortify our intuition.

[4]For `assertion` targets this requirement is obvious. Obviously, `axiom`s do not need proofs, but certain forms of definitions need well-definedness proofs (see Section 27.3). These are included in the definition of a grounded proof.

Listing 51.2: An OMDoc Representation of a Proof by Cases

```
<assertion xml:id="t1" theory="sets">
  <h:p>If a ∈ U or a ∈ V, then a ∈ U ∪ V.</h:p>
  <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">
    <h:p> We prove the assertion by a case analysis.</h:p>
  </omtext>
  <derive xml:id="t1_p1_l1">
    <h:p>If a ∈ U, then a ∈ U ∪ V.</h:p>
    <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">
    <h:p>If a ∈ V, then a ∈ U ∪ V.</h:p>
    <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">
    <h:p> We have considered both cases, so we have a ∈ U ∪ V.</h:p>
  </derive>
</proof>
```

This proof is in sequent style: The statement of all local claims is in self-contained `FMP`s that mark up the statement in `assumption`/`conclusion` form, which makes the logical dependencies explicit. In this example we use inference rules from the calculus "SK",Gentzen's sequent calculus for classical first-order logic [Gen35], which we assume to be formalized in a theory `SK`. Note that local assumptions from the `FMP` should not be referenced outside the `derive` step they were made in. In effect, the `derive` element serves as a grouping device for local assumptions.

Note that the same effect as embedding a `proof` element into a `derive` step can be obtained by specifying the `proof` at top-level and using the optional `for` attribute to refer to the identity of the enclosing proof step (given by its optional `xml:id` attribute), we have done this in the proof in Listing 51.3, which expands the `derive` step with identifier `t1_p1_l1` in Listing 51.2.

Listing 51.3: An External Expansion of Step `t_1_p1_l1` in Listing 51.2

```
<definition xml:id="union.def" for="union">
  ∀P, Q, x.x ∈ P ∪ Q ⇔ x ∈ P ∨ x ∈ Q
</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>
```

```
29    <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>
34      <method xref="sk.omdoc#SK.cut">
          <premise xref="#t1_p1_l1.d2"/>
          <premise xref="#t1_p1_l1.d3"/>
        </method>
      </derive>
39  </proof>
```

[=scoping-proofs]

Chapter 52

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 50.1. The main new feature is that hypotheses can be introduced at some point in the proof, and are discharged later. As a consequence, they can only be used in certain parts of the proof. The hypothesis is inaccessible for inference outside the nearest ancestor `proof` element of the `hypothesis`.

Let us now reconsider the proof in Figure 49.1. 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 49.1) 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 50.1 we have chosen to view step 6 in Figure 49.1 as a top-down proof step rather than a proof comment.

To sum up, every proof step is characterized by a current thesis and a *context*, which is the set of all the local declarations, hypotheses, and local definitions in scope. Furthermore, a step can either introduce a new hypothesis, definition, or declaration or can just be a forward or backward reasoning step. It is a forward reasoning `derive` step if it leaves the current thesis as it is. It is a backward reasoning `derive` step if it opens new subproofs, each one characterized by a new thesis and possibly a new context.

Listing 52.1: A top-down Representation of the Proof in Figure 49.1.

```
1 <assertion xml:id="a1">
```

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

`proof` elements are considered to be non-assertional in OMDoc, since they do not make assertions about mathematical objects themselves, but only justify such assertions. The assertional elements inside the proofs are governed by the scoping mechanisms discussed there, so that using them in a context where assertional elements are needed, can be forbidden.

[=proofobjects]

Chapter 53

Formal Proofs as Mathematical Objects

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 of formal proofs via the Curry/Howard/DeBruijn Isomorphism (see e.g. [Tho91] for an introduction and Figure 53.1 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 53.1 we have assumed a theory `PL0ND` for the calculus of natural deduction in propositional logic which provides the necessary symbols (see Listing 53.1).

`proofobject`

Definition 53.0.1 The **proofobject** element contains an optional multilingual group of `h:p` elements which describes the formal proof as well as a proof object which can be an OpenMath object, Content-MathML expression, 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 53.1.

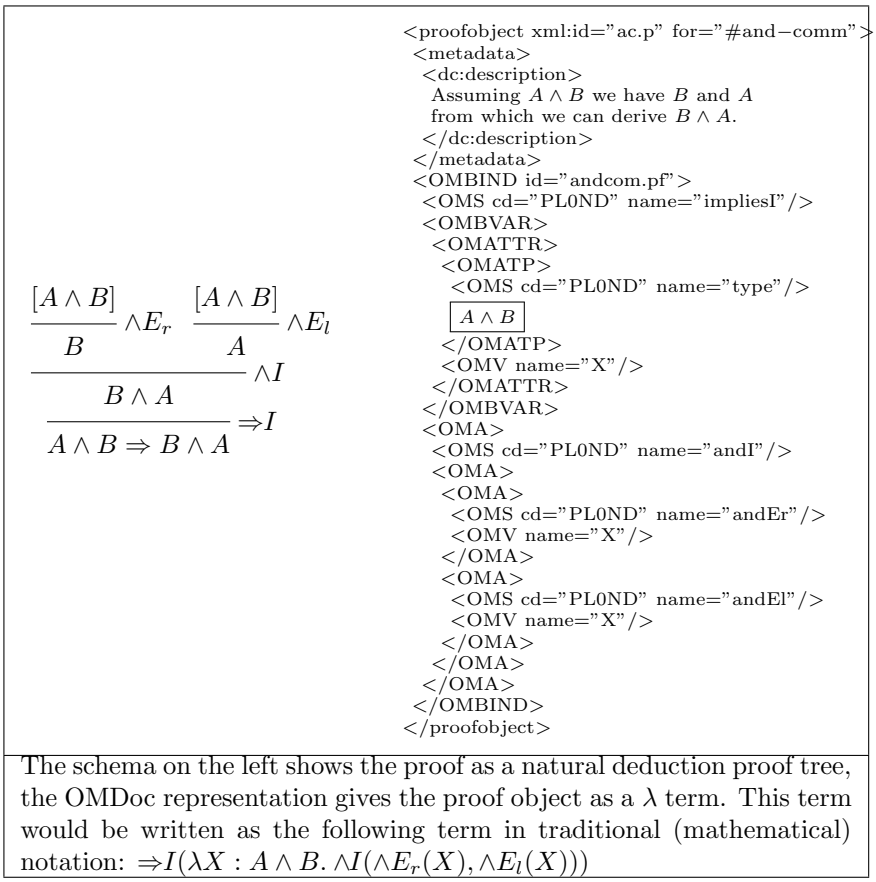
Listing 53.1: A Theory for Propositional Natural Deduction

```
<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">
    <h:p>Conjunction introduction, if we can derive A and B,
      then we can conclude A ∧ B.</h:p>
  </definition>
  ...
</theory>
```

In particular, it is possible to use a `definition` element to define a derived inference rule by simply specifying the proof term as a definiens:

```
<symbol name="andcom">
  <metadata><dc:description>Commutativity for ∧</dc:description></metadata>
  <type system="prop-as-types">(A ∧ B) → (B ∧ A)</type>
</symbol>
```



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

```
<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>
 <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>
</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: $\Rightarrow I(\lambda X : A \wedge B.\, \wedge I(\wedge E_r(X), \wedge E_l(X)))$

Figure 53.1: A Proof Object for the Commutativity of Conjunction

```
<definition xml:id="andcom.def" for="#andcom" type="simple">
  <OMR href="#andcom.pf"/>
</definition>
```

Like `proof`s, `proofobject`s elements are considered to be non-assertional in OMDoc, since they do not make assertions about mathematical objects themselves, but only justify such assertions.

Part XXVI

Complex Theories (Modules CTH and DG)

In Chapter 30 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 Chapter 30 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.

Element	Attributes		M	Content
	Required	Optional	D	
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, type, hiding, consistency, exhaustivity	−	requation*, measure?, ordering?
inclusion	via	xml:id, conservativity, conservativity-just	−	EMPTY
theory-inclusion	from, to	xml:id, class, style, conservativity, conservativity-just	+	(CMP*,FMP*, morphism, obligation*)
axiom-inclusion	from, to	xml:id, class, style, conservativity, conservativity-just	+	morphism?, obligation*

Figure 53.2: Complex Theories in OMDoc

Chapter 54

Inheritance via Translations

[=morphisms]

Literal inheritance of symbols is often insufficient to re-use mathematical structures and theories efficiently. Consider for instance the situation in the elementary algebraic hierarchy: for a theory of rings, we should be able to inherit the additive group structure from the theory `group` of groups and the structure of a multiplicative monoid from the theory `monoid`: A ring is a set R together with two operations $+$ and $*$, such that $(R,+)$ is a group with unit 0 and inverse operation $-$ and $(R^*,*)$ is a monoid with unit 1 and base set $R^*:=\{r\in R\big|r\neq 0\}$. 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 54.1).

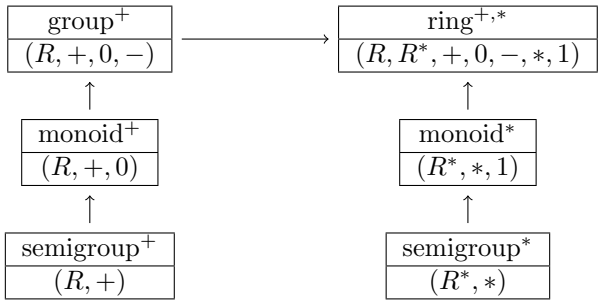


Figure 54.1: A Theory of Rings via Simple Inheritance

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 54.2 shows a theory graph that defines a theory of rings by importing the monoid axioms via the morphism σ. With this translation, we do not have to duplicate the `monoid` and `semigroup` theories and can even move the definition of $\cdot^*$ operator into the theory of monoids, where it intuitively belongs[2].

Formally, we extend the notion of inheritance given in Chapter 30 by allowing a target theory to import another a source theory **via a morphism** : Let $\mathcal{S}$ be a theory with theory-constitutive

[1] which seems negligible in this simple example, but in real life, each instance of multiple inheritance leads to a *multiplication* of all dependent theories, which becomes an exponentially redundant management nightmare.

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

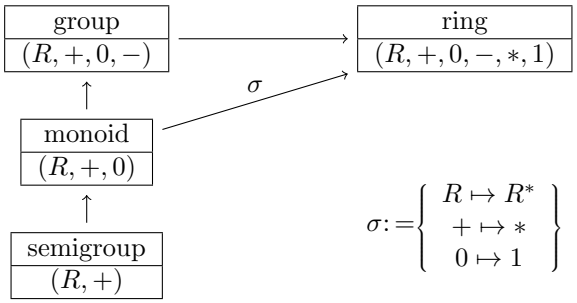


Figure 54.2: A Theory of Rings via Morphisms

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 $\sigma(\forall x.x + 0 = x) = \forall x.x * 1 = x$.

`morphism`

Definition 54.0.1 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 27.3. The optional attribute `type` allows to specify whether the function is really recursive (value `recursive`) or pattern-defined (value `pattern`).

As in the case of the `definition` element, termination of the defined function can be specified using the optional child elements `measure` and `ordering`, or the optional attributes `uniqueness` and `existence`, which point to uniqueness and existence assertions. Consistency and exhaustivity of the recursive equations are specified by the optional attributes `consistency` and `exhaustivity`.

Listing 54.1 gives the OMDoc representation of the theory graph in Figure 54.2, assuming the theories in Listing 31.2.

Listing 54.1: A Theory of Rings by Inheritance Via Renaming

```
<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>
        <OMS cd="monoid" name="set"/>
        <OMA><OMS cd="monoid" name="setstar"/>
          <OMS cd="semigroup" name="set"/>
        </OMA>
      </requation>
      <requation>
        <OMS cd="monoid" name="op"/>
        <OMS cd="ring" name="times"/>
      </requation>
      <requation>
        <OMS cd="monoid" name="neut"/>
        <OMS cd="ring" name="one"/>
      </requation>
    </morphism>
  </imports>
  <axiom xml:id="ring.distribution">
    <CMP><OMS cd="semigroup" name="op"/> distributes over
      <OMS cd="ring" name="times"/>
    </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

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

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 [Mos04b; MAH06] for details. The definition of hiding used there is more general. The variant used here arises as the special case where the hiding morphism, which goes against the import direction, is an inclusion; then the symbols that are not in the image are the hidden ones. If we restrict ourselves to hiding defined symbols, then the situation becomes simpler to understand: A morphism that hides a (defined) symbol s will translate the theory-constitutive elements of the source theory by expanding definitions. Thus s will not be present in the target theory, but all the contributions of the theory-constitutive elements of the source theory will have been inherited. Say, we want to define the concept of a sorting function, i.e. a function that — given a list L as input — returns a returns a permutation L' of L that is ordered. In the situation depicted in Figure 56.1, 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.

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).

Chapter 55

Postulated Theory Inclusions

[=theory-morphisms]

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[1] to those of $\mathcal{T}$, such that for every theory-constitutive statement $\mathbf{S}$ of $\mathcal{S}$, $\sigma(\mathbf{S})$ is provable in $\mathcal{T}$ (we say that $\sigma(\mathbf{S})$ is a $\mathcal{T}$**-theorem**).

In OMDoc, we weaken this logical property to a structural one: We say that a theory-constitutive statement $\mathbf{S}$ in theory $\mathcal{S}$ is **structurally included** in theory $\mathcal{T}$ via σ, if there is an assertional element $\mathbf{T}$ 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.

Definition 55.0.1 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 Chapter 53 induces a theory inclusion from $\mathcal{S}$ into $\mathcal{T}$[2] (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 Chapter 56). 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).

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

[2] 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.

We call a theory inclusion $\sigma: \mathcal{S} \to \mathcal{T}$ **conservative** , iff $\mathbf{A}$ is already a $\mathcal{S}$-theorem for all $\mathcal{T}$-theorems of the from $\sigma(\mathbf{A})$. If the morphism σ is the identity, then this means the local axioms in $\mathcal{T}$ only affect the local symbols of $\mathcal{T}$, and do not the part inherited from $\mathcal{S}$. In particular, conservative extensions of consistent theories cannot be inconsistent. For instance, if all the local theory-constitutive elements in $\mathcal{T}$ are symbol declarations with definitions, then conservativity is guaranteed by the special form of the definitions. We can specify conservativity of a theory inclusion via the `conservativity`. The values `conservative` and `conservative` are used for the two cases discussed above. There is a third value: `conservative`, which we will not explain here, but refer the reader to [MAH06].

`theory-inclusion`

Definition 55.0.2 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.

`obligation`

Definition 55.0.3 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.

Listing 55.1 shows a theory inclusion from the theory `group` defined in Listing 31.2 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 Chapter 56).

Listing 55.1: 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>
```

Chapter 56

Local- and Required Theory Inclusions

[=restinf]

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 Part IX for a discussion). If these conditions are not met, the theory intuitively does not make sense. So rather than simply stating (or importing) these assumptions as theory-constitutive statements — which would make the theory inconsistent, when they are not met — they can be stated as well-definedness conditions. Usually, these conditions can be posited as theory inclusions, so checking these conditions is a purely structural matter, and comes into the realm of OMDoc's structural methods.

`inclusion`

Definition 56.0.1 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 56.1[1]. 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`.

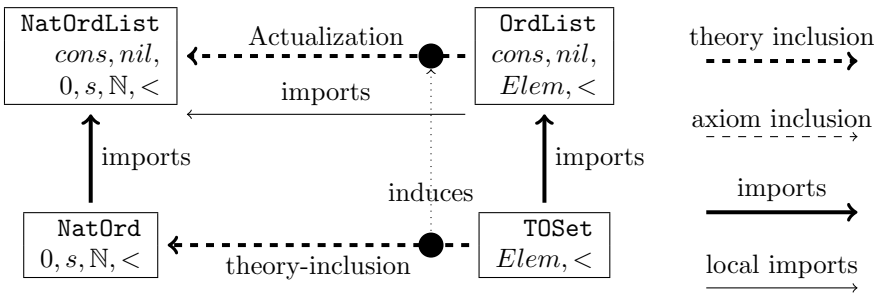


Figure 56.1: A Structured Specification of Lists (of Natural Numbers)

[1]This example is covered in detail in Part IX.

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.

To allow for a local variant of inheritance, the CTH module adds an attribute `type` to the `imports` element. This can take the values `global` (the default) and `local`. In the latter case, only the theory-constitutive statements that are local to the source theory are imported.

`axiom-inclusion`

Definition 56.0.2 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.

Chapter 57

Induced Assertions and Expositions

[=assertionvia]

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 55.1, we might have the following situation:

Listing 57.1: 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 55.1, 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 55.1. 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[1] 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.

[1] 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.

Another example is the presence of different normalization factors in physics or branch cuts in elementary complex functions. In both cases there is a plethora of definitions, which all describe essentially the same objects (see e.g. [Bra+02] for an overview over the branch cut situation). Reading materials that are based on the "wrong" definition is a nuisance at best, and can lead to serious errors. Being able to adapt documents by translating them from the author theory to the user theory by a previously established theory morphism can alleviate both.

Mathematics and science are full of such situations, where objects can be viewed from different angles or in different representations. Moreover, no single representation is "better" than the other, since different views reveal or highlight different aspects of the object (see [KK06] 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.

Chapter 58

Development Graphs (Module DG)

[=dgraph]

The OMDoc module DG for development graphs 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 managementoftheory 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.

58.1 Introduction

As we have seen in the example in Listing 55.1, 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 58.1, 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 58.1 and have the formula-mappings α, β, and γ) are easy to prove[1].

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.

Definition 58.1.1 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 morphism $\tau := \sigma_n \circ \cdots \circ \sigma_1 \circ \sigma$),

[1] 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).

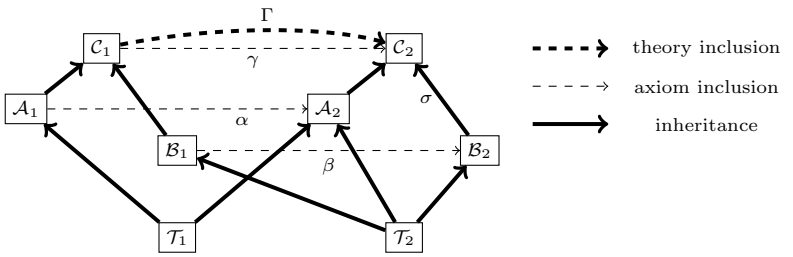


Figure 58.1: A Development Graph with Theory Inclusions

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).

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

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 58.1 — 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[2].

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

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

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 58.2.

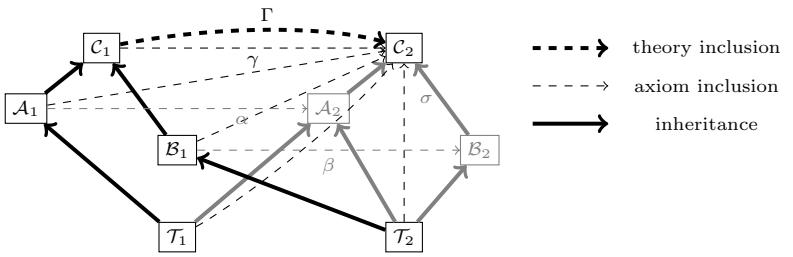


Figure 58.2: A Decomposition for the theory inclusion Γ

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 $\mathcal{X}$ 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.

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

58.2 An OMDoc Infrastructure for Development Graphs (Module DG)

Definition 58.2.1 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. `decomposition`

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`.

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

Figure 58.3: Development Graphs in OMDoc

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.

Definition 58.2.2 A **path-just** element justifies an `axiom-inclusion` by reference to other `axiom-inclusion` or `theory-inclusion` elements. **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 chain, i.e if we have `globals="... #ref1 #ref2 ..."` there must be theory inclusions σ_i with `xml:id="ref`i`"`, such that the target theory of σ_1 is the source theory of σ_2. `path-just`

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 58.1 to 58.2 in OMDoc syntax. Another — more mathematical — example is carried out in detail in Part X.

Listing 58.1: The OMDoc representation of the theories in Figure 58.1.

```
<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 58.2: The OMDoc Representation of the Inclusions in Figure 58.1.

```
<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 `obligation`s to conserve space. From the axiom inclusions, we can now build four more via path justifications:

Listing 58.3: The Induced Axiom Inclusions in Figure 58.1.

```
<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>
```

Part XXVII

Notation and Presentation (Module PRES)

[=presintro] As we have seen, OMDoc is concerned mainly with the content and structure of mathematical documents, and offers a complex infrastructure for dealing with that. However, mathematical texts often carry typographic conventions that cannot be determined by general principles alone. Moreover, non-standard presentations of fragments of mathematical texts 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.

Element	Attributes		Content
	Required	Optional	
omstyle	element	for, xml:id, xref, class, style	(style|xslt)*

Figure 58.4: The OMDoc Elements for Notation Information

The normal (but of course not the only) way to generate presentation from XML documents is to use XSLT style sheets (see Part XXXII for other applications). XSLT [Cla99b] is a general transformation language for XML. XSLT programs (often called **style sheet** s) consist of a set of **template** s (rules for the transformation of certain nodes in the XML tree). These templates are recursively applied to the input tree to produce the desired output.

The general approach 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 Part XXXII, 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.

Chapter 59

Defining Notations

We propose to encode the presentational characteristics of mathematical objects declaratively in *notation definitions* , which are part of the representational infrastructure and consist of "prototypes" (patterns that are matched against content representations) and "renderings" (that are used to construct the corresponding presentations). Note that since we have reified the notations, we can now devise a flexible management process for notations. For example, we can capture the notation preferences of authors, aggregators, and readers and adapt documents to these. We propose an elaborated mechanism to collect notations from various sources and specify notation preferences below.

59.1 Syntax of Notation Definitions

We will now present an abstract version of the presentation starting from the observation that in content markup formalisms for mathematics formulae are represented as "formula trees". Concretely, we will concentrate on OpenMath objects, the conceptual data model of OpenMath representations, since it is sufficiently general, and work is currently under way to unify the semantics of MathML with the one of OpenMath. Furthermore, we observe that the target of the presentation process is also a tree expression: a layout tree made of layout primitives and glyphs, e. g., a presentation MathML or LaTeX expression.

To specify notation definitions, we use the one given by the abstract grammar from Figure 59.1. Here $|$, $[-]$, $-^*$, and $-^+$ denote alternative, bracketing, and non-empty and possibly empty repetition, respectively. The non-terminal symbol ω is used for patterns ϕ that do not contain jokers. Throughout this article, we will use the non-terminal symbols of the grammar as meta-variables for objects of the respective syntactic class.

Intuitions The intuitive meaning of a *notation definition* $ntn = \phi_1, \ldots, \phi_r \vdash (\lambda_1 : \rho_1)^{p_1}, \ldots, (\lambda_s : \rho_s)^{p_s}$ is the following: If an object matches one of the patterns ϕ_i, it is rendered by one of the renderings ρ_i. Which rendering is chosen, depends on the active rendering context, which is matched against the context annotations λ_i; context annotations are usually lists of key-value pairs and their precise syntax is given in Section **??**. The integer values p_i give the output precedences of the renderings, which are used to dynamically determine the placement of brackets.

The *patterns* ϕ_i are formed from a formal grammar for a subset of OpenMath objects extended with named jokers. The jokers $\underline{o}[\phi]$ and $\underline{l}(\phi)$ correspond to $\backslash(\phi\backslash)$ (or $\backslash(.\backslash)$ if ϕ is omitted) and $\backslash(\phi\backslash)^+$ in Posix regular expression syntax ([88]) – except that our patterns are matched against the list of children of an OpenMath object instead of against a list of characters. Here underlined variables denote names of jokers that can be referred to in the renderings ρ_i. We need two special jokers $\underline{s}$ and $\underline{v}$, which only match OpenMath symbols and variables, respectively. The *renderings* ρ_i are formed by a formal syntax for simplified XML extended with means to refer to the jokers

Notation declarations	ntn	::=	$\phi^+ \vdash [(\lambda\colon\rho)^p]^+$
Patterns	ϕ	::=	
Symbols			$\sigma(n,n,n)$
Variables		|	$\upsilon(n)$
Applications		|	$@(\phi[,\phi]^+)$
Binders		|	$\beta(\phi,\Upsilon,\phi)$
Attributions		|	$\alpha(\phi,\sigma(n,n,n)\mapsto\phi)$
Symbol/Variable/Object/List jokers		|	$\underline{s} \mid \underline{\upsilon} \mid \underline{o}[\phi] \mid \underline{o} \mid \underline{l}(\phi)$
Variable contexts	Υ	::=	ϕ^+
Match contexts	M	::=	$[q\mapsto X]^*$
Matches	X	::=	$\omega^*\|S^*\|(X)$
Empty match contexts	μ	::=	$[q\mapsto H]^*$
Holes	H	::=	$\_\|\text{“”}\|(H)$
Context annotation	λ	::=	C^*
Renderings	ρ	::=	
XML elements			$\langle S\rangle\rho^*\langle/\rangle$
XML attributes		|	$S = \text{”}\rho^*\text{”}$
Texts		|	S
Symbol or variable names		|	$\underline{q}$
Matched objects		|	$\underline{q}^p$
Matched lists		|	$\texttt{for}(\underline{q},I,\rho^*)\{\rho^*\}$
Precedences	p	::=	$-\infty\|I\|\infty$
Names	n,s,v,l,o	::=	C^+
Integers	I	::=	integer
Qualified joker names	q	::=	$l/q\|s\|v\|o\|l$
Strings	S	::=	C^*
Characters	C	::=	character except /

Table 59.1: The Grammar for Notation Definitions

used in the patterns. When referring to object jokers, input precedences are given that work together with the output precedences of renderings.

Match contexts are used to store the result of matching a pattern against an object. Due to list jokers, jokers may be nested; therefore, we use qualified joker names in the match contexts (which are transparent to the user). Empty match contexts are used to store the structure of a match context induced by a pattern: They contain holes that are filled by matching the pattern against an object.

Example We will use a multiple integral as an example that shows all aspects of our approach in action.

$$\int_{a_1}^{b_1}\ldots\int_{a_n}^{b_n}\sin x_1 + x_2\; dx_n\ldots dx_1.$$

Let *int*, *iv*, *lam*, *plus*, and *sin* abbreviate symbols for integration, closed real intervals, lambda abstraction, addition, and sine. We intend *int*, *lam*, and *plus* to be flexary symbols, i. e., symbols that take an arbitrary finite number of arguments. Furthermore, we assume symbols *color* and *red* from a content dictionary for style attributions. We want to render into LaTeX the OpenMath object

$$@\big(\,int, @(iv,a_1,b_1),\ldots,@(iv,a_n,b_n),\\ \beta\big(\,lam,\upsilon(x_1),\ldots,\upsilon(x_n),\ \alpha(@(plus,@(sin,\upsilon(x_1)),\upsilon(x_2)),color\mapsto red)\big)\big)$$

as `\int_{`a_1`}^{`b_1`}`$\ldots$`\int_{`a_n`}^{`b_n`}\color{red}{\sin` x_1`+`x_2`}d`$x_n\ldots dx_1$

We can do that with the following notations:

$$@(int, \underline{\texttt{ranges}}(@(iv, \underline{\texttt{a}}, \underline{\texttt{b}})), \beta(lam, \underline{\texttt{vars}}(\underline{\texttt{x}}), \underline{\texttt{f}}))$$
$$\vdash ((format = latex) : \texttt{for}(\underline{\texttt{ranges}})\{\backslash\texttt{int\_\{ }\underline{\texttt{a}}^{\infty}\texttt{ \}\^{}\{ }\underline{\texttt{b}}^{\infty}\texttt{ \}\} }\underline{\texttt{f}}^{\infty}\ \texttt{for}(\underline{\texttt{vars}}, -1)\{\texttt{d }\underline{\texttt{x}}^{\infty}\})^{-\infty}$$
$$\alpha(\underline{a}, color \mapsto \underline{\texttt{col}}) \vdash ((format = latex) : \{\backslash color\{\ \underline{\texttt{col}}\ \}\ \underline{\texttt{a}}^{\infty}\ \})^{-\infty}$$
$$@(plus, \underline{\texttt{args}}(\underline{\texttt{arg}})) \vdash ((format = latex) : \texttt{for}(\underline{\texttt{args}}, +)\{\underline{\texttt{arg}}\})^{10}$$
$$@(sin, \underline{\texttt{arg}}) \vdash ((format = latex) : \backslash sin\ \underline{\texttt{arg}})^{0}$$

The first notation matches the application of the symbol *int* to a list of ranges and a lambda abstraction binding a list of variables. The rendering iterates first over the ranges, rendering them as integral signs with bounds, then recurses into the function body $\underline{\texttt{f}}$, then iterates over the variables, rendering them in reverse order prefixed with d. The second notation is used when $\underline{\texttt{f}}$ recurses into the presentation of the function body $\alpha(@(plus, @(sin, \upsilon(x_1)), \upsilon(x_2)), color \mapsto red)$. It matches an attribution of *color*, which is rendered using the LaTeX color package. The third notation is used when $\underline{\texttt{a}}$ recurses into the attributed object $@(plus, @(sin, \upsilon(x_1)), \upsilon(x_2))$. It matches any application of *plus*, and the rendering iterates over all arguments, placing the separator + in between. Finally, *sin* is rendered in a straightforward way. We omit the notation that renders variables by their name.

The output precedence $-\infty$ of *int* makes sure that the integral as a whole is never bracketed. And the input precedences ∞ makes sure that the arguments of *int* are never bracketed. Both are reasonable because the integral notation provides its own fencing symbols, namely $\int$ and d. The output precedences of *plus* and *sin* are 10 and 0, which means that *sin* binds stronger; therefore, the expression $\sin x$ is not bracketed either. However, an inexperienced user may wish to display these brackets. Therefore, our rendering does not completely suppress them. Rather, we annotate them with an "elision level", which is computed as the difference of the two precedences. This information can then be used by active documents (see section **??**).

Well-formed Notations A notation definition $\phi_1, \ldots, \phi_r \vdash (\lambda_1 : \rho_1)^{p_1}, \ldots, (\lambda_s : \rho_s)^{p_s}$ is well-formed if all ϕ_i are well-formed patterns that induce the same empty match contexts, and all ρ_i are well-formed renderings with respect to that empty match context.

Every pattern ϕ generates an *empty match context* $\mu(\phi)$ as follows:

- For an object joker $\underline{o}[\phi]$ or $\underline{o}$ occurring in ϕ but not within a list joker, $\mu(\phi)$ contains $o \mapsto \_$.
- For a symbol or variable with name n occurring in ϕ but not within a list joker, $\mu(\phi)$ contains $n \mapsto$ "".
- For a list joker $\underline{l}(\phi')$ occurring in ϕ, $\mu(\phi)$ contains
  - $l \mapsto (\_)$, and
  - $l/n \mapsto (H)$ for every $n \mapsto H$ in $\mu(\phi')$.

In an empty match context, a hole $\_$ is a placeholder for an object, "" for a string, $(\_)$ for a list of objects, $((\_))$ for a list of lists of objects, and so on. Thus, symbol, variable, or object joker in ϕ produce a single named hole, and every list joker and every joker within a list joker produces a named list of holes (H). For example, the empty match context induced by the pattern in the notation for *int* above is

$$\texttt{ranges} \mapsto (\_),\ \texttt{ranges/a} \mapsto (\_),\ \texttt{ranges/b} \mapsto (\_),\ \texttt{f} \mapsto \_,$$
$$\texttt{vars} \mapsto (\_),\ \texttt{vars/x} \mapsto (\text{""})$$

A pattern ϕ is well-formed if it satisfies the following conditions:

- There are no duplicate names in $\mu(\phi)$.
- No jokers occur within object jokers.
- List jokers may not occur as direct children of binders or attributions.
- At most one list joker may occur as a child of the same application, and it may not be the first child.
- At most one list joker may occur in the same variable context.

These restrictions guarantee that matching an OpenMath object against a pattern is possible in at most one way. In particular, no backtracking is needed in the matching algorithm.

Assume an empty match context μ. We define well-formed renderings with respect to μ as follows:

- $\langle S\rangle\rho_1,\ldots,\rho_r\langle/\rangle$ is well-formed if all ρ_i are well-formed.
- $S =$ "$\rho_1,\ldots,\rho_r$" is well-formed if all ρ_i are well-formed and are of the form S' or $\underline{n}$. Furthermore, $S =$ "$\rho_1,\ldots,\rho_r$" may only occur as a child of an XML element rendering.
- S is well-formed.
- $\underline{n}$ is well-formed if $n \mapsto$ "" is in μ.
- $\underline{o}^p$ is well-formed if $o \mapsto \_$ is in μ.
- $\texttt{for}(\underline{l}, I, sep)\{body\}$ is well-formed if $l \mapsto (\_)$ or $l \mapsto (\text{""})$ is in μ, all renderings in *sep* are well-formed with respect to μ, and all renderings in *body* are well-formed with respect to μ^l. The step size I and the separator *sep* are optional, and default to 1 and the empty string, respectively, if omitted.

Here μ^l is the empty match context arising from μ if every $l/q \mapsto (H)$ is replaced with $q \mapsto H$ and every previously existing hole named q is removed. Replacing $l/q \mapsto (H)$ means that jokers occurring within the list joker l are only accessible within a corresponding rendering $\texttt{for}(\underline{l}, I, \rho^*)\{\rho^*\}$. And removing the previously existing holes means that in $@(\underline{o}, \underline{l}(\underline{o}))$, the inner object joker shadows the outer one.

59.2 Semantics of Notation Definitions

The rendering algorithm has two levels. The high-level takes as input a document *Doc*, a notation database *DB*, and a rendering context Λ. The intuition of *DB* is that it is essentially a set of notation definitions. In practice this set of notation definitions will be large and highly structured; therefore, we assume a database maintaining it. The rendering context Λ is a list of context annotations that the database uses to select notations, e. g., the requested output format and language. The algorithm outputs *Doc* with OpenMath objects in it replaced with their rendering. It does so in three steps:

1. In a preprocessing step, *Doc* is scanned and notation definitions given inside *Doc* are collected. If *Doc* imports or includes other documents, these are retrieved and processed recursively. And if *Doc* references external notation definitions, these are retrieved as well. Together with the notations already present in the database, these notations form the notation context Π.
2. In a second preprocessing step, Π is normalized by grouping together renderings of the same patterns. Then for each pattern, the triples $(\lambda\colon\rho)^p$ pertaining to it are filtered and ordered according to how well λ matches Λ. This process can also scan for and store arbitrary other information in *Doc* or in *DB*: For example, *Doc* can contain what we call notation tags (see Section **??**) that explicitly relate an OpenMath object and a rendering.

3. *Doc* is traversed, and the low-level algorithm is invoked on every OpenMath object found.

The preprocessing steps are described in detail in Sec. **??**. In the following, we describe the low-level algorithm.

The low-level algorithm takes as input an OpenMath object ω and the notation context Π. It returns the rendering of ω. If the low-level algorithm is invoked recursively to render a subobject of ω, it takes an input precedence p, which is used for bracket placement, as an additional argument.

It computes its output in two steps.

1. ω is matched against the patterns in the notation definitions in Π until a matching pattern ϕ is found. The notation definition in which ϕ occurs induces a list $(\lambda_1 : \rho_1)^{p_1}, \ldots, (\lambda_n : \rho_n)^{p_n}$ of context-annotations, renderings, and output precedences. The first one of them is chosen unless Π contains a notation tag that selects a different one for ω.

2. The output is ${\rho_j}^{M(\phi,\omega)}$, the rendering of ρ_j in context $M(\phi, \omega)$ as defined below. Additionally, if $p_j > p$, the output is enclosed in brackets.

Semantics of Patterns The semantics of patterns is that they are matched against OpenMath objects. Naturally, every OpenMath object matches against itself. Symbol, variable, and object jokers match in the obvious way. A list joker $\underline{l}(\phi)$ matches against a non-empty list of objects all matching ϕ.

Let ϕ be a pattern and ω a matching OpenMath object. We define a match context $M(\phi, \omega)$ as follows.

- For a symbol or variable joker with name n that matched against the sub-object ω' of ω, $M(\phi, \omega)$ contains $n \mapsto S$ where S is the name of ω'.
- For an object joker $\underline{o}[\phi']$, $M(\phi, \omega)$ contains $o \mapsto \phi'$.
- For an object joker $\underline{o}$ that matched against the sub-object ω' of ω, $M(\phi, \omega)$ contains $o \mapsto \omega$.
- If a list joker $\underline{l}(\phi')$ matched a list $\omega_1, \ldots, \omega_r$, then $M(\phi, \omega)$ contains
  - $l \mapsto (\omega_1, \ldots, \omega_r)$, and
  - for every l/q in $\mu(\phi)$: $l/q \mapsto (X_1, \ldots, X_r)$ where $q \mapsto X_i$ in $M(\phi', \omega_i)$.

We omit the precise definition of what it means for a pattern to match against an object. It is, in principle, well-known from regular expressions. Since no backtracking is needed, the computation of $M(\phi, \omega)$ is straightforward. We denote by $M(q)$, the lookup of the match bound to q in a match context M.

Semantics of Renderings If ϕ matches against ω and the rendering ρ is well formed with respect to $\mu(\phi)$, the intuition of $\rho^{M(\phi,\omega)}$ is that the joker references in ρ are replaced according to $M(\phi, \omega) =: M$. Formally, ρ^M is defined as follows.

The rendering is either a string or a sequence of XML elements. We will use $O + O'$ to denote the concatenation of two outputs O and O'. By that, we mean a concatenation of sequences of XML elements or of strings if O and O' have the same type (string or XML). Otherwise, $O + O'$ is a sequence of XML elements treating a string as an XML text node. This operation is associative since consecutive text nodes can always be merged.

- $\langle S \rangle \rho_1 \ \ldots \ \rho_r \langle / \rangle$ is rendered as an XML element with name S. The attributes are those ${\rho_i}^M$ that are rendered as attributes. The children are the concatenation of the remaining ${\rho_i}^M$ preserving their order.
- $S =$ "$\rho_1 \ \ldots \ \rho_r$" is rendered as an attribute with label S and value ${\rho_1}^M + \ldots + {\rho_n}^M$ (which has type text due to the well-formedness).

- S is rendered as the text S.
- $\underline{s}$ and $\underline{v}$ are rendered as the text $M(s)$ or $M(v)$, respectively.
- $\underline{o}^p$ is rendered by applying the rendering algorithm recursively to $M(o)$ and p.
- $\texttt{for}(\underline{l}, I, \rho_1\ \ldots\ \rho_r)\{\rho'_1\ \ldots\ \rho'_s\}$ is rendered by the following algorithm:
  1. Let $sep := {\rho_1}^M + \ldots + {\rho_r}^M$ and t be the length of $M(l)$.
  2. For $i = 1, \ldots, t$, let $R_i := {\rho'_1}^{M^l_i} + \ldots + {\rho'_s}^{M^l_i}$.
  3. If $I = 0$, return nothing and stop. If I is negative, reverse the list R, and invert the sign of I.
  4. Return $R_I + sep + R_{2*I} \ldots + sep + R_T$ where T is the greatest multiple of I smaller than or equal to t.

Here the match context M^l_i arises from M as follows

- replace $l \mapsto (X_1\ \ldots\ X_t)$ with $l \mapsto X_i$,
- for every $l/q \mapsto (X_1\ \ldots\ X_t)$ in M: replace it with $q \mapsto X_i$, and remove a possible previously defined match for q.

Example Consider the example introduced in Section **??**. There we have

$$\begin{aligned}\omega = \ & @\big(\mathit{int}, @(\mathit{iv}, a_1, b_1), \ldots, @(\mathit{iv}, a_n, b_n),\\ & \beta\big(\mathit{lam}, \upsilon(x_1), \ldots, \upsilon(x_n),\ \alpha(@(\mathit{plus}, @(\mathit{sin}, \upsilon(x_1)), \upsilon(x_2)), \mathit{color} \mapsto \mathit{red})\big)\big)\end{aligned}$$

And Π is the given list of notation definitions. Let $\Lambda = (\mathit{format} = \mathit{latex})$. Matching ω against the patterns in Π succeeds for the first notation definitions and yields the following match context M:

$$\texttt{ranges} \mapsto (@(\mathit{iv}, a_1, b_1), \ldots, @(\mathit{iv}, a_n, b_n)),\ \texttt{ranges/a} \mapsto (a_1, \ldots, a_n),$$

$$\texttt{ranges/b} \mapsto (b_1, \ldots, b_n),\ \texttt{f} \mapsto \alpha(@(\mathit{plus}, @(\mathit{sin}, \upsilon(x_1)), \upsilon(x_2)), \mathit{color} \mapsto \mathit{red}),$$

$$\texttt{vars} \mapsto (\upsilon(x_1), \ldots, \upsilon(x_n)),\ \texttt{vars/x} \mapsto (x_1, \ldots, x_n)$$

In the second step, a specific rendering is chosen. In our case, there is only one rendering, which matches the required rendering context Λ, namely

$$\rho = \texttt{for}(\underline{\texttt{ranges}})\{\backslash\texttt{int\_}\{\ \underline{\texttt{a}}^\infty\ \}\hat{}\{\ \underline{\texttt{b}}^\infty\ \}\}\ \underline{\texttt{f}}^\infty\ \ \texttt{for}(\underline{\texttt{vars}}, -1)\{\texttt{d}\ \underline{\texttt{x}}^\infty\})^{-\infty}$$

To render ρ in match context M, we have to render the three components and concatenate the results. Only the iterations are interesting. In both iterations, the separator *sep* is empty; in the second case, the step size I is -1 to render the variables in reverse order.

[=ext]

Part XXVIII

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		M	Content
	Req.	Optional	D	
private		xml:id, for, theory, requires, type, reformulates, class, style	+	data+
code		xml:id, for, theory, requires, type, class, style	+	input?, output?, effect?, data+
input		xml:id, style, class	+	h:p*
output		xml:id, style, class	+	h:p*
effect		xml:id, style, class	+	h:p*
data		format, href, size, original, pto, pto-version	–	<![CDATA[...]]>

Figure 59.1: The OMDoc Auxiliary Elements for Non-XML Data

[1] Compare Part XII in the OMDoc Primer.

Chapter 60

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 Part XXXVI 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.

Definition 60.0.1 For this kind of data, OMDoc provides the **private** and **code** elements. As the name suggests, the latter is intended for program code[1] and the former for system-specific data that is not program code.

`private`

`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 Chapter 30) 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.

`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

[1] 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.

instance, the knowledge in the assertion in Listing 60.1 can be used as an algebraic simplification rule in the ANALYTICA theorem prover [Cla+03] based on the MATHEMATICA computer algebra system.

The `private` and `code` elements contain an optional `metadata` element and a set of `data` elements that contain or reference the actual data.

Listing 60.1: Reformulating Mathematical Knowledge

```
<assertion xml:id="ALGX0">
  <h:p>If a, b, c, d are numbers, then we have a + b(c + d) = a + bc + bd.</h:p>
</assertion>
<private xml:id="alg-expr-1" pto="Analytica" reformulates="ALGX0">
 <data format="mathematica-5.0">
   <![CDATA[SIMPLIFYRULES[a_ + b_*(c_ + d_) :> a + b*c + b*d /; NumberQ[b]]]]>
 </data>
</private>
```

`data`

Definition 60.0.2 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`[2], 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 60.2 we use a `private` element to specify data for an image[3] in various formats, which is useful in a content markup format like OMDoc as the transformation process can then choose the most suitable one for the target.

Listing 60.2: 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>
```

Definition 60.0.3 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

`input`

`output`

`effect`

[2] 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

[3] actually Figure 7.1 from Part VI

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 mathematical vernacular marked up in a sequence of `h:p` elements. 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><h:p>None.</h:p></effect>
```

These documentation elements are followed by a set of `data` elements that contain or reference the program code itself. Listing 61.2 shows an example of a `code` element used to store Java code for an applet.

Listing 60.3: 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><h:p>None: the applet handles input itself.</h:p></input>
  <output><h:p>The result of the integration.</h:p></output>
  <effect><h:p>None.</h:p></effect>
</code>
```

Chapter 61

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[1], it seems difficult to support in a content markup format like OMDoc.

The challenge here is to come up with a format-independent representation of the applet functionality, so that the OMDoc representation can be transformed into the specific form needed by the respective presentation format. Most user agents for these presentation formats have built-in mechanisms for processing common data types such as text and various image types. In some instances the user agent may pass the processing to an external application ("plug-ins"). These need information about the location of the object data, the MIME type associated with the object data, and additional values required for the appropriate processing of the object data by the object handler at run-time.

Element	Attributes		M	Content
	Req.	Optional	D	
omlet	data,	xml:id, action, show, actuate, class, style	+	h:p* & param* & data*
param	name	value, valuetype	-	EMPTY

Figure 61.1: The OMDoc Elements for External Objects

Definition 61.0.1 In OMDoc, we use the **omlet** element for applets. It generalizes the HTML applet concept in two ways: The computational engine is not restricted to plug-ins of the browser (we do not know what the result format and presentation engine will be) and the program code can be included in the OMDoc document, making document-centered computation easier to manage. `omlet`

Like the `xhtml:object` tag, the `omlet` element can be used to wrap any text. In the OMDoc context, this means that the children of the `omlet` element can be any elements or text that can occur in the `h:p` 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 61.2) in the result document presentation, and hands off the presentation and interaction with the document in this space to the applet process. The data for the external object is referenced in two possible ways. Either via

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

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 61.2.

The behavior of the external object is specified in the attributes `action`, `show` and `actuate` attributes[2].

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 61.1, where the `data` attribute points to the `private` element in Listing 60.2. For presentation, e.g. as XHTML in a modern browser, this would be transformed into an `xhtml:object` element [The02], whose specific attributes are determined by the information in the `omlet` element here and those `data` children of the `private` element specified in the `data` attribute of the `omlet` that are chosen for presentation in XHTML. If the action specified in the `action` attribute is impossible (e.g. if the contents of the `data` target cannot be presented), then the content of the `omlet` element is processed as a fallback.

Listing 61.1: An `omlet` for an Image

```
1 <omlet data="#legacy" show="embed">A Fragment of Bourbaki's Algebra</omlet>
```

In Listing 61.2 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 61.2: An `omlet` that Calls the Java Applet from Listing 60.3.

```
<omtext xml:id="monp_1">
 <h:p>Let practice integration!</h:p>
  <h:p><omlet data="#callMint" action="execute" style="width:320;height:200">
4       No plug-in found for callMint!
    </omlet></h:p>
</omtext>
```

In this example, the Java applet did not need any parameters (compare the documentation in the `input` element in Listing 60.3).

In the applet in Listing 61.3 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 Part XII.

[2]These latter two attributes are modeled after the XLink [DeR+01] attributes `show` and `actuate`.

Listing 61.3: An `omlet` for Connecting to a Theorem Prover

```
<h:p> Let us prove it interactively:
  <omlet data="#sendtoTP" action="display">
    <param name="timeout" value="30" valuetype="data"/>
    <param name="performative" value="prove"/>
    <param name="problem" value="#ALGX0" valuetype="object"/>
    <param name="description" value="http://example.org/prob17.html" valuetype="ref"/>
    <param name="instance">
     <om:OMA><OMS name="root" cd="arith1"/>
        <om:OMI>3</om:OMI><om:OMI>3</om:OMI>
      </om:OMA>
   </param>
    Sorry, no theorem prover available!
  </omlet>
</h:p>
```

Definition 61.0.2 For parameter passing, we use the **param** elements which specify a set of values that may be required to process the object data by a plug-in at run-time. Any number of `param` elements may appear in the content of an `omlet` element. Their order does not carry any meaning. The `param` element carries the attributes

`param`

`name` This required attribute defines the name of a run-time parameter, assumed to be known by the plug-in. Any two `param` children of an `omlet` element must have different `name` values.

`value` This attribute specifies the value of a run-time parameter passed to the plug-in for the key `name`. Property values have no meaning to OMDoc; their meaning is determined by the plug-in in question.

`valuetype` This attribute specifies the type of the `value` attribute. The value `data` (the default) means that the value of the `value` will be passed to the plug-in as a string. The value `ref` specifies that the value of the `value` attribute is to be interpreted as a URI reference that designates a resource where run-time values are stored. Finally, the value `object` specifies that the `value` value points to a `private` or `code` element that contains a multi-format collection of `data` elements that carry the data.

If the `param` element does not have a `value` attribute, then it may contain a list of mathematical objects encoded as OpenMath, content MathML, or `legacy` elements.

[=quiz]

Part XXIX

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 placeholder for more pedagogically informed developments in the future (see Part XL and [Gog+03] for an example in the OMDoc framework).

Element	Attributes		M	Content
	Req.	Optional	D	
exercise		xml:id, class, style	+	h:p*,hint?,(solution*|mc*)
hint		xml:id, class, style	+	h:p*
solution		xml:id, for, class, style	+	⟪*top-level element*⟫
mc		xml:id, for, class, style	–	choice, hint?, answer
choice		xml:id, class, style	+	h:p*
answer	verdict	xml:id, class, style	+	h:p*

Figure 61.2: The OMDoc Auxiliary Elements for Exercises

Definition 61.0.3 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 as mathematical vernacular in a sequence of `h:p` elements. This information can be augmented by hints (using the `hint` element) and a solution/assessment block (using the `solution` and `mc` elements). `exercise`

The `hint` and `solution` elements can occur as children of `exercise`; or outside, referencing it in their optional `for` attribute. This allows a flexible positioning of the hints and solutions, e.g. in separate documents that can be distributed separately from the `exercise` elements.

Definition 61.0.4 The **hint** element contains mathematical vernacular as a sequence of `h:p` elements for the hint text. The **solution** element can contain 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. `hint` `solution`

Listing 61.4: An Exercise from the TEXBook

```
<exercise xml:id="TeXBook-18-22">
  <h: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:</h:p>

  <h:p style="display:block;font-family:fixed">
    $\{\,p\mid\hbox{$p$ and $p+2$ are prime}\,\}$
  </h:p>

  <h: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
    <h:span style="font-family:fixed">\hbox</h:span> does not vary with the inter-word
    glue in the line that contains it. Explain how the given formula could be
    typeset with line breaks.</p>
  </h:p>
  <hint>
    <h:p>Go back and forth between math mode and horizontal mode.</h:p>
  </hint>
  <solution>
    <h:p>
      <h:span style="font-family:fixed">
       $\{\,p\mid p$~and $p+2$ are prime$\,\}$
      </h:span>,
      assuming that <h:span style="font-family:fixed">\mathsurround</h:span> is
      zero. The more difficult alternative '<h:span style="font-family:fixed">
      $\{\,p\mid p\\ {\rm and}\ p+2\rm\ are\ prime\,\}$</h:span>'
      is not a solution, because line breaks do not occur at
      <h:span style="font-family:fixed">\␣</h:span> (or at glue of any
      kin) within math formulas. Of course it may be best to display a formula like
      this, instead of breaking it between lines.
    </h:p>
  </solution>
</exercise>
```

Multiple-choice exercises (see Listing 61.5) are represented by a group of `mc` elements inside an `exercise` element.

`mc`

Definition 61.0.5 An **mc** element represents a single choice in a multiple choice element. It contains the elements below (in this order).

`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 mathematical vernacular in a sequence of `h:p` elements.

`choice`

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

`answer` for the feedback to the user. This can be the correct answer, or some other feedback (e.g. another hint, without revealing the correct answer). The `verdict` attribute specifies the truth of the answer, it can have the values `true` or `false`. This element is required, inside a `mc`, since the `verdict` is needed. It can be empty if no feedback is available. Furthermore, the **answer** element carries the `xml:id`, `style`, and `class` attributes and contains mathematical vernacular as a sequence of `h:p` elements.

`answer`

Listing 61.5: A Multiple-Choice Exercise in OMDoc

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

[=document-model]

Part XXX

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 Part XV to Part XXVIII, 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.

Chapter 62

XML Document Models

[=XMLDom]

XML supports the task of defining a particular class of documents (e.g. the class of OMDoc documents) with formal grammars such as the document type definition (DTD) or an XML schema, that can be used for mechanical document validation. Surprisingly, XML leaves the task of specifying document equality to be clarified in the (informal) specifications, such as this OMDoc specification. As a consequence, current practice for XML applications is quite varied. For instance, the OpenMath standard (see [Bus+04] and Chapter 19) gives a mathematical object model for OpenMath objects that is specified independently of the XML encoding. Other XML applications like e.g. presentation MathML [Aus+03b] or XHTML [The02] specify models in form of the intended screen presentation, while still others like the XSLT [Cla99b] give the operational semantics.

For a formal definition let $\mathcal{K}$ be a set of documents. We take a **document model** to be a partial equivalence relation[1]. 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 [KK06] for an exploration of possible document models for mathematics). Concretely, OMDoc documents admit the OMDoc document model that we will specify in section Chapter 62 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.

[1] A partial equivalence relation is a symmetric transitive relation. We will use $[d]_{\mathcal{X}}$ for the **equivalence class** of d, i.e. $[d]_{\mathcal{X}} := \{e | d\mathcal{X}e\}$ $\mathcal{X}$ on documents, such that $\{d | d\mathcal{X}d\} = \mathcal{K}$

[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

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 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 Chapter 62.

To get a feeling for the issues involved, let us compare the OMDoc elements in Listings 62.1 to 63.1 below. For instance, the serialization in Listing 62.2 is XML-equal to the one in Listing 62.1, but not to the one in Listing 63.1.

Listing 62.1: An OMDoc Definition

```
<docalt>
  <definition xml:id="comm.def.en" for="comm">
    <h:p xml:lang="en">
      An operation <OMV name="op" id="op"/> is called commutative, iff
      <OMA id="comm1"><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>
      for all <OMV id="x" name="X"/> and <OMV id="y" name="Y"/>.
    </h:p>
  </definition
  <definition xml:id="comm.def.de" for="#comm">
    <h:p xml:lang="de">
      Eine Operation <OMR href="#op"/> heißt kommutativ, falls
      <OMR href="#comm1"/> für alle <OMR href="#x"/> und <OMR href="#y"/>.
    </h:p>
  </definition>
</docalt>
```

Listing 62.2: An XML-equal serialization for Listing 62.1

```
<definition for="comm" xml:id="comm.def.de" >
 <h:p xml:lang='de'> <!-- Note the unabbreviated empty element -->
   Eine Operation <OMR href="#op"/> heißt
   kommutativ, falls <OMR href='comm1'/> für alle
   <OMR href="#x"/> und <OMR href='y'/>.
  </h:p>
</definition>
```

Chapter 63

The OMDoc Document Model

[=omdom]

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 children of an `docalt` element is irrelevant, since they form a multilingual group which form the base for multilingual text assembly. Other facets of the OMDoc document model are motivated by presentation-independence, for instance the distribution of whitespace is irrelevant even in text nodes, to allow formatting and reflow in the source code, which is not considered to change the information content of a text.

Listing 63.1: An OMDoc-Equal Representation for Listings 62.1 and 62.2

```
<docalt>
  <definition xml:id="comm.def.de" for="comm">
    <h:p xml:lang="de">Eine Operation <OMR href="#op"/>
      heißt kommutativ, falls
      <OMA id="comm1"><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>
      für alle <OMR href="#x"/> und <OMR href="#y"/>.
  </h:p>
  </definition>
  <definition xml:id="comm.def.en" for="#comm">
    <h:p xml:lang="en">
      An operation <OMV id="op" name="op"/> is called commutative,
       iff <OMR href="#comm1"/> for all <OMV id="x" name="X"/> and
      <OMV id="y" name="Y"/>.
    </h:p>
  </definition>
</docalt>
```

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:OMA` with `xml:id="comm1"` in Listing 63.1 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 63.1. We have discussed a machine-readable form of these equality constraints in the XML schema for OMDoc in [KA03].

The last rule in Figure 63.1 is probably the most interesting, as we have seen in Part XX, OMDoc documents have both formal and informal aspects, they can contain *narrative* as well as *narrative-structured* information. The latter kind of document contains a formalization of a mathematical theory, as a reference for automated theorem proving systems. There, logical dependencies play a

#	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.	`requation` `dc:description` `sortdef` `data` `dc:title` `solution`
3	DAG encoding	Directedacyclicgraphs 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`

Figure 63.1: The OMDoc Document Model

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 Part XXI)

Chapter 64

OMDoc Sub-Languages

[=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 ?all? 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 ?all? 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 64.1 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.

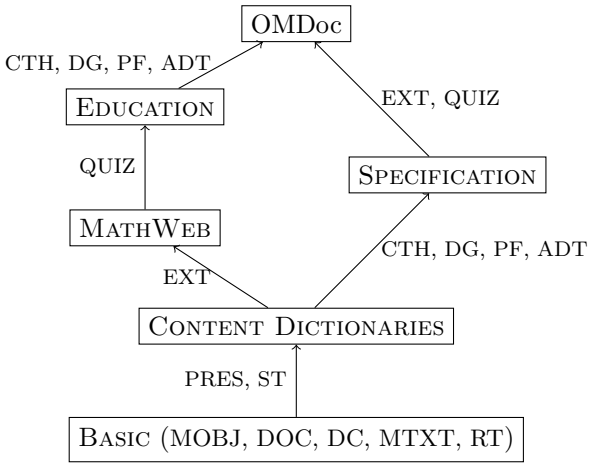


Figure 64.1: OMDoc Sub-Languages and Modules

The sub-language identifiers can be used as values of the `modules` attribute on the `omdoc` and `omdoc` elements. Used there, they abbreviate the list of modules these sub-languages contain.

64.1 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 ?top-level?). 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.

64.2 OMDoc Content Dictionaries

Content Dictionaries are used to define the meaning of symbols in the OpenMath standard [Bus+04], they are the mathematical documents referred to in the `cd` attribute of the `om:OMS` element. To express contentdictionaries 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 Chapter 30).

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 Part VII, 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.

64.3 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 [Mos04b]) offer the necessary infrastructure, but have a syntax that is not integrated with web standards.

The Specification OMDoc sub-language adds the modules ADT and CTH to the language of OMDoc content dictionaries. The resulting language is equivalent to the Casl standard, see [Aut+00a; 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.

64.4 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.

64.5 Educational OMDoc

OMDoc is currently used as a content-oriented basis for various systems for mathematics education (see e.g. Part XI 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.

64.6 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 [**WalMue:dtdg99**] or XHTML [The02] could be extended with the OMDoc statement level. Including modules MOBJ, DC, and (parts of) MTXT, but not RT and DOC would result in content formats that mix the document-level structure of these formats. Another example is the combination of XML-RPC envelopes and OMDoc documents used for interoperability in Part XII.

Part XXXI

Processing OMDoc

In documentwe will present an abstract library for processing collections of OMDoc documents.

Chapter 65

OMDoc resources

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

65.1 The OMDoc Web Site, Wiki, and Mailing List

The main web site for the OMDoc format is `http://www.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 [Kohb].

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

65.2 The OMDoc distribution

All resources on the OMDoc web site are available from the MathWeb Subversion repository [] 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 [**omdoc.org:svn**] for a general introduction to the setup. The resources for version 1.6 of the OMDoc format which is described in this book are accessible on the web at `https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.6` via a regular web browser[1]. 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/branches/omdoc-1.6
```

This will create a directory `omdoc-1.6`, with the sub-directories

[1] Ongoing development of the OMDoc format can be accessed via the head revision of the repository at `https://svn.omdoc.org/repos/omdoc/trunk`.

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 ?? 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`.

65.3 The OMDoc bug tracker

`MathWeb.org` supplies a BugZilla bug-tracker [05a] 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 [05b], 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 65.1).

65.4 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 V1.6//EN`) into absolute URLs like `http://www.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[2] using a `nextCatalog` element of the form

```
1 <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).

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

65.5 External Resources

The OMDoc format has been used on a variety of projects. Chapter **??** 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.html`. These projects have contributed tools, code, and documentation to the OMDoc format, often stressing their special vantage points and applications of the format.

[=validating]

Part XXXII

Validating OMDoc Documents

In Chapter **??** 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[3], 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 XIV 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 **??** and **??**. 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 Presentation-MathML, 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[4] to be RelaxNG-valid.

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 Part LIV for reference.

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

[4]An OMDoc document is called **normalized** , iff all required attributes are present. Given a DTD or XML schema that specifies default values, there are standard XML tools for XML-normalization that can be pipelined to allow RelaxNG validation, so this is not a grave restriction.

65.6 Validation with Document Type Definitions

The OMDoc document type definition [Kohc] can be referenced by the public identifier `"-//OMDoc//DTD OMDoc V1.6//EN"` (see Section 65.4). 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 stand-alone 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 V1.6//EN"
                       "http://www.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 65.4). 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[5] may be conserved by dropping it.

65.6.1 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;">
```

in the DTD. This one makes the abbreviation `%assertiontype;` available for the string "`theorem|proposition|lemma` (in the DTD of the document in Listing 65.1). 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 65.1 for an example, which modifies the parameter entity `%otherassertiontype;` 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 65.1: A Document Type Declaration with Internal Subset

```
<!DOCTYPE omdoc PUBLIC "-//OMDoc//DTD OMDoc V1.6//EN"
                       "http://www.omdoc.org/dtd/omdoc.dtd"
  [<!ENTITY % otherassertiontype "observation">]>
...
<assertion type="observation">...</assertion>
...
```

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

65.6.2 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[6] 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 65.2.

Listing 65.2: A normalized OMDoc document without DTD

```
<?xml version="1.0" standalone="yes"?>
<omdoc xml:id="something" version="1.6" xmlns="http://omdoc.org/ns">
 ...
</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.

65.6.3 Modularization

In OMDoc1.2 the DTD has been modularized according to the W3C conventions for DTD modularization [Alt+10]. This partitions the DTD into DTD modules that correspond to the OMDoc modules discussed in Part XIV 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 **??**) has been deselected, presumably, as the document does not contain proofs.

```
<!DOCTYPE omdoc PUBLIC "-//OMDoc//DTD OMDoc V1.6//EN"
                       "http://www.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 **??**, 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 [Alt+10], 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 modules ⟪*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.

65.6.4 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 [Alt+10] 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

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

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 [Alt+10] 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 [BHL99] 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://omdoc.org/ns">
...
</omdoc:omdoc>
```

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

```
<!DOCTYPE omdoc:omdoc PUBLIC "-//OMDoc//DTD OMDoc V1.6//EN"
                             "http://www.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://omdoc.org/ns`	`omdoc:`

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

65.7 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 Part LIV. Just as in the case for the OMDoc DTD, we provide schemata for the OMDoc sub-languages discussed in Section **??**. 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[7]; thus validation tools (see `http://relaxng.org` for an overview) have to be given a grammar as

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

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 [**Stallman:em02**] for editing XML files, as it provides powerful RelaxNG-based editing support (validation, completion, etc.).

65.8 Validation with XML Schema

For validation[8] with respect to XML schemata (see []) we provide an XML schema for OMDoc [Kohe], which is generated from the RelaxNG schema in Part LIV via the trang system described above. Again, schemata for the sub-languages discussed in Section **??** 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 65.3 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 65.3: 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://omdoc.org/ns
                           http://omdoc.org/xsd/omdoc.xsd">
 ...
</omdoc>
```

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

Part XXXIII

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 [Cla99b] 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 Part LI 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 65.2) or on the web at [Kohf].

65.9 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 ??, 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 Content-MathML 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 ??) before it generates the templates[9].

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[10]. 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 65.1: 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 65.11), 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`.

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.

[9] 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 ??, but hampers mixing presentation information from multiple sources. An implementation based on modes would probably have advantages in this direction in the long run.

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

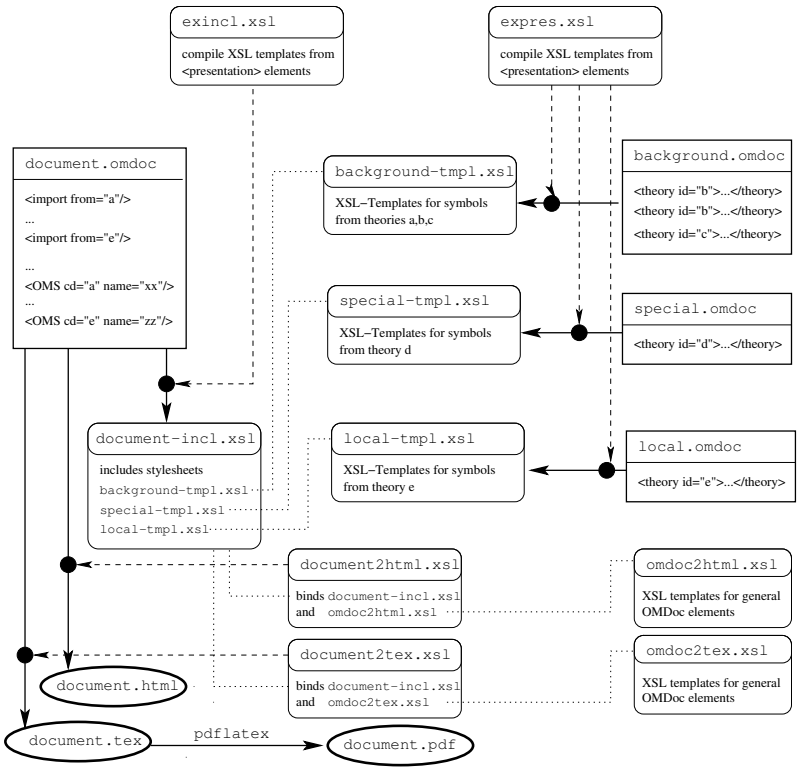


Figure 65.1: The OMDoc presentation Process

65.10 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 **??**, OpenMath and Content-MathML provide a good solution at the level of mathematical objects, which is sufficient for systems like computer algebra systems. OMDoc adds the level of mathematical statements and theories to add support for automated reasoning systems and formal specification systems.

To make practical use of the OMDoc format as an interlingua, we have to support building OMDoc interfaces. An XSLT style sheet is a simple way to come up with (the input half) of an OMDoc interface. A more efficient way would be to integrate an XML 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 65.4: 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> -&gt; </text><recurse select="*[3]"/>
    <text>]</text>
  </style>
  <style format="nuprl">
    <recurse select="*[2]/*"/><text> -&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 [Ben+97], INKA [HS96], PVS [ORS92], *λClam* [RSG98], TPS [And+96] and COQ [Tea] see Section **??**

65.11 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 **??**. 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`.

Part XXXIV

OMDoc Applications and Projects

We present 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; Koh06b; BF06].

Chapter 66

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://www.omdoc.org` for more and current information.

66.1 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 streams. Some of these systems also have input filters for OMDoc like the ✓`eriFun` described in Part XLIX, but most rely on the OMDoc transformation to their native input syntax described in ?omdoctosys?.

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 **C**Point add-in for MS PowerPoint (Part XLV), the SENTIDO plugin for MOZILLA-based browsers, and the plugin for TEX_{MACS} (Part XLVIII) 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 Part XXXIV 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 Part XLVI for details.

Mathematical Knowledge Bases The MBASE and MAYA systems described in Part XXXVI and Part XLIII are web-based mathematical knowledge bases that offer the infrastructure for a universal, distributed repository of formalized mathematics represented in the OMDoc format.

[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.

66.2 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 ?mtxt?).

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 Part XXXVIII to Part XL 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]).

Part XXXV

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

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.[2] For a more detailed history see [Gonb].

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 66.1 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

[2] This offers an alternative to the OQMath wrapper mentioned in Part XL.

```
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'?>
<omdoc xmlns='http://omdoc.org/ns' version='1.6'
       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>1994-04-04T18:43:00+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>
```

Figure 66.1: A minimal QMath document (top left) and its OMDoc result (bottom). Some symbol definitions are displayed in the top right.

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.

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 "$".

While QMath was a good improvement over manual typing of the OMDoc XML, it does not

```
<?xml version='1.0' encoding='UTF-8' standalone='no'?>
<?QMath
:en
Context: "Mathematics/Arithmetic"
?>
<omdoc xmlns='http://omdoc.org/ns' 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>
```

Figure 66.2: The same example document, using QMath only for the formulas.

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 ?omdocmode? or the OQMath mode for jEdit in Part XLI. Another is to use the Sentido browser/editor in Part XXXV, which reimplements and extends QMath's functionality.

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

Part XXXVI

Sentido: an Integrated Environment for OMDoc

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.

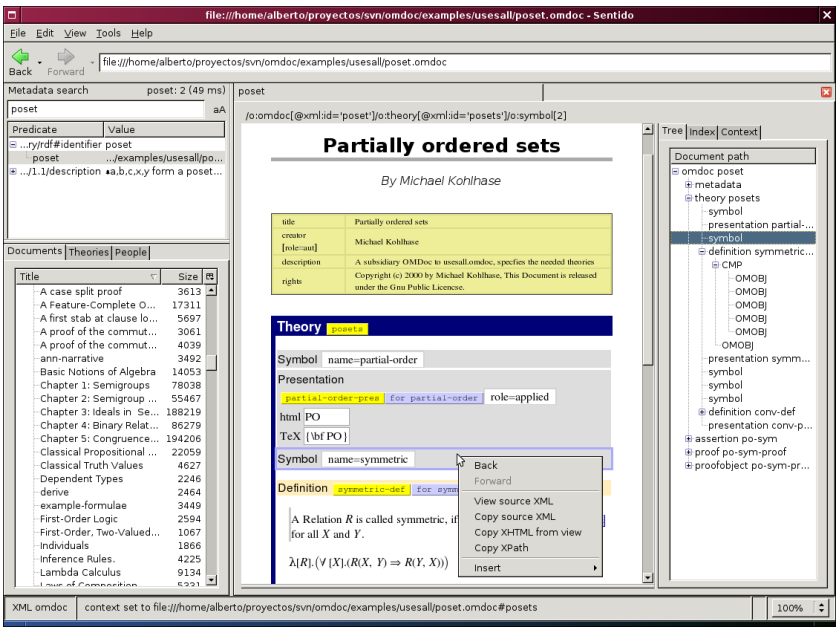


Figure 66.3: Sentido after indexing the OMDoc repository in the library (left) and loading a document from it (center and right).

Figure 66.3 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.

Chapter 67

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[1] 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[2]. The search pattern accepts regular expression syntax, as shown in Figure 67.1.

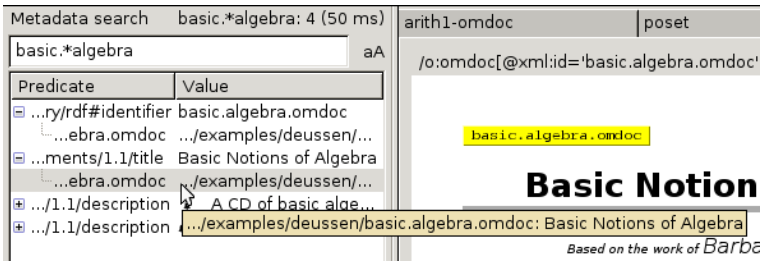


Figure 67.1: Metadata Search in SENTIDO: tooltips show the content of the cropped entries.

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 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

[1] "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."

[2] 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.

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 67.2).

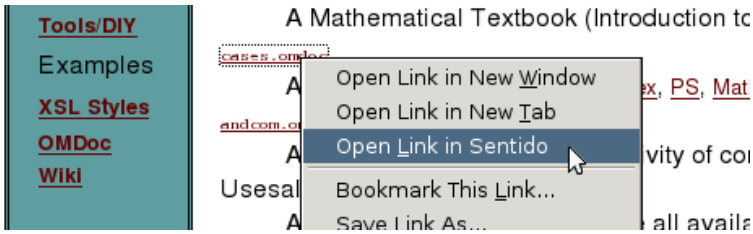


Figure 67.2: MOZILLA's Context Menu after Installing SENTIDO.

Chapter 68

Formula Editing

Mathematical expressions are entered using a selectable linear syntax, translated by a new version of the QMath parser described in Part XXXIV. 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.

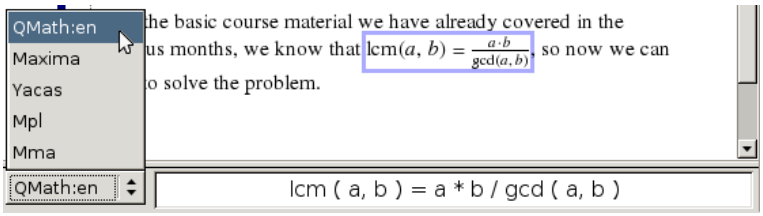


Figure 68.1: The formula editor under the document view, with the input syntax menu and the text field where the formula is typed, which updates continuously the internal OpenMath representation and the MathML view.

When the cursor enters a formula, the linear input field appears at the bottom of the document view, as seen in Figure 68.1. 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 68.2).

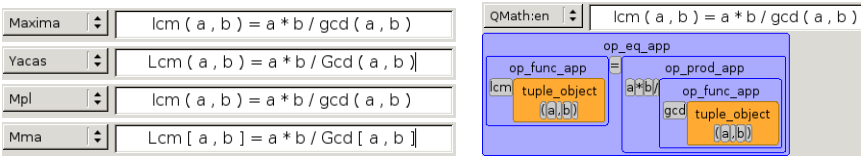


Figure 68.2: The formula is translated by Sentido each time the user selects another syntax (left, the vertical line is the blinking caret), and it is possible to view the parse tree (right), updated as the input is modified.

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®.

Chapter 69

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[1] 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 [Gona], 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]).

[1] Some of those early experiments with Mozilla inspired work done on adapting OpenOffice and TEXMACS for OMDoc in collaboration with George Goguadze [GG03]

Part XXXVII

MBase, an Open Mathematical Knowledge Base

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 [96] 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.

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 69.1). 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 [] 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 [G S95], of which mozart (see []) 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. for specialized searching (see [Duc98] for a related system and the origin of this idea).

MBase supports transparent distribution of data among several MBase servers (see [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

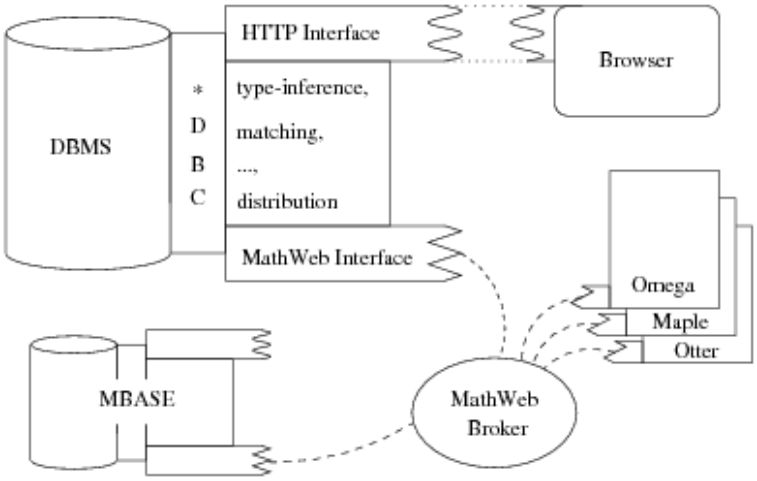


Figure 69.1: System Architecture

data they contain: *archive servers* contain data that is referred to by other MBases, and *scratchpad* 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.

Part XXXVIII

A Search Engine for Mathematical Formulae

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.

Chapter 70

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 72.1 in Chapter 71). 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.

Chapter 71

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 [Gro+03] 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.

Chapter 72

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 [Aus+03b] 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 72.1. Note that we do not know the integration limits or whether the formula is complete or not.

Listing 72.1: 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>
```

Chapter 73

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 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.

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:
$eq(f(x), y)$

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 72.1) in the query $\int_?^? s^2(t)dt$ is made generic, therefore the query would also find the variant $\frac{1}{T}\int_0^T s^2(x)dx = \Sigma_{k=-\infty}^{\infty} \mid c_k \mid^2$.

Chapter 74

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.

Chapter 75

Case Studies and Results

We have tested our implementation on the content repository of the ConneXions Project, available via the OAI protocol [Lag+02]. 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.

Part XXXIX

Semantic Interrelation and Change Management

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 [Kri+03; Kri+04a; Kri+04b]).

Chapter 76

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 MMiSS LaTeX 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.

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 MMiSS LaTeX `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.

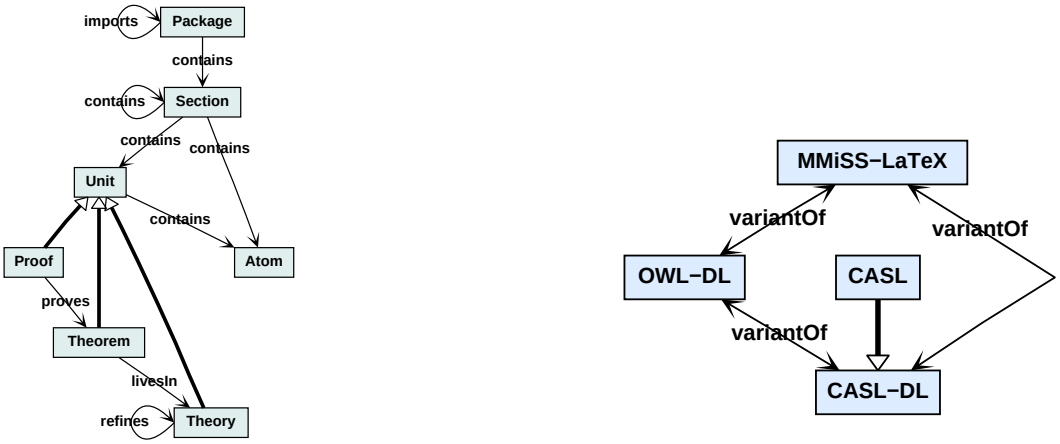


Figure 76.1: (a) Parts of the System's Ontology (b) Formalism variants

Chapter 77

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. 76.1a). 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 "light-weight" 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 ?hets?).

Chapter 78

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 76.1b shows the possible variants for declaring ontology components (see [M+04] 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.

Chapter 79

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.

Part XL

MathDox: Mathematical Documents on the Web

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.

Chapter 80

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. []).

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].

Chapter 81

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 [**WalMue:dtdg99**] 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 [Bus+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 [Fra+99]. 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.

Chapter 82

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 82.1. We explain in some detail the main components shown in this picture.

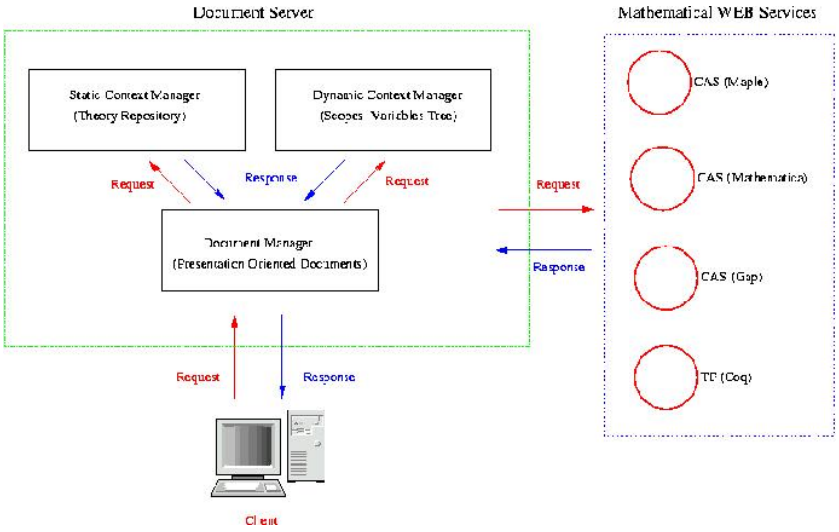


Figure 82.1: The MathDox software

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 [] inside a web server. It is not a monolithic entity. As shown in Figure 82.1, 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 []. The mechanism extends the phrasebook set-up for OpenMath [Cap+00; CCR00]. For constructing specific OpenMath services, we employ our Java OpenMath library ROML [].

Chapter 83

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.

Part XLI

OMDoc in ActiveMath

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.

Chapter 84

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 learningobjects [1], 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 84.1. It includes the XML-RPC web communication protocol for its simplicity and support. In addition, an event framework enables the asynchronous messaging for any changes.

A complex subsystem in its own right is ActiveMath's exercise subsystem [GGM05] 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.

84.1 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

[1] 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.

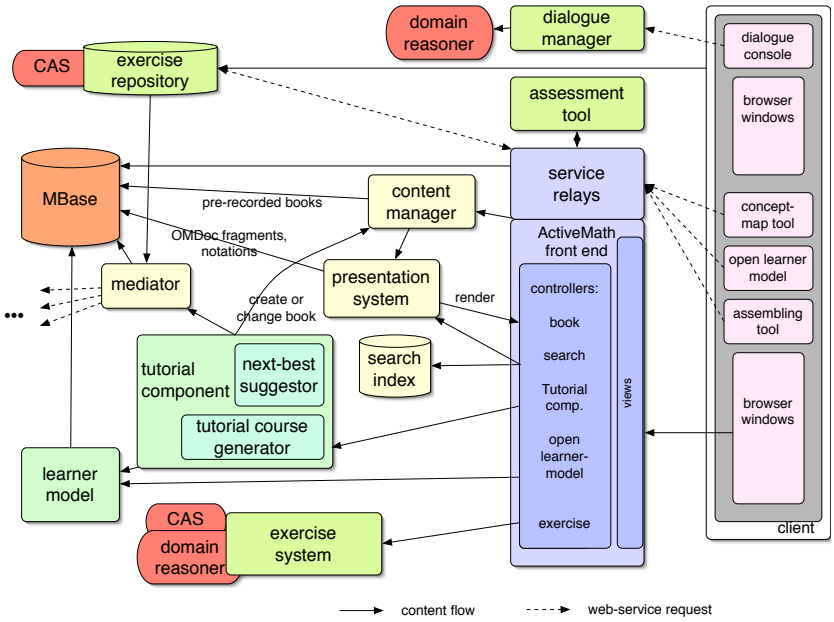


Figure 84.1: The Components, Services and Information Flow in ActiveMath

the Semantic Web coin. Hence, the developments for ActiveMath also include the reuse and interoperability of components and tools [Mel+06].

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

Chapter 85

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 [Gog+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) [Kli+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 [MG04b]. 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 [MG04b]).

The new exercise representation of ActiveMath was the basis for extending the Math QTI standard [MG04a]. 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 [GGM05].

Chapter 86

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.

86.1 Computer Algebra Services

Computer algebra system (CAS) — currently Yacas [], Maxima [], and Wiris [] — 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.

86.2 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 [Ull+04]. 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 [Man+06]. These special presentations are integrated into the presentation process upon request.

86.3 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.

86.4 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.

86.5 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.

86.6 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 Part XLI.

Part XLII

Authoring Tools for ActiveMath

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.

Chapter 87

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.

Chapter 88

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.[1] 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.

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

Part XLIII

SWiM – An OMDoc-based Semantic Wiki

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 [Sch+06].

Chapter 89

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 [Völ+; 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 [Völ+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.

Chapter 90

Design of SWiM

90.1 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.[1] Users are strongly encouraged to define at most one theory per wiki page and to roll out non-constitutive statements (see Chapter 25) to separate pages, referencing their context theory. As constitutive 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 [**URL:jena:web**]. 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:

- 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

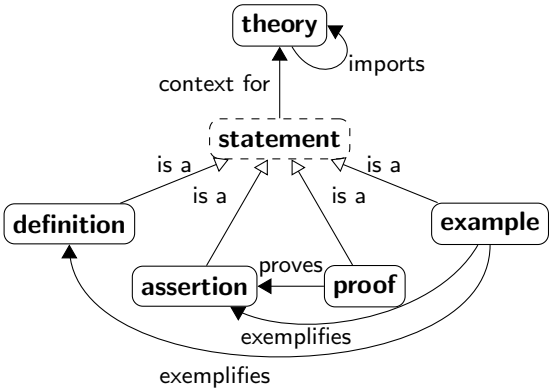


Figure 90.1: Subset of OMDoc's system ontology

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

[1] 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.

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.

90.2 User Interface and Interaction Model

Pages can be rendered to XHTML plus presentational MathML using the transformations described in Part XXXII. 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 Part X. 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 90.2. 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.

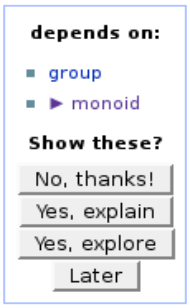


Figure 90.2: The command buttons to navigate along the dependencies

Chapter 91

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 Part XXXIV) 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.

Part XLIV

Maya: Maintaining Structured Developments

Chapter 92

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 [Mos04b] or VSE-SL [Aut+00b]. 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[1]) representation of structured specifications, and which provide the logical basis for the *Complex theories* and *Development graphs* of OMDoc[2]. Relying on development graphs enables MAYA to support the use of various (structured) specification languages like OMDoc, CASL [Mos04b], and VSE-SL [Aut+00b] 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 [Aut+99].

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

[1] This includes, for instance, that it does not require a particular logic (see e.g. [MAH06] for more details).
[2] These are the modules CTH and DG, respectively.

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.

Chapter 93

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 93.1 shows the graphical user interface of MAYA. The right hand side shows the development graph in MAYA for a formalisation of groups. The formalisation was given in OMDoc and imported into MAYA from the OMDoc database MBASE (see [KF01] and Part XXXVI).

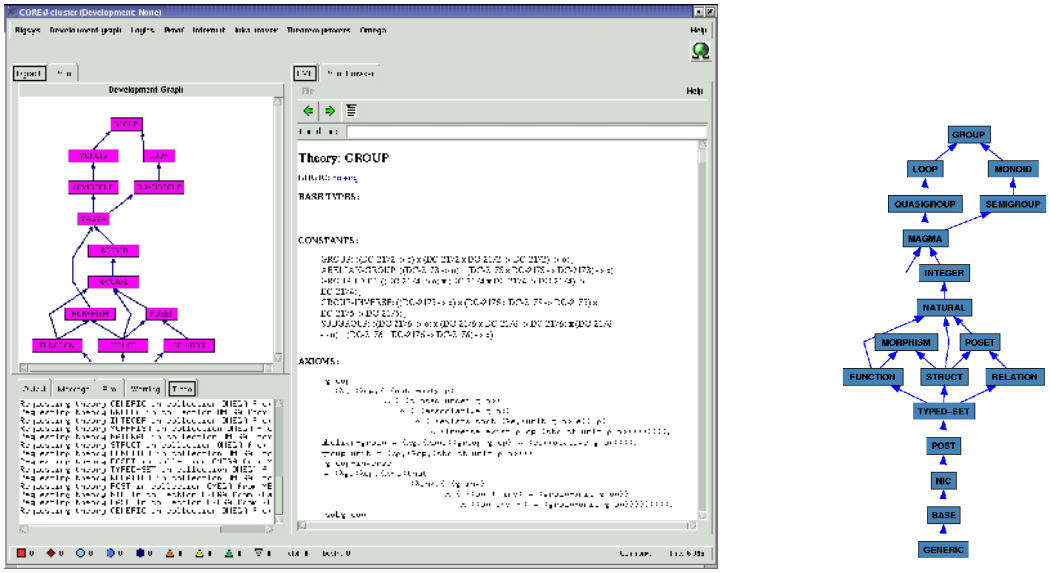


Figure 93.1: The graphical user interface of MAYA & the development graph for the OMDoc Representation of Groups in MBASE

Chapter 94

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).

Chapter 95

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.

Chapter 96

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.

Chapter 97

Conclusion and System Availability

The MAYA-system is mostly implemented in Common Lisp while parts of the GUI, shared with the ΩMEGA-system [Sie+99], are written in Mozart. The CASL-parser is provided by the CoFI-group in Bremen (see ?hets?). The MAYA-system is available from the MAYA-web-page at `http://www.dfki.de/~inka/maya.html`.

The Heterogeneous Tool Set (HETS, see ?hets?) 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.

Part XLV

Hets: The Heterogeneous Tool Set

Chapter 98

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, [**hets06**]) 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, multi-lateral *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.

Chapter 99

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; Mos+05] 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".

Chapter 100

The Architecture of the Hets System

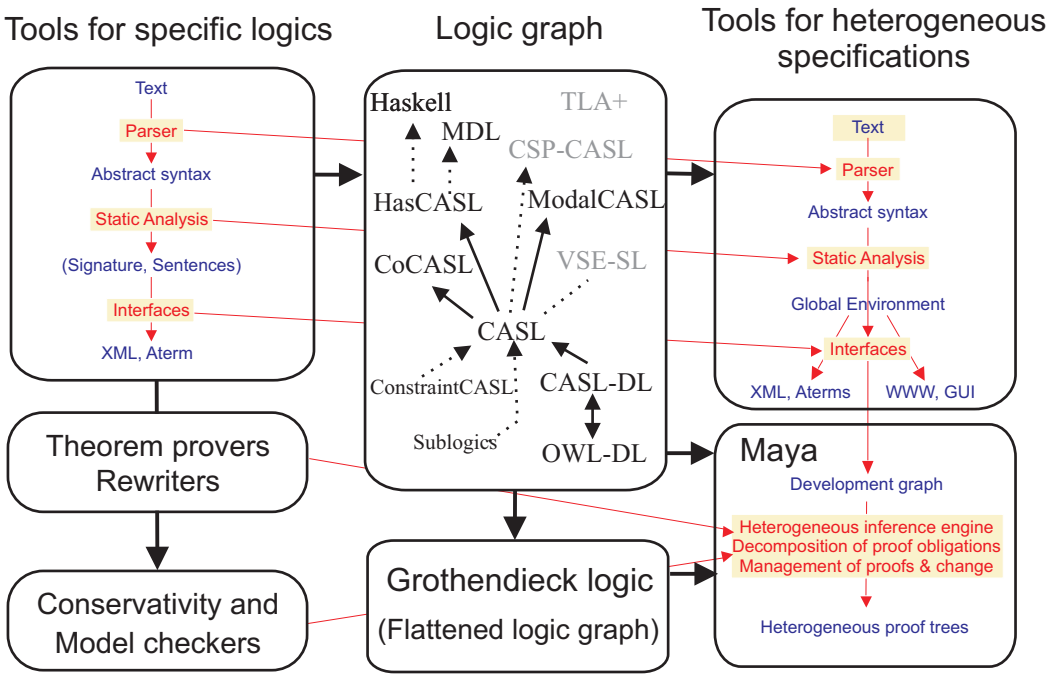


Figure 100.1: Architecture of the heterogeneous tool set

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. 100.1). The graph of currently supported logics and logic translations is shown in Fig. 100.2. 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, and this separation of concerns is essential to keep the tool manageable from a software engineering point of view.

Heterogeneous Casl (HetCasl; see [Mos04a]) includes the structuring constructs of Casl,

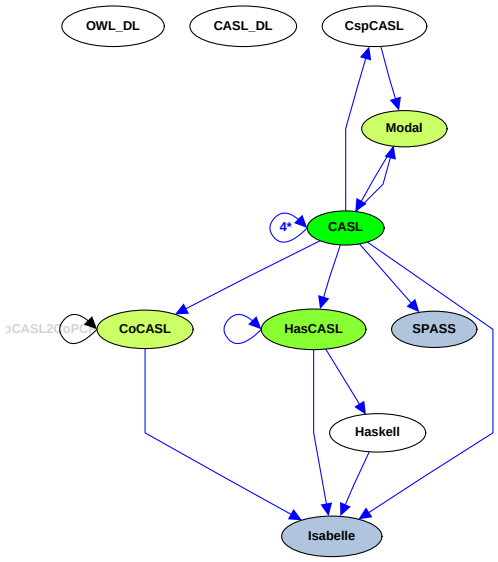


Figure 100.2: Graph of logics currently supported by Hets. The more an ellipse is filled, the more stable is the implementation of the logic.

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 Part XLIII), Hets provides a representation of structured specifications which are the logical basis for the *Complex theories* and *Development graphs* of OMDoc[1].

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 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.

[1] These are the modules CTH and DG, respectively.

Part XLVI

CPoint: An OMDoc Editor in MS PowerPoint

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.

Chapter 101

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**.

Chapter 102

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 [**Kohlhase:OvercomingProprietaryHurdles**]) **C**Point makes these semantic authoring tools available through a toolbar in the PPT menu (see Figure 102.1) 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.

Figure 102.1: 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 102.2 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 102.2 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.

The information annotated in these processes can be exploited for added-value services.

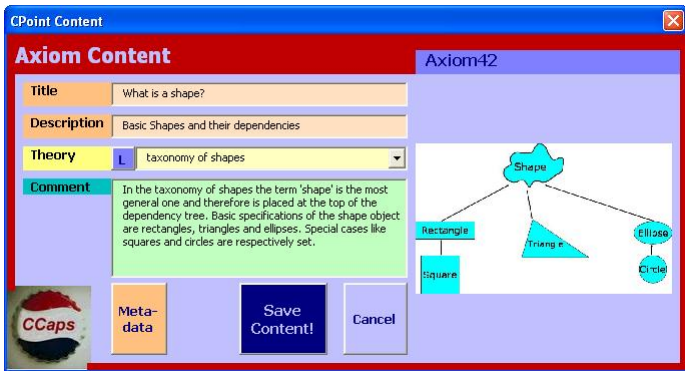


Figure 102.2: The CPoint Content Form for an Axiom Object

OMDoc Conversion The heart of CPoint is the functionality for converting a fully (CPoint-)edited presentation into a valid OMDoc document. This generated OMDoc document can for instance be read into computer-supported education systems like ActiveMath (see [Mel+01] and Part XL).

Added-Value Services As author support is essential for the motivation doing the semantic markup process, CPoint offers the following added-value services:

Content Search and Navigation CPoint'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 CPoint'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. CPoint 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 CPoint. 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.

Chapter 103

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 CPoint novices will be implemented. This CPointBasic module will try to overcome the heavily form-oriented format of CPoint. In a next step the growing of a CPoint user will be supported by offering advanced CPoint 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 CPoint are replaced by "social tags". Another CPoint project is its extension for usage by teachers in school, which usefulness has already been established in [Koh06a]. The newest project at the International University of Bremen is the implementation of a CPoint-like editor for MS Word.

Part XLVII

sTeX: A LaTeX-Based Workflow for OMDoc

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.

Chapter 104

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[1], 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.
- 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.

[1] supplying macros e.g. for sections, paragraphs, theorems, definitions, etc.

Chapter 105

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 [Mil]. 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[1], 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.

[=content-markup-LaTeX]

[1] i.e. all macros, environments, and syntax extensions used int the source document

Chapter 106

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"[1] for binomial coefficients: $\binom{n}{k}$, ${}_nC^k$, $\mathcal{C}_k^n$, and $\mathcal{C}_n^k$ 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        (106.1)
```

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 (106.1), 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.

[1] The first one is standard e.g. in Germany and the US, the third one in France, and the last one in Russia

Chapter 107

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 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 [**Kohlhase:smtl05**] 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 $\sum_{i=1}^{n} 2i-1$ 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).

```
\begin{module}[id=arith]
  \symdef{Sumfromto}[4]{\sum_{#1=#2}^{#3}{#4}}
  \symdef[local]{arbitraryn}{n}
  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 [**Kohlhase:smtl05**], 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 Part XL).

Part XLVIII

Standardizing Context in System Interoperability

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 Part XII. 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 [Ben+97], INKA [HS96], PVS [ORS92], *λClam* [RSG98], TPS [And+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 [KK06] 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 Chapter 107 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 Chapter 108.

Chapter 108

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 108.1, 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

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	

Figure 108.1: Set Theories in ΩMEGA and PVS

their independent growth over 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 108.2 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 108.1) 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.

Note that *technically* we do not need to change the interface language of the mathematical software systems[1], we only rationally reconstruct their meaning in terms of the new integration

[1]This is important if we want to integrate proprietary software systems, where we have no control over the interfaces.

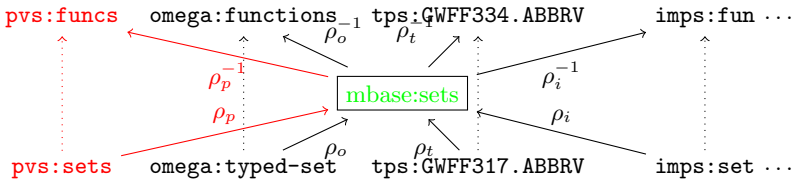


Figure 108.2: Theory Translations for System Integration

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.

Chapter 109

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 [Bru80]), 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.

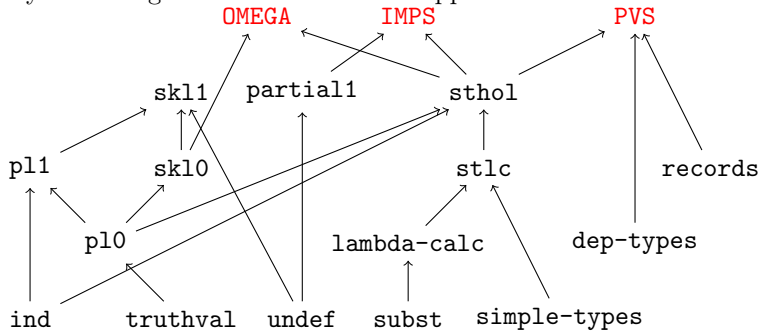


Figure 109.1: A Hierarchy of Logical Languages

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 109.1. 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.

Chapter 110

Logic Interoperability via Logic Morphisms

In theoretical accounts of the integration of logical languages, one finds categorical accounts like the one described in ?hets? 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 110.0.1 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 $\mathcal{D}: \mathcal{H} \vdash_{\mathcal{C}} \mathbf{A}$. Let $\mathcal{S}$ and $\mathcal{S}'$ be logical systems, then a **logic morphism** $\mathcal{F}: \mathcal{S} \to \mathcal{S}'$ consists of a **language morphism** $\mathcal{F}^{\mathcal{L}}: \mathcal{L} \to \mathcal{L}'$ and a **calculus morphism** $\mathcal{F}^{\mathcal{D}}$ from $\mathcal{C}$-derivations to $\mathcal{C}'$-derivations, such that for any $\mathcal{C}$-derivation $\mathcal{D}: \mathcal{H} \vdash_{\mathcal{C}} \mathbf{A}$ we have $\mathcal{F}^{\mathcal{D}}(\mathcal{D}): \mathcal{F}^{\mathcal{L}}(\mathcal{H}) \vdash_{\mathcal{C}'} \mathcal{F}^{\mathcal{L}}(\mathbf{A})$.

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. $\mathcal{R}([+: \mathbb{N} \to \mathbb{N} \to \mathbb{N}]) = \forall X, Y. \mathbb{N}(X) \wedge \mathbb{N}(Y) \Rightarrow \mathbb{N}(X + Y)$. 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 110.1, where $\mathbb{A}, \mathbb{B}, \ldots$ are sort symbols.

$$\mathcal{R}\left(\frac{\mathbf{A}: \mathbb{B} \to \mathbb{C} \quad \mathbf{B}: \mathbb{B}}{\mathbf{AB}: \mathbb{C}}\right) = \dfrac{\dfrac{\forall X. \mathbb{B}(X) \Rightarrow \mathbb{C}(\mathbf{A}X)}{\mathbb{B}(\mathbf{B}) \Rightarrow \mathbb{C}(\mathbf{AB})} \quad \mathbb{B}(\mathbf{B})}{\mathbb{C}(\mathbf{AB})}$$

$$\mathcal{R}\left(\frac{\forall X_{\mathbb{B}}.\mathbf{A} \quad \mathbf{B}: \mathbb{B}}{[\mathbf{B}/X]\mathbf{A}}\right) = \dfrac{\dfrac{\forall X. \mathbb{B}(X) \Rightarrow \mathcal{R}(\mathbf{A})}{\mathbb{B}(\mathcal{R}(\mathbf{B})) \Rightarrow \mathcal{R}([\mathbf{B}/X]\mathbf{A})} \quad \mathbb{B}(\mathbf{B})}{\mathcal{R}([\mathbf{B}/X]\mathbf{A})}$$

Figure 110.1: Relativization Morphism on Proofs

In Definition **??** 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 Part XXIV). 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 **??** now simply takes on the form of a type compatibility condition.

Part XLIX

Integrating Proof Assistants as Plugins in a Scientific Editor

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 [Hoe01] in order to provide direct access from it to the mathematics assistance system ΩMEGA [**OMEGA02**; SBA05]. Generally, we aim at an approach that is not dependent on the particular proof assistant system to be integrated [Aut+06].

TeXmacs [Hoe01] 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 [**OMEGA02**; 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 TEX$_{\text{MACS}}$'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 TEX$_{\text{MACS}}$. 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 TEX$_{\text{MACS}}$ 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 TEX$_{\text{MACS}}$ to ΩMEGA is to limit the details that must be provided by the user in the TEX$_{\text{MACS}}$ 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 representation format for this pure mathematical content implemented both in TEX$_{\text{MACS}}$ 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 TEX$_{\text{MACS}}$ by the language S. Thus, if we denote the document format of TEX$_{\text{MACS}}$ by T, we define a semantic document format $T+S$ as a document format still accepted by TEX$_{\text{MACS}}$.

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 TEX$_{\text{MACS}}$ 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 TEX$_{\text{MACS}}$.

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 [Aut+03; AF06], which we developed from previous experiences with tutorial dialogs about mathematical proofs between a computer and students [Pin+04].

Part L

OMDoc as a Data Format for √eriFun

√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 L), 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 47), 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.

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.

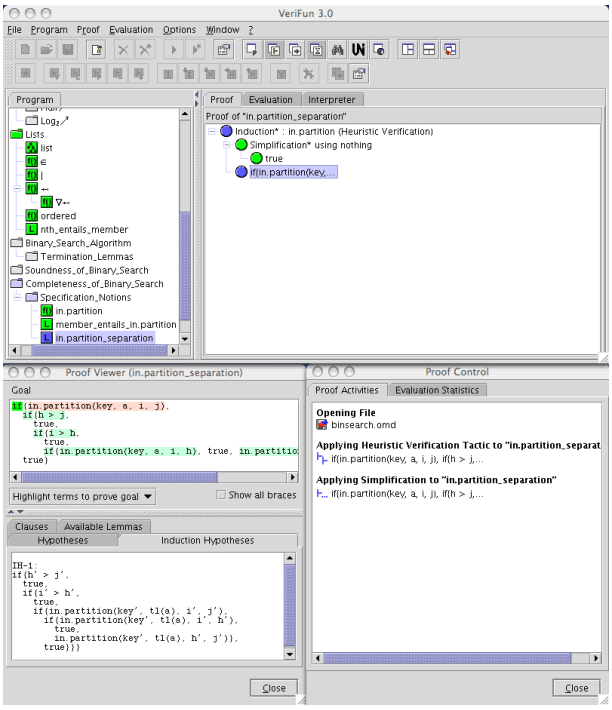


Figure 110.2: A √eriFun session

- 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.

Chapter 111

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[1] 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 (Section 27.2). 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 111.3). By using this technique we can use any character string[2] 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[3].

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.

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.

[1] Actually there are also automatically system-generated elements included, but we may neglect those at this point.

[2] ✓`eriFun` has full UNICODE [Inc03] support

[3] As a consequence the previous mentioned special encoding feature does not hold for outfix functions

Listing 111.1: A polymorphic √veriFun 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 √veriFun could be polymorphic (Listing 111.1). This led to the additional, optional `parameters` attribute of an `adt` element (Listing 111.2). Within this new attribute one can declare by a comma separated list the names of type variables of the abstract data type.

Listing 111.2: 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 111.3 illustrates the corresponding `presentation` elements of the ADT in Listing 111.2.

Listing 111.3: 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>
```

Chapter 112

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, √`veriFun` generated knowledge became accessible, robust, interchangeable and transparent.

Part LI

Appendix

In this appendix, we document the changes of the OMDoc format over the versions, provide quick reference tables, and discuss the validation helps

Part LII

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.

OMDoc1.1 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.

OMDoc1.2 is the mature version in the OMDoc1 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 [DCM03a]). 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 1.2 [Koh06c]. For older changes see Appendix A there.

Appendix A

Changes from OMDoc1.2 to OMDoc1.6

OMDoc1.6 is the first step towards a second version of the OMDoc format, the changes we see here are more disruptive, aimed at regularizing the concepts underlying the language. Old functionality will largely be kept for backwards compatibility.

One of the larger technical changes is that the OMDoc namespace changed from `http://www.mathweb.org/omdoc` to `http://omdoc.org/ns` for the OMDoc2 format (see Chapter 17).

element	state	comments	cf.
`definition`	cha	The `type` may no longer have the value `informal`, definitions are "informal", iff they do not have formal parts.	Definition ????
`om:*`	cha	The `cref` attributes that were introduced in OMDoc1.2 for parallel markup with cross-references are no longer needed.	Definition ????
`p`	cha	The `p` has been shifted to module DOC	?eldef.p?
`omd`	new	The `metadata` element can now contain an element `omd` for a generic metadatum.	?eldef.omd?
`cc:license`	ext	The `cc:license` license can now contain a multilingual `CMP` group can be used to give a natural language explanation of the license grant.	Definition ????
`cc:permissions`	ext	The `cc:permissions` license can now contain a multilingual `CMP` group can be used to give a natural language explanation of the license grant.	Definition ????
`cc:prohibitions`	ext	The `cc:prohibitions` license can now contain a multilingual `CMP` group can be used to give a natural language explanation of the license grant.	Definition ????
`cc:requirements`	ext	The `cc:requirements` license can now contain a multilingual `CMP` group can be used to give a natural language explanation of the license grant.	Definition ????
`phrase`	cha	`type` attribute no longer supports the values of the `omtext` `type` attribute but accepts the values nucleus and satellite. `for` and `relation` attributes are introduced as additional support for the satellite value of the `type` attribute. The values for the `relation` attribute are editted variant of the values for the `type` attribute used in OMDoc1.2.	?eldef.phrase?

Appendix B

Planned Changes for OMDoc2.0

Part LIII

Quick-Reference Table to the OMDoc Elements

Element	p.	Mod.	Required Attribs	Optional Attribs	M D	Content
adt	Definition ????	ADT		xml:id, type, style, class, theory, generated-from, generated-via	+	sortdef+
alternative	?eldef.alternative?	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	Definition ????	QUIZ	verdict	xml:id, style, class	+	CMP*, FMP*
m:apply	Definition ????	MML		id, xlink:href	–	bvar?, ⟨⟨*CMel*⟩⟩*
argument	Definition ????	ADT	sort		+	selector?
assertion	Definition ????	ST		xml:id, type, theory, generated-from, generated-via, style, class	+	CMP*, FMP*
assumption	?eldef.assumption?	MTXT		xml:id, inductive, style, class	+	CMP*, (OMOBJ |m:math |legacy)?
attribute	?eldef.attribute?	PRES	name		–	(value-of| text)*
axiom	Definition ????	ST	name	xml:id, type, generated-from, generated-via, style, class	+	CMP*, FMP*
axiom-inclusion	Definition ????	CTH	from, to	xml:id, style, class, theory, generated-from, generated-via	+	morphism?, (path-just| obligation*)

m:bvar	Definition ????	MML		id, xlink:href	–	ci*
m:ci	Definition ????	MML		id, xlink:href	–	PCDATA
m:cn	Definition ????	MML		id, xlink:href	–	([0-9]|,|.) (*|e([0-9]|,|.)*)?
choice	Definition ????	QUIZ		xml:id, style, class	+	CMP*, FMP*
CMP	?eldef.CMP?	MTXT		xml:lang, xml:id	–	(text| OMOBJ |m:math |legacy | with | term | omlet)*
code	Definition ????	EXT		xml:id, for, theory, generated-from, generated-via, requires, style, class	+	input?, output?, effect?, data+
conclusion	?eldef.conclusion?	MTXT		xml:id, style, class	+	CMP*, (OMOBJ |m:math |legacy)?
constructor	Definition ????	ADT	name	type, scope, style, class, theory, generated-from, generated-via	+	argument*, recognizer?
dc:contributor	Definition ????	DC		xml:id, role, style, class	–	⟪*text*⟫
dc:creator	Definition ????	DC		xml:id, role, style, class	–	⟪*text*⟫
m:csymbol	Definition ????	MML	definitionURL	id, xlink:href	–	EMPTY
data	Definition ????	EXT		format, href, size, original	–	<![CDATA[...]]>
dc:date	Definition ????	DC		action, who	–	ISO 8601 norm
dd	?eldef.dd?	RT		xml:id, style, class, index, verbalizes	+	CMPcontent
di	?eldef.di?	RT		xml:id, style, class, index, verbalizes	+	dt+,dd*
dl	?eldef.dl?	RT		xml:id, style, class, index, verbalizes	+	li*
dt	?eldef.dt?	RT		xml:id, style, class, index, verbalizes	+	CMPcontent
decomposition	Definition ????	DG	links	theory, generated-from, generated-via	–	EMPTY
definition	Definition ????	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	?eldef.dc:description?	DC		xml:lang	–	CMPcontent
derive	Definition ????	PF		xml:id, style, class	–	CMP*, FMP?, method?
effect	Definition ????	EXT		xml:id, style, class	–	CMP*,FMP*
element	?eldef.element?	PRES	name	xml:id, cr, ns	–	(attribute| element| text| recurse)*

example	Definition ????	ST	for	xml:id, type, assertion, proof, style, class, theory, generated-from, generated-via	+	CMP*| (OMOBJ |m:math |legacy)?
exercise	Definition ????	QUIZ		xml:id, type, for, from, style, class, theory, generated-from, generated-via	+	CMP*, FMP*, hint?, (solution*|mc*)
FMP	?eldef.FMP?	MTXT		logic, xml:id	–	(assumption*, conclusion*)|OMOBJ |m:math |legacy
dc:format	Definition ????	DC			–	fixed: "application/omdoc+xml"
hint	Definition ????	QUIZ		xml:id, style, class, theory, generated-from, generated-via	+	CMP*, FMP*
hypothesis	Definition ????	PF		xml:id, style, class, inductive	–	CMP*, FMP*
dc:identifier	Definition ????	DC		scheme	–	ANY
ide	Definition ????	RT	index	xml:id,sort-by,see, seealso, links, style, class	–	idp*
idp	Definition ????	RT		xml:id,sort-by,see, seealso, links, style, class	–	CMPcontent
idt	Definition ????	RT		style, class	–	CMPcontent
idx	Definition ????	RT		xml:id,sort-by,see, seealso, links, style, class	–	idt?,idp+
ignore	Definition ????	DOC		type, comment	–	ANY
imports	Definition ????	CTH	from	xml:id, type, style, class	+	morphism?
inclusion	Definition ????	CTH	for	xml:id	–	
input	Definition ????	EXT		xml:id, style, class	–	CMP*,FMP*
insort	Definition ????	ADT	for		–	
dc:language	Definition ????	DC			–	ISO 8601 norm
li	?eldef.li?	RT		xml:id, style, class, index, verbalizes	–	Math Vernacular
cc:license	Definition ????	DC		jurisdiction	–	permissions, prohibitions, requirements
link	?eldef.link?	RT		xml:id, style, class, index, verbalizes	–	Math Vernacular
m:math	Definition ????	MML		id, xlink:href	–	$\langle\!\langle CMel \rangle\!\rangle$+
mc	Definition ????	QUIZ		xml:id, style, class, theory, generated-from, generated-via	–	choice, hint?, answer
measure	Definition ????	ST		xml:id	–	OMOBJ |m:math |legacy
metadata	Definition ????	DC		inherits	–	(dc-element)*
method	Definition ????	PF	xref		–	(OMOBJ |m:math |legacy| premise | proof | proofobject)*

morphism	Definition ????	CTH		xml:id, base, consistency, exhaustivity, type, hiding, style, class	–	requation*, measure?, ordering?
note	Definition ????	RT		type,xml:id, style, class, index, verbalizes	–	Math Vernacular
obligation	Definition ????	CTH	induced-by, assertion	xml:id	–	EMPTY
om:OMA	Definition ????	OM		id, cdbase	–	⟪*OMel*⟫*
om:OMATTR	Definition ????	OM		id, cdbase	–	⟪*OMel*⟫
om:OMATP	Definition ????	OM		cdbase	–	(OMS, (⟪*OMel*⟫ | om:OMFOREIGN))+
om:OMB	Definition ????	OM		id, class, style, class	–	#PCDATA
om:OMBIND	Definition ????	OM		id, cdbase	–	⟪*OMel*⟫, om:OMBVAR, ⟪*OMel*⟫?
om:OMBVAR	Definition ????	OM			–	(om:OMV | om:OMATTR)+
om:OMFOREIGN	Definition ????	OM		id, cdbase	–	ANY
omdoc	Definition ????	DOC		xml:id,type, version, style, class, xmlns, theory, generated-from, generated-via	+	(top-level element)*
om:OME	Definition ????	OM		xml:id	–	(⟪*OMel*⟫)?
om:OMR	Definition ????	OM	href		–	
om:OMF	Definition ????	OM		id, dec, hex	–	#PCDATA
ol	?eldef.ol?	RT		xml:id, style, class, index, verbalizes	–	li*
om:OMI	Definition ????	OM		id, class, style	–	[0-9]*
omlet	Definition ????	EXT		id, argstr, type, function, action, data, style, class	+	ANY
omstyle	?eldef.omstyle?	PRES	element	for, xml:id, xref, style, class	–	(style|xslt)*
om:OMS	Definition ????	OM	cd, name	class, style	–	EMPTY
omtext	?eldef.omtext?	MTXT		xml:id, type, for, from, style, theory, generated-from, generated-via	+	CMP+, FMP?
om:OMV	Definition ????	OM	name	class, style	–	EMPTY
ordering	Definition ????	ST		xml:id	–	OMOBJ |m:math |legacy
output	Definition ????	EXT		xml:id, style, class	–	CMP*,FMP*
p	?eldef.p?	RT		xml:id, style, class, index, verbalizes	–	Math Vernacular
param	Definition ????	EXT	name	value, valuetype	–	EMPTY
path-just	Definition ????	DG	local, globals	for, xml:id	–	EMPTY
cc:permissions	Definition ????	DC		reproduction, distribution, derivative_works	–	EMPTY
premise	Definition ????	PF	xref		–	EMPTY

presentation	?eldef.presentation?	PRES	for	xml:id, xref, fixity, role, lbrack, rbrack, separator, bracket-style, style, class, precedence, crossref-symbol	–	(use | xslt | style)*
private	Definition ????	EXT		xml:id, for, theory, generated-from, generated-via, requires, reformulates, style, class	+	data+
cc:prohibitions	Definition ????	DC		commercial_use	–	EMPTY
proof	Definition ????	PF		xml:id, for,theory, generated-from, generated-via, style, class	+	(symbol | definition | omtext | derive | hypothesis)*
proofobject	Definition ????	PF		xml:id, for, theory, generated-from, generated-via, style, class	+	CMP*, (OMOBJ |m:math |legacy)
dc:publisher	Definition ????	DC		xml:id, style, class	–	ANY
ref	Definition ????	DOC		xref, type	–	ANY
recognizer	Definition ????	ADT	name	type, scope, role, style, class	+	
recurse	?eldef.recurse?	PRES		select	–	EMPTY
dc:relation	Definition ????	DC			–	ANY
requation	Definition ????	ST		xml:id, style, class	–	(OMOBJ |m:math |legacy),(OMOBJ |m:math |legacy)
cc:requirements	Definition ????	DC		notice, copyleft, attribution	–	EMPTY
dc:rights	Definition ????	DC			–	ANY
selector	Definition ????	ADT	name	type, scope, role, total, style, class	+	
solution	Definition ????	QUIZ		xml:id, for, style, class, theory, generated-from, generated-via	+	(CMP*, FMP*) | proof
sortdef	Definition ????	ADT	name	role, scope, style, class	+	(constructor|insort)*
dc:source	Definition ????	DC			–	ANY
style	?eldef.style?	PRES	format	xml:lang, requires	–	(element | text | recurse | value-of)*
dc:subject	Definition ????	DC		xml:lang	–	CMPcontent
symbol	Definition ????	ST	name	role, scope, style, class,generated-from,generated-via	+	type*
table	?eldef.table?	RT		xml:id, style, class, index, verbalizes	–	tr*
term	Definition ????	MTXT	cd, name	xml:id, role, style, class	–	CMP content
text	?eldef.text?	PRES			–	#PCDATA
td	?eldef.td?	RT		xml:id, style, class, index, verbalizes	–	Math Vernacular
th	?eldef.th?	RT		xml:id, style, class, index, verbalizes	–	Math Vernacular
theory	Definition ????	ST	xml:id	cdbase, style, class	+	(statement|theory)*

theory-inclusion	Definition ????	CTH	from, to	xml:id, style, class, theory, generated-from, generated-via	+	(morphism, decomposition?)
tr	?eldef.tr?	RT		xml:id, style, class, index, verbalizes	–	(td|th)*
dc:title	Definition ????	DC		xml:lang	–	CMPcontent
tgroup	Definition ????	DOC		xml:id, type, style, class, modules, generated-from, generated-via	+	top-level or theory-constitutive element*
type	Definition ????	ST	system	xml:id, for, style, class	–	CMP*, (OMOBJ |m:math |legacy)
dc:type	Definition ????	DC			–	fixed: "Dataset" or "Text" or "Collection"
ul	?eldef.ul?	RT		xml:id, style, class, index, verbalizes	–	li*
use	?eldef.use?	PRES	format	xml:lang, requires, fixity, lbrack, rbrack, separator, crossref-symbol, element, attributes	–	(use | xslt | style)*
value-of	?eldef.value-of?	PRES	select		–	EMPTY
phrase	?eldef.phrase?	MTXT		xml:id, style, class, index, verbalizes, type	–	CMP content
xslt	?eldef.xslt?	PRES	format	xml:lang, requires	–	XSLT fragment

Part LIV

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.	
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, omdoc	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	omdoc, 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.	

Part LV

The RelaxNG Schema for OMDoc

We reprint the modularized RelaxNG schema for OMDoc here. It is available at `http://www.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 typedefinitions and even XML schemata can be generated from it by standard tools.

The RelaxNG schema consists of the grammar fragments for the modules (see Chapter C to Chapter O).

Appendix C

The Sub-Language Drivers

The Schema comes in two parts: strict OMDoc

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.6)
# $Id: omdoc-strict.rnc 8550 2009-11-07 06:38:23Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/trunk/schema/rnc/strict/omdoc-strict.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://omdoc.org/ns"
namespace local = ""

start = omdoc

# whenever we want to leave the models open
Anything = (AnyElement | text)*
AnyElement = element * {(attribute * {text} | AnyElement | text)*}

# all the explicitly namespaced attributes, except xml:lang, which handled explicitly
nonlocal.attribs = attribute * - (local:* | xml:*) {text}*

id.attribs = attribute xml:id {xsd:ID}? & nonlocal.attribs
idrest.attribs = empty

## MMT URIs
MMTURI = MURI | DURI | SURI
## Document URI: DURI ::= URI without Fragment
DURI = xsd:anyURI
## Module URI: MURI ::= DURI?QName | ?/QName
MURI = xsd:anyURI
## Symbol URI: SURI ::= MURI?QName | ??/QName
SURI = xsd:anyURI

name.attrib = attribute name {xsd:string}
from.attrib = attribute from {MMTURI}
to.attrib = attribute to {MMTURI}

include "omdocmobj.rnc"
include "meta-strict.rnc"
include "doc-strict.rnc"
include "mtxt-strict.rnc"
include "st-strict.rnc"
include "biform.rnc"
include "notation-strict.rnc"
```

and pragmatic OMDoc

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.6)
# $Id: omdoc.rnc 8553 2009-11-07 15:42:34Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/trunk/schema/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 "strict/omdoc-strict.rnc"

# the attributes for CSS and PRES styling
css.attribs = attribute style {xsd:string}?&attribute class {xsd:string}?
xref.attrib = attribute xref {MMTURI}
```

```
idrest.attribs &= css.attribs&attribute xml:base {MMTURI}?
id.attrib &= idrest.attribs
fori.attrib = attribute for {MMTURI}?

tref = attribute tref {MMTURI}

# ********* think about his again.
omdoc.toplevel.attribs = id.attribs, attribute generated-from {MMTURI}?

include "pragmatic/omdocdc.rnc"
include "pragmatic/omdoccc.rnc"
include "pragmatic/omdocdoc.rnc"
include "pragmatic/omdocmtxt.rnc"

include "pragmatic/notation-mmt.rnc"
include "pragmatic/omdocst.rnc"
include "pragmatic/omdocpf.rnc"
include "pragmatic/omdocadt.rnc"
include "pragmatic/omdocext.rnc"
include "pragmatic/omdocquiz.rnc"
```

Appendix D

Module MOBJ: Mathematical Objects and Text

The RNC module MOBJ includes the representations for mathematical objects and defines the `legacy` element (see Part XV for a discussion). It includes the standard RelaxNG schema for OpenMath (we have reprinted it in Appendix **??**) adding the OMDoc identifier and CSS attributes to all elements. If also includes a schema for MathML (see Appendix **??**).

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.6) Module MOBJ
# $Id: omdocmobj.rnc 8937 2011-08-08 07:26:52Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/trunk/schema/rnc/strict/omdocmobj.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2009 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://omdoc.org/ns"
namespace om = "http://www.openmath.org/OpenMath"

# the legacy element, it can encapsulate the non-migrated formats
legacy.attribs = id.attribs & attribute formalism {xsd:anyURI}?
legacy.textformat = "TeX" | "LaTeX" | "ASCII"
legacy.text = legacy.attribs & attribute format {legacy.textformat} & text
legacy.any = legacy.attribs & attribute format {xsd:anyURI} & Anything
legacy.model = legacy.text | legacy.any

legacy = element legacy {tref|legacy.model}

OMel = grammar {include "../../openmath/openmath3.rnc"
                {start = omel}
                common.attributes &= parent idrest.attribs & parent nonlocal.attribs}

OMS = grammar {include "openmath/openmath3.rnc"
               {start = OMS}
               common.attributes &= parent idrest.attribs & parent nonlocal.attribs}

cmml = grammar {include "../../mathml3/mathml3-common.rnc"
                include "../../ mathml3/mathml3-strict-content.rnc"}

# ****** do something about the cdbase of mmt.
mmtcd = attribute cdbase {""}? & attribute cd {"mmt"}
identity = element om:OMS {mmtcd & attribute name {"identity"}}
composition = element om:OMS {mmtcd & attribute name {"composition"}}
morphismapplication = element om:OMS {mmtcd & attribute name {"morphismapplication"}}

morphism = OMS
         | element om:OMA {identity, theo}
         | element om:OMA {composition, morphism*}

theo = OMS

mobj = legacy | OMel | cmml
```

Appendix E

Module MTXT: Mathematical Text

The RNC module MTXT provides infrastructure for mathematical vernacular (see Part XIX for a discussion).

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.6) Module MTXT
# $Id: mtxt-strict.rnc 8768 2010-12-06 12:57:01Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/trunk/schema/rnc/strict/mtxt-strict.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://omdoc.org/ns"

omdoc.class &= omtext*

rst.attribs = attribute verbalizes {MMTURI}?

omtext.attribs= id.attribs & rst.attribs
omtext.model = metadata.class & p*
omtext = element omtext {tref|(omtext.attribs & omtext.model)}

triple.att = attribute cdbase {xsd:anyURI}? &
             attribute name {xsd:NCName}? &
             attribute cd {xsd:NCName}?

term.attribs = id.attribs & triple.att
term.model = p.model
\term = element term {tref|(term.attribs & term.model)}

p = grammar {include "pxhtml.rnc"
                {Inline.model = (text | (parent metadata.class) | Inline.class)*}
                Inline.class |= parent op.class
                span.attlist &= parent rst.attribs
                start = p}
p.class = grammar {include "pxhtml.rnc"
                      Inline.class |= parent op.class
                      start = Inline.class}

p.model = (text | p.class)*

op.class = \term | mobj | note | idx | citation
p.class |= op.class

note = element note {tref|(id.attribs,attribute type {xsd:NMTOKEN}?,(p* | p.model))}

index.att = attribute index {xsd:NCName}?

idep.attribs = attribute sort-by {text}? &
               attribute see {xsd:anyURI}? &
               attribute seealso {xsd:anyURI}? &
               attribute links {list {xsd:anyURI*}}?

idx.attribs = id.attribs|xref.attrib
idx.model = idt?,ide+
idx = element idx {idx.attribs & idx.model}
```

```
ide.attribs = index.att & idep.attribs
ide.model = idp*
ide = element ide {ide.attribs,ide.model}

idt.attribs = idrest.attribs
idt.model = p.model
idt = element idt {idt.attribs & idt.model}

idp.attribs = index.att
idp.model = p.model
idp = element idp {idp.attribs & idp.model}

citation.attribs = attribute href {xsd:anyURI} | attribute bibref {text}
citation.model = empty
citation = element citation {citation.attribs & citation.model}
```

And now the pragmatic language

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.6) Module MTXT
# $Id: omdocmtxt.rnc 8633 2010-04-12 06:45:10Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/trunk/schema/rnc/pragmatic/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://omdoc.org/ns"

# textblocktype commented and left until further notice, rsttype recovered in
# the grammar with its initial purpose in the omtext element

#textblocktype = "introduction" | "background" | "motivation" | "scenario" |
#                "contribution" | "evaluation" | "results" | "discussion" |
#                "conclusion"

rsttype = "abstract" | "introduction" | "annote" |
          "conclusion" | "thesis" | "comment" | "antithesis" |
          "elaboration" | "motivation" | "evidence" | "note" |
          "warning" | "question" | "answer" | "transition"

# relationtype - used in phrase element introduced instead
relationtype = "antithesis" | "circumstance" | "concession" | "condition" |
               "evidence" | "means" | "preparation" | "purpose" | "cause" |
               "consequence" | "elaboration" | "restatement" | "solutionhood"

statementtype = "axiom" | "definition" | "example" | "proof" |
                "derive" | "hypothesis" | "notation"

assertiontype = "assertion" | "theorem" | "lemma" | "corollary" |
                "proposition" | "conjecture" | "false-conjecture" |
                "obligation" | "postulate" | "formula" | "assumption" |
                "rule"

# omtext can take as argument rsttype and to extend extensibility if a type is not
# specified, an typeURI is offered as additional option


omtext.attribs &= ( attribute type {(rsttype | statementtype | assertiontype)}
                  | attribute typeURI {xsd:anyURI})? &
                  attribute for {MMTURI}?&
                  attribute from {MMTURI}?

# in phrase element the type attribute is changed to take values from nucleus and
# satellite, also the relation attribute is introduced having values of type relationtype,
# and the for attribute is introduce to connect the satellite with the corresponding
# nucleus.

phrase.attribs &= attribute type {"nucleus"} |
                  (attribute type {"satellite"} &
                   attribute relation {relationtype} &
                   attribute for {MMTURI})
```

Appendix F

Module DOC: Metadata

For the treatment of metadata we include a generic version of the Dublin Core vocabulary for bibliographic metadata (see the schema at Chapter F), and extend it with MARC relator roles (see Part XXI and Section 46.0 for a discussion and Chapter G for the schema) and a content-oriented version of Creative Commons License specifications (see Chapter H for the schema).

```
# A RelaxNG for Open Mathematical documents (OMDoc 1.6) Module META
# $Id: meta-strict.rnc 8540 2009-10-20 10:14:01Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/trunk/schema/rnc/meta-strict.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2007-2008 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://omdoc.org/ns"

# for the moment, we may get regexp at some point.
curie = xsd:string
curies = xsd:string
safecurie = xsd:string

rel.attrib = attribute rel {curies}
rev.attrib = attribute rev {curies}
content.attrib = attribute content {xsd:string}
about.attrib = attribute about {xsd:anyURI|safecurie}
resource.attrib = attribute resource {xsd:anyURI|safecurie}
property.attrib = attribute property {curies}
datatype.attrib = attribute datatype {curie}
typeof.attrib = attribute typeof {curies}

meta.attribs = id.attribs & property.attrib?& datatype.attrib?
meta.model = content.attrib | Anything | (content.attrib & Anything)
meta = element meta {tref|(meta.attribs & meta.model)}

mlink.attribs = id.attribs & rel.attrib? & rev.attrib? & resource.attrib?
mlink.class = resource* & mlink* & meta*
mlink.model = attribute href {curie}|mlink.class
mlink = element link {tref|(mlink.attribs,mlink.model)}

resource.attribs = id.attribs & typeof.attrib? & about.attrib?
resource.class = meta* & mlink*
resource = element resource {tref|(resource.attribs & resource.class)}

metadata.class = meta* & mlink*
rdfa.attribs = rel.attrib? & rev.attrib? & content.attrib? & about.attrib?
                & resource.attrib? & property.attrib? & datatype.attrib?
                & typeof.attrib?

id.attribs &= rdfa.attribs
```

Appendix G

Dublin Core Metadata

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.6) Module DC
# $Id: omdocdc.rnc 8812 2011-01-13 15:07:34Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/trunk/schema/rnc/pragmatic/omdocdc.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2010 Michael Kohlhase, released under the GNU Public License (GPL)

# we include the dublin core and MARC elements, filling them with our content types
dublincore = grammar {include "MARCRelators.rnc"
      include "dublincore.rnc"
           {dc.date = parent nonlocal.attribs &
                        attribute action {xsd:NMTOKEN}? &
                        attribute who {xsd:anyURI}? &
                        (xsd:date|xsd:dateTime)
            dc.identifier = parent tref|(parent nonlocal.attribs & attribute scheme {xsd:NMTOKEN} & text)
            dc.type = parent tref|(parent nonlocal.attribs & ("Dataset" | "Text" | "Collection"))
            dc.text = parent tref|(parent nonlocal.attribs & parent p.model)
            dc.person = parent tref|(parent nonlocal.attribs & attribute role {MARCRelators}? & parent p.model)
           dc.rights = parent tref|(parent nonlocal.attribs & parent p.model)}}

metadata.class &= dublincore
```

```
# A RelaxNG schema for the Dublin Core elements
# $Id: dublincore.rnc 8550 2009-11-07 06:38:23Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/trunk/schema/rnc/pragmatic/dublincore.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2008 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace dc = "http://purl.org/dc/elements/1.1/"

## the various content models, specialize for integration
dc.person = text
dc.publisher = text
dc.text = text
dc.format = text
dc.source = text
dc.language = text
dc.rights = text
dc.relation = text
dc.date = xsd:dateTime
dc.type = text
dc.identifier = text

# the model of the Dublin Metadata initiative (http://purl.org/dc)
start = contributor* & creator* & rights* & subject* & title* & description* &
          publisher* & date* & type* & format* & identifier* & source* & language* & relation*

contributor = element contributor {dc.person}
creator = element creator {dc.person}
title = element title {dc.text}
subject = element subject {dc.text}
description = element description {dc.text}
publisher = element publisher {dc.publisher}
type = element type {dc.type}
format = element format {dc.format}
source = element source {dc.source}
language = element language {dc.language}
relation = element relation {dc.relation}
```

```
rights  = element rights {dc.rights}
date = element date {dc.date}
 identifier  = element  identifier  {dc. identifier }
```

Appendix H

MARC Relators for Bibliographic Roles

```
# the MARC relator set; see http://www.loc.gov/marc/relators
MARCRelators =
        "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"
```

Appendix I

Creative Commons Licenses

```
# A RelaxNG for Open Mathematical documents (OMDoc 1.6) Module CC
# $Id: omdoccc.rnc 8740 2010-09-27 09:16:55Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/trunk/schema/rnc/pragmatic/omdoccc.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2010 Michael Kohlhase, released under the GNU Public License (GPL)

# we include the OMDoc version of cc metadata and specialize the description
license = grammar {include "creativecommons.rnc" {description = parent p*}}

metadata.class &= license*
```

```
# A RelaxNG for Creative Commons License Specifications
# $Id: creativecommons.rnc 8550 2009-11-07 06:38:23Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/trunk/schema/rnc/pragmatic/creativecommons.rnc $
# Copyright (c) 2008 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace cc = "http://creativecommons.org/ns"

iana.tld = ("ac"|"ad"|"ae"|"af"|"ag"|"ai"|"al"|"am"|"an"|"ao"|"aq"|"ar"|"as"|"at"|"au"|"aw"|"ax"|"az"|
            "ba"|"bb"|"bd"|"be"|"bf"|"bg"|"bh"|"bi"|"bj"|"bm"|"bn"|"bo"|"br"|"bs"|"bt"|"bv"|"bw"|"by"|"bz"|
            "ca"|"cc"|"cd"|"cf"|"cg"|"ch"|"ci"|"ck"|"cl"|"cm"|"cn"|"co"|"cr"|"cs"|"cu"|"cv"|"cx"|"cy"|"cz"|
            "de"|"dj"|"dk"|"dm"|"do"|"dz"|"ec"|"ee"|"eg"|"eh"|"er"|"es"|"et"|"fi"|"fj"|"fk"|"fm"|"fo"|"fr"|
            "ga"|"gb"|"gd"|"ge"|"gf"|"gg"|"gh"|"gi"|"gl"|"gm"|"gn"|"gp"|"gq"|"gr"|"gs"|"gt"|"gu"|"gw"|"gy"|
            "hk"|"hm"|"hn"|"hr"|"ht"|"hu"|"id"|"ie"|"il"|"im"|"in"|"io"|"iq"|"ir"|"is"|"it"|"je"|"jm"|"jo"|"jp"|
            "ke"|"kg"|"kh"|"ki"|"km"|"kn"|"kp"|"kr"|"kw"|"ky"|"kz"|"la"|"lb"|
            "lc"|"li"|"lk"|"lr"|"ls"|"lt"|"lu"|"lv"|"ly"|
            "ma"|"mc"|"md"|"mg"|"mh"|"mk"|"ml"|"mm"|"mn"|"mo"|"mp"|"mq"|"mr"|"ms"|"mt"|"mu"|"mv"|"mw"|"mx"|"my"|"mz"|
            "na"|"nc"|"ne"|"nf"|"ng"|"ni"|"nl"|"no"|"np"|"nr"|"nu"|"nz"|"om"|
            "pa"|"pe"|"pf"|"pg"|"ph"|"pk"|"pl"|"pm"|"pn"|"pr"|"ps"|"pt"|"pw"|"py"|"qa"|"re"|"ro"|"ru"|"rw"|
            "sa"|"sb"|"sc"|"sd"|"se"|"sg"|"sh"|"si"|"sj"|"sk"|"sl"|"sm"|"sn"|"so"|"sr"|"st"|"sv"|"sy"|"sz"|
            "tc"|"td"|"tf"|"tg"|"th"|"tj"|"tk"|"tl"|"tm"|"tn"|"to"|"tp"|"tr"|"tt"|"tv"|"tw"|"tz"|"ua"|
            "ug"|"uk"|"um"|"us"|"uy"|"uz"|"va"|"vc"|"ve"|"vg"|"vi"|"vn"|"vu"|"wf"|"ws"|"ye"|"yt"|"yu"|"za"|"zm"|"zw")

license.attribs = attribute jurisdiction {iana.tld}? &
                  attribute version {xsd:string}?
license.model = permissions,prohibitions,requirements,description
license = element license {license.attribs & license.model}

permissions.attribs = attribute reproduction {"permitted" | "prohibited"} &
                      attribute distribution {"permitted" | "prohibited"} &
                      attribute derivative_works {"permitted" | "prohibited"}
permissions.model = description
permissions = element permissions {permissions.attribs & permissions.model}

prohibitions.attribs = attribute commercial_use {"prohibited" | "permitted"}
prohibitions.model = description
prohibitions = element prohibitions {prohibitions.attribs & prohibitions.model}

requirements.attribs = attribute notice {"required" | "not_required"} &
                       attribute attribution {"required" | "not_required"} &
                       attribute copyleft {"required" | "not_required"}
requirements.model = description
requirements = element requirements {requirements.attribs & requirements.model}

description.attribs = empty
description.model = text
description = element description {description.attribs & description.model}
```

start = license

Appendix J

Module DOC: Document Infrastructure

The RNC module DOC specifies the document infrastructure of OMDoc documents (see Part XX for a discussion).

```
# A RelaxNG for Open Mathematical documents (OMDoc 1.6) Module DOC
# $Id: doc-strict.rnc 8867 2011-01-19 10:37:44Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/trunk/schema/rnc/strict/doc-strict.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://omdoc.org/ns"

omdoc.attribs = id.attribs &
                attribute uri {MMTURI}? &
                attribute version {xsd:string {pattern = "1.6"}}? &
                attribute base {MMTURI}?
omdoc.model = metadata.class & omdoc.class
omdoc.class = empty
omdoc = element omdoc {tref|(omdoc.attribs&omdoc.model)}
```

```
# A RelaxNG for Open Mathematical documents (OMDoc 1.6) Module DOC
# $Id: omdocdoc.rnc 8740 2010-09-27 09:16:55Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/trunk/schema/rnc/pragmatic/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://omdoc.org/ns"
# extend the stuff that can go into a mathematical text

omdoc.class &= ignore* & omgroup* & tableofcontents*

ignore.attribs = id.attribs &
                 attribute type {xsd:string}? &
                 attribute comment {xsd:string}?
ignore.model = Anything
ignore = element ignore {tref|(ignore.attribs & ignore.model)}

ogroup.attribs = id.attribs,
                 attribute type {xsd:anyURI}?,
                 attribute modules {xsd:anyURI}?,
                 attribute layout {text}?

group.attribs =toplevel.attribs  & ogroup.attribs
group.model = metadata.class & omdoc.class
omgroup = element omgroup {tref|(group.attribs & group.model)}

# rhetoricalblocktype introduced having the same values as the textblocktype
# plus abstract and entities, to be used in the type attribute of the
# omdoc element, for now commented - until further notice!

# rhetoricalblocktype = textblocktype | "abstract" | "entities"

# ogroup.attribs = (attribute type { rhetoricalblocktype } | attribute typeURI {xsd:anyURI})?,
#                  attribute modules {xsd:anyURI}?,
```

```
#                  attribute layout {"sequence" | "itemize" | "enumeration" | "sectioning"}?

tableofcontents.attribs = attribute level {xsd:nonNegativeInteger}?
tableofcontents.model = empty
tableofcontents = element tableofcontents {tableofcontents.attribs & tableofcontents.model}

index.attribs = id.attribs
index.model = empty
index = element index{index.attribs & index.model}

bibliography.attribs = id.attribs, attribute files {text}
bibliography.model = empty
bibliography = element bibliography {bibliography.attribs & bibliography.model}

# we extend the omdoc element by the group attributes
omdoc.attribs &=ogroup.attribs
```

Appendix K

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 Part XVIII for a discussion.

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.6) Module ST

# $Id: st-strict.rnc 8554 2009-11-07 15:52:49Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/trunk/schema/rnc/strict/st-strict.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://omdoc.org/ns"

theory.attribs = name.attrib & attribute meta {MMTURI}?
theory.class = theory* & \include* & constitutive & structure* & omtext*
theory.model = metadata.class & theory.class
theory = element theory {tref|(theory.attribs & theory.model)}

type.attribs = empty
type.model = metadata.class & mobj
type = element type {tref|(type.attribs & type.model)}

supertype.attribs = empty
supertype.model = metadata.class & mobj
supertype = element supertype {tref|(supertype.attribs & supertype.model)}

sdef.attribs = empty
sdef.model = metadata.class & mobj
sdef = element definition {tref|(sdef.attribs&sdef.model)}

arguments = xsd:integer | "*"
constant.attribs = name.attrib & attribute arguments {arguments}?
constant.class = type? & supertype? & sdef?
constant.model = metadata.class & constant.class
constitutive = element constant {constant.attribs & attribute role {consrole}? & constant.model}*
nonconstit = element constant {constant.attribs & attribute role {noncrole}? & constant.model}*

noncrole= "theorem" | "proof"
synrole = "binder" | "semantic-attribution" | "attribution" | "key"
consrole = "element" | "sort" | "axiom" | "judgment" | "error" | "errortype" | "level" | synrole | noncrole

conass.attribs = name.attrib
conass.model = mobj
conass = element conass {tref|(conass.attribs & conass.model)}

strass.attribs = name.attrib
strass.model = morphism
strass = element strass {tref|(strass.attribs & strass.model)}

assignment = conass | strass

structure.attribs = name.attrib & from.attrib
structure.class = (\include | assignment)* | element definition {morphism}
```

```
structure.model = metadata.class & structure.class
structure = element structure {tref|(structure.attribs & structure.model)}

view.attribs = structure.attribs & to.attrib
view.model = structure.model
view = element view {tref|(view.attribs & view.model)}

\include = element \include {from.attrib}

omdoc.class &= theory* & view* & nonconstit
```

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.6) Module ST
# $Id: omdocst.rnc 8812 2011-01-13 15:07:34Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/trunk/schema/rnc/pragmatic/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://omdoc.org/ns"

constitutive.class = symbol* & axiom* & definition* & imports*
nonconstit.class &= assertion* & type* & alternative* & example*
omdoc.class &= nonconstit.class & theory*

constitutive.attribs = id.attribs & attribute generated-from {SURI}?
sym.role.attrib = attribute role {"type" | "sort" | "object" |
                                  "binder" | "attribution" | "application" | "constant" |
                                  "semantic-attribution" | "error"}
theory-unique = xsd:NCName
scope.attrib = attribute scope {"global" | "local"}?
symbol.attribs = scope.attrib &
                 name.attrib &
                 constitutive.attribs &
                 sym.role.attrib?
symbol.model = metadata.class & type*
symbol = element symbol {tref|(symbol.attribs & symbol.model)}

axiom.attribs = constitutive.attribs &
                fori.attrib &
                attribute type {xsd:string}?
axiom.model = metadata.class & p*
axiom = element axiom {tref|(axiom.attribs & axiom.model)}

for.attrib = attribute for {SURI}

#simple definitions
exists.attrib = attribute existence {SURI}
def.simple.attribs = attribute type {"simple"} & exists.attrib?
def.simple = def.simple.attribs & mobj

#implicit definitions
unique.attrib = attribute uniqueness {SURI}
def.implicit.attribs = attribute type {"implicit"} & exists.attrib? & unique.attrib?
def.implicit = def.implicit.attribs & mobj

#definitions by (recursive) equations
exhaust.attrib = attribute exhaustivity {SURI}
consist.attrib = attribute consistency {SURI}
def.eq.attribs = attribute type {"pattern"|"inductive"}? &
                 exhaust.attrib? & consist.attrib?
def.eq.model = requation*,measure?,ordering?
def.eq = def.eq.attribs & def.eq.model

#all definition forms, add more by extending this.
defs.all = def.simple|def.implicit|def.eq

# Definitions contain math vernacular, FMPs and concept specifications.
# The latter define the set of concepts defined in this element.
# They can be reached under this name in the content dictionary
# of the name specified in the theory attribute of the definition.
definition.attribs =constitutive.attribs & for.attrib
definition = element definition {tref|(definition.attribs & defs.all)}

requation.attribs = id.attribs
requation.model = mobj,mobj
requation = element requation {tref|(requation.attribs & requation.model)}

measure.attribs = id.attribs
measure.model = mobj
measure = element measure {tref|(measure.attribs & measure.model)}
```

```
ordering.attribs = id.attribs & attribute terminating {SURI}?
ordering.model = mobj
ordering = element ordering {tref|(ordering.attribs & ordering.model)}

# the non-constitutive statements, they need a theory attribute
toplevel.attribs &= attribute theory {MURI}?

ded.status.class = "satisfiable" | "counter-satisfiable" | "no-consequence" |
                   "theorem" | "conter-theorem" | "contradictory-axioms" |
                   "tautologous-conclusion" | "tautology" | "equivalent" |
                   "conunter-equivalent" | "unsatisfiable-conclusion" | "unsatisfiable"

just-by.attrib = attribute just-by {SURI}
assertion.attribs =toplevel.attribs &
                   attribute type {assertiontype}? &
                   attribute status {ded.status.class}? &
                   just-by.attrib?
assertion.model = metadata.class & p*
assertion = element assertion {tref|(assertion.attribs & assertion.model)}
# the assertiontype has no formal meaning yet, it is solely for human consumption.
# 'just-by' is a list of URIRefs that point to proof objects, etc that justifies the status.

## just-by, points to the theorem justifying well-definedness
## entailed-by, entails, point to other (equivalent definitions
## entailed-by-thm, entails-thm point to the theorems justifying
## the entailment relation)
alternative.attribs =toplevel.attribs & for.attrib &
                     ((attribute equivalence {SURI},
                                 attribute equivalence-thm {SURI}) |
                                (attribute entailed-by {SURI} &
                                 attribute entails {SURI} &
                                 attribute entailed-by-thm {SURI} &
                                 attribute entails-thm {SURI}))
alternative.model = defs.all
alternative = element alternative {tref|(alternative.attribs & alternative.model)}

example.attribs =toplevel.attribs &
                 for.attrib &
                 attribute type {"for" | "against" }? &
                 attribute assertion {SURI}?
example.model = metadata.class & (p*,mobj)*
example = element example {tref|(example.attribs & example.model)}

theory.attribs &= id.attribs&
                  attribute cdurl {xsd:anyURI}?&
                  attribute cdbase {xsd:anyURI}?&
                  attribute cdreviewdate {xsd:date}?&
                  attribute cdversion {xsd:nonNegativeInteger}?&
                  attribute cdrevision {xsd:nonNegativeInteger}?&
                  attribute cdstatus {" official " | "experimental" | "private" | "obsolete"}?

theory.class &= tgroup*

imports.attribs = id.attribs & from.attrib
imports.model = metadata.class
imports = element imports {tref|(imports.attribs & imports.model)}

tgroup.attribs =constitutive.attribs & ogroup.attribs
tgroup.model = metadata.class & theory.class
tgroup = element omgroup {tref|(tgroup.attribs & tgroup.model)}
```

Appendix L

Module ADT: Abstract Data Types

The RNC module ADT specifies the grammar for abstract data types in OMDoc, see Chapter 47 for a discussion.

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.6) Module ADT
# $Id: omdocadt.rnc 8740 2010-09-27 09:16:55Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/trunk/schema/rnc/pragmatic/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)

default namespace omdoc = "http://omdoc.org/ns"
omdoc.class &= adt*

adt.sym.attrib = id.attribs,scope.attrib,attribute name {xsd:NCName}

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

adt.attribs = toplevel.attribs &
                    attribute parameters {list {xsd:NCName*}}?
adt.class = sortdef+
adt.model = metadata.class & adt.class
adt = element adt {tref|(adt.attribs & adt.model)}

adttype = "loose" | "generated" | "free"
sortdef.attribs = adt.sym.attrib &
                          attribute role {"sort"}? &
                          attribute type {adttype}?
sortdef.model = metadata.class & constructor* & insort* & recognizer?
sortdef = element sortdef {tref|(sortdef.attribs &sortdef.model)}

insort.attribs = attribute for {SURI}
insort.model = empty
insort = element insort {tref|(insort.attribs & insort.model)}

constructor.attribs = adt.sym.attrib & sym.role.attrib?
constructor.model = metadata.class & argument*
constructor = element constructor {tref|(constructor.attribs & constructor.model)}

recognizer.attribs = adt.sym.attrib & sym.role.attrib?
recognizer.model = metadata.class
recognizer = element recognizer {tref|(recognizer.attribs & recognizer.model)}

argument.attribs = empty
argument.model = type & selector?
argument = element argument {tref|(argument.attribs & argument.model)}

selector.attribs = adt.sym.attrib &
                          sym.role.attrib? &
                          attribute total {"yes" | "no"}?
selector.model = metadata.class
selector = element selector {tref|(selector.attribs & selector.model)}
```

Appendix M

Module PF: Proofs and Proof objects

The RNC module PF deals with mathematical argumentations and proofs in OMDoc, see Part XXIV for a discussion.

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.6) Module PF
# $Id: omdocpf.rnc 8740 2010-09-27 09:16:55Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/trunk/schema/rnc/pragmatic/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://omdoc.org/ns"

omdoc.class            &= proof* & proofobject*

proof.attribs =toplevel.attribs & fori.attrib
proof.model = metadata.class & omtext* & symbol* & definition* & derive* & hypothesis*
proof = element proof {tref|(proof.attribs & proof.model)}

proofobject.attribs = proof.attribs
proofobject.model = metadata.class & mobj
proofobject = element proofobject {tref|(proofobject.attribs & proofobject.model)}

derive.attribs = id.attribs & attribute type {"conclusion" | "gap"}?
derive.model = metadata.class & p* & method?
derive = element derive {tref|(derive.attribs & derive.model)}

hypothesis.attribs = id.attribs&attribute inductive {"yes" | "no"}?
hypothesis.model = metadata.class & p*
hypothesis = element hypothesis {tref|(hypothesis.attribs &hypothesis.model)}

method.attribs = id.attribs & xref.attrib?
method.model = mobj* & premise* & proof* & proofobject*
method = element method {tref|(method.attribs & method.model)}
# 'xref' is a pointer to the element defining the method

premise.attribs = xref.attrib & attribute rank {xsd:nonNegativeInteger}?
premise.model = empty
premise = element premise {tref|(premise.attribs & premise.model)}

# The rank of a premise specifies its importance in the inference rule.
# Rank 0 (the default) is a real premise, whereas positive rank signifies
# sideconditions of varying degree.
```

Appendix N

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 Part XXVII for a discussion).

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.6) Module EXT
# $Id: omdocext.rnc 8740 2010-09-27 09:16:55Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/trunk/schema/rnc/pragmatic/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://omdoc.org/ns"

omdocext.class = private* & code* & omlet*
omdoc.class &= omdocext.class

private.attribs = toplevel.attribs &
                  fori.attrib &
                  attribute requires {MMTURI}? &
                  attribute reformulates {SURI}?
private.model = metadata.class & data+
private = element private {private.attribs & private.model}
# reformulates is a URIref to the omdoc elements that are reformulated by the
# system-specific information in this element

code.attribs = private.attribs
code.model = metadata.class & data* & input* & output* & effect*
code = element code {tref|(code.attribs & code.model)}

input.attribs = id.attribs
input.model = p*
input = element input {tref|(input.attribs & input.model)}

output.attribs = id.attribs
output.model = p*
output = element output {tref|(output.attribs & output.model)}

effect.attribs = id.attribs
effect.model = p*
effect = element effect {tref|(effect.attribs & effect.model)}

data.attribs = id.attribs &
               attribute href {xsd:anyURI}? &
               attribute size {xsd:string}? &
               attribute pto {xsd:string}? &
               attribute pto-version {xsd:string}? &
               attribute original {"external" | "local"}?

data.textformat = "TeX"
data.text = data.attribs & attribute format {data.textformat}? & text
data.any = data.attribs & attribute format {xsd:anyURI}? & Anything
data.model = data.text | data.any
data = element data {tref|data.model}

omlet.attribs = id.attribs &
                attribute action {"display" | "execute" | "other"}? &
```

```
                              attribute show {"new" | "replace" | "embed" | "other"}? &
                              attribute actuate {"onPresent" | "onLoad" | "onRequest" | "other"}?
omlet.param = p* & param*
omlet.data = attribute data {xsd:anyURI}|(private|code)
omlet.model = metadata.class & omlet.param & omlet.data
omlet = element omlet {tref|(omlet.attribs & omlet.model)}

param.attribs = id.attribs &
                              attribute name {xsd:string} &
                              attribute value {xsd:string}? &
                              attribute valuetype {"data" | "ref" | "object"}?
param.model = mobj?
param = element param {tref|(param.attribs & param.model)}
```

Appendix O

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 Part XXVI for a discussion).

```
# A RelaxNG for Open Mathematical documents (OMDoc 1.6) Module PRES
# $Id: notation-strict.rnc 8937 2011-08-08 07:26:52Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/trunk/schema/rnc/strict/notation-strict.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2004-2008 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://omdoc.org/ns"
theory.class &= notation*

prototype.attribs = empty
prototype.model = protoexp
prototype = element prototype {tref|(prototype.attribs & prototype.model)}

protoexp = grammar {include "openmath/openmath3.rnc"
                      {start = omel
                       common.attributes = attribute id {xsd:ID}?&parent idrest.attribs}
                    omel |= parent proto.class
                    omvar |= parent proto.class
                    common.attributes &= parent ntn.attrib}
         | grammar {include "../../mathml3/mathml3.rnc" {start = ContExp}
                    ContExp |= parent proto.class
                    ci |= parent proto.class
                    CommonAtt &= parent ntn.attrib}

precedence.att = attribute precedence {xsd:integer}
context.att = attribute xml:lang {text}? &
                      attribute context {text}? &
                      attribute variant {text}?

format.att = attribute format {text}?

rendering.attribs = precedence.att? & context.att & format.att
rendering.model = renderexp
rendering = element rendering {tref|(rendering.attribs & rendering.model)}

renderexp = grammar {include "../../mathml3/mathml3-common.rnc" {start = PresentationExpression}
                                    include "../../ mathml3/mathml3-presentation.rnc"
                                    PresentationExpression |= parent render.class
                                    CommonAtt &= parent ntn.attrib
                                    mtable.content.class |= parent render.class
                                    mtr.content.class |= parent render.class}
                  | (pdata|render.class)*

pdata.attribs = empty
pdata.model = text
pdata = element pdata {pdata.attribs & pdata.model}

iterexp = grammar {include "../../mathml3/mathml3.rnc"
        {start = PresentationExpression|mtr|mlabeledtr|mtd}
                  PresentationExpression |= parent render.class
                  MathML.Common.attrib &= parent ntn.attrib
```

```
                    mtable.content.class |= parent render.class
                    mtr.content.class |= parent render.class}

notation.attribs = id.attribs & triple.att
notation.model = metadata.class & p* & prototype+ & rendering*
notation = element notation {tref|(notation.attribs & notation.model)}

# we extend the content and presentation models by metavariables
proto.class = exprlist | expr
render.class = render | iterate
ntn.attrib = attribute cr {text}? & attribute egroup {text}?

exprlist.attribs = name.attrib
exprlist.model = protoexp*
exprlist = element exprlist {exprlist.attribs & exprlist.model}

expr.attribs = name.attrib
expr.model = empty
expr = element expr {expr.attribs & expr.model}

iterate.attribs = name.attrib & precedence.att?
iterate.model = separator & iterexp*
iterate = element iterate {iterate.attribs & iterate.model}

render.attribs = name.attrib & precedence.att?
render.model = empty
render = element render {render.attribs & render.model}

separator.attribs = empty
separator.model = renderexp*
separator = element separator {separator.attribs & separator.model}
```

```
default namespace omdoc = "http://www.omdoc.org/ns"
datatypes xsd = "http://www.w3.org/2001/XMLSchema-datatypes"

##################################
#module level

omdoc.class &= notationset*

notationset.attribs = name.attrib? &
                      attribute base {MMTURI}? &
                      attribute includeDefault {bool}?
notationset.model = (\include | mmtnotation)*

notationset = element notationset {notationset.attribs & notationset.model}

##################################
#symbol level


mmtnotation = element notation {name.attrib?, attribute for {MMTURI}?, (simple | complex)}
simple = attribute role {simplerole}, attribute inherit {bool}?, pres*
complex = attribute role {complexrole}, attribute precedence {precedence}?,
          attribute fixity {fixity}?, attribute application-style {applicationstyle}?,
          attribute associativity {associativity}?, attribute implicit {int}?,
          (element main {pres*}? & element separator {pres*}? &
          element brackets {pres*}? & element nobrackets {pres*}? & element ebrackets {pres*}?)

fixity = "pre" | "post" | "in" | "inter" | "bind" | "special"
applicationstyle = "math" | "lc"
associativity = "none" | "left" | "right"


##################################
#presentation items
pres = \text | \element | newline | tab | components | recurse | component | mmtindex | id | ifpresent | nset | hole | elevel

\text = element text {attribute value {xsd:string}}
\element = element element {attribute prefix {xsd:string}?, attribute name {xsd:string}, (pres | \attribute)*}
\attribute = element attribute {attribute prefix {xsd:string}?, attribute name {xsd:string}, pres*}
newline = element newline {empty}
tab = element tab {empty}
components = element components {
   attribute begin {int}?, attribute end {int}?, attribute step {int}?,
   (element body {pres*}? & element separator {pres*}? & element pre {pres*}? & element post {pres*}?)
}
recurse = element recurse {attribute offset {int}?, prec.attrib?}
```

```
component = element component {attribute index {int}, prec.attrib?}
mmtindex = element index {attribute offset {int}}
id = element id {empty}
ifpresent = element ifpresent {attribute index {int}, element then {pres*}, element else {pres*}?}
nset = element nset {empty}
hole = element hole {pres*}
elevel = element elevel {empty}

prec.attrib = attribute precedence {precedence}

##################################
# datatypes

int = xsd:integer
bool = "yes" | "no"

simplerole.class = "Toplevel" | "Theory" | "View" | "DefinedView"
      | Constant | "Structure" | "DefinedStructure" | "Conass" | "Strass"
      | "toplevel" | "theory" | "view"
      | "constant" | "structure" | "conass" | "strass"
      | "variable"
simplerole = list {simplerole.class*}

Constant = "Element" | "Predicate" | "Sort" | "Proof" | "Axiom" | "Rule" | "Judgment" |
                    "Level" | "Binder" | "Key" | "Error"
complexrole.class = "variable" | "application" | "binding" | "attribution" |
                                  "morphism-application" | "identity" | "composition"
complexrole = list {complexrole.class*}
precedence = int | "infinity" | "-infinity"
```

Appendix P

Module QUIZ: Infrastructure for Assessments

The RNC module QUIZ provides a basic infrastructure for various kinds of exercises (see Part XXVIII for a discussion).

```
# A RelaxNG schema for Open Mathematical documents (OMDoc 1.6) Module QUIZ
# $Id: omdocquiz.rnc 8740 2010-09-27 09:16:55Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/trunk/schema/rnc/pragmatic/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://omdoc.org/ns"

omdoc.class &= exercise* & hint* & mc* & solution*

exercise.attribs = id.attribs & fori.attrib
exercise.model = metadata.class & p* & hint* & (solution*|mc*)
exercise = element exercise {tref|(exercise.attribs & exercise.model)}

hint.attribs =toplevel.attribs & fori.attrib
hint.model = metadata.class & p*
hint = element hint {tref|(hint.attribs & hint.model)}

solution.attribs =toplevel.attribs & fori.attrib
solution.model = metadata.class & p* & omdoc.class
solution = element solution {tref|(solution.attribs & solution.model)}

mc.attribs =toplevel.attribs & fori.attrib
mc.model = choice,hint?,answer
mc = element mc {tref|(mc.attribs & mc.model)}

choice.attribs = id.attribs
choice.model = metadata.class & p*
choice = element choice {tref|(choice.attribs & choice.model)}

answer.attribs = id.attribs & attribute verdict {"true" | "false"}?
answer.model = metadata.class & p*
answer = element answer {tref|(answer.attribs & answer.model)}
```

Part LVI

The RelaxNG Schemata for Mathematical Objects

For completeness we reprint the RelaxNG schemata for the external formats OMDoc makes use of.

Appendix Q

The RelaxNG Schema for OpenMath

For completeness we reprint the RelaxNG schema for OpenMath, the original can be found in the OpenMath3 standard under development

```
# RELAX NG Schema for OpenMath 3
# $Id: openmath3.rnc 8550 2009-11-07 06:38:23Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/trunk/schema/rnc/openmath/openmath3.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.content = xsd:string {pattern = "\s*(-\s?)?[0-9]+(\s[0-9]+)*\s*"}

OMI = element OMI {common.attributes & OMI.content}
# 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
56  OMA = element OMA { compound.attributes & omel+ }

    # binding constructor
    OMBIND = element OMBIND { compound.attributes & (omel, OMBVAR, omel)}

61  # the condition element
    OMC = element OMC {common.attributes & omel}

    # variables used in binding constructor
    OMBVAR = element OMBVAR { common.attributes & omvar+ }
66
    # error constructor
    OME = element OME { common.attributes & (OMS, (omel|OMFOREIGN)* )}

    # attribution constructor and attribute pair constructor
71  OMATTR = element OMATTR { compound.attributes & (OMATP, omel)}

    OMATP = element OMATP { compound.attributes & (OMS, (omel | OMFOREIGN) )+ }

    # foreign constructor
76  OMFOREIGN = element OMFOREIGN {compound.attributes &
                                                    attribute encoding {xsd:string}?&
                                                    (omel|notom)* }

    # Any elements not in the om namespace
81  # (valid om is allowed as a descendant)
    notom = text
    #  (element * - om:* {attribute * { text }*,(omel|notom)*}
    #   | text)

86  # reference constructor
    OMR = element OMR {common.attributes &attribute href {xsd:anyURI}}
```

Appendix R

The RelaxNG Schema for MathML

For completeness, we reprint the RelaxNG schema for MathML. It comes in three parts, the schema driver with common parts, and the parts for content- and presentation MathML which we will present in the next two subsections.

```
#     This is the Mathematical Markup Language (MathML) 3.0, an XML
#     application for describing mathematical notation and capturing
#     both its structure and content.
#
#     Copyright 1998−2009 W3C (MIT, ERCIM, Keio)
#
#     Use and distribution of this code are permitted under the terms
#     W3C Software Notice and License
#     http://www.w3.org/Consortium/Legal/2002/copyright−software−20021231


default namespace m = "http://www.w3.org/1998/Math/MathML"

## Content MathML
include "mathml3−content.rnc"

## Presentation MathML
include "mathml3−presentation.rnc"

## math and semantics common to both Content and Presentation
include "mathml3−common.rnc"
```

```
#     This is the Mathematical Markup Language (MathML) 3.0, an XML
#     application for describing mathematical notation and capturing
#     both its structure and content.
#
#     Copyright 1998−2009 W3C (MIT, ERCIM, Keio)
#
#     Use and distribution of this code are permitted under the terms
#     W3C Software Notice and License
#     http://www.w3.org/Consortium/Legal/2002/copyright−software−20021231


default namespace m = "http://www.w3.org/1998/Math/MathML"
namespace local = ""

start = math

math = element math {math.attributes,MathExpression*}
MathExpression = semantics

NonMathMLAtt = attribute (* − (local:*|m:*)) {xsd:string}

CommonDeprecatedAtt = attribute other {text}?

CommonAtt = attribute id {xsd:ID}?,
            attribute xref {text}?,
            attribute class {xsd:NMTOKENS}?,
```

```
                attribute style {xsd:string}?,
                attribute href {xsd:anyURI}?,
                CommonDeprecatedAtt,
                NonMathMLAtt*


    math.attributes = CommonAtt,
                      attribute display {"block" | "inline"}?,
                      attribute maxwidth {length}?,
                      attribute overflow {"linebreak" | "scroll" | "elide" | "truncate" | "scale"}?,
                      attribute altimg {xsd:anyURI}?,
                      attribute altimg-width {length}?,
                      attribute altimg-height {length}?,
                      attribute altimg-valign {length | "top" | "middle" | "bottom"}?,
                      attribute alttext {text}?,
                      attribute cdgroup {xsd:anyURI}?,
                      math.deprecatedattributes

    # the mathml3-presentation schema adds additional attributes
    # to the math element, all those valid on mstyle

    math.deprecatedattributes = attribute mode {xsd:string}?,
                                attribute macros {xsd:string}?


    name = attribute name {xsd:NCName}
    cd = attribute cd {xsd:NCName}

    src = attribute src {xsd:anyURI}?

    annotation = element annotation {annotation.attributes,text}

    annotation-xml.model = (MathExpression|anyElement)*

    anyElement = element (* - m:*) {(attribute * {text}|text| anyElement)*}

    annotation-xml = element annotation-xml {annotation.attributes,
                                             annotation-xml.model}
    annotation.attributes = CommonAtt,
                            cd?,
                            name?,
                            DefEncAtt,
                            src?

    DefEncAtt = attribute encoding {xsd:string}?,
                attribute definitionURL {xsd:anyURI}?

    semantics = element semantics {semantics.attributes,
                                   MathExpression,
                                   (annotation|annotation-xml)*}
    semantics.attributes = CommonAtt,DefEncAtt,cd?,name?


    length = xsd:string {
      pattern = '\s*((-?[0-9]*(\.[0-9]*)?(e[mx]|in|cm|mm|p[xtc]|%)?)|(negative)?((very){0,2}thi(n|ck)|medium)mathspace)\s*'
    }
```

Appendix S

Presentation MathML

```
1
    #     This is the Mathematical Markup Language (MathML) 3.0, an XML
    #     application for describing mathematical notation and capturing
    #     both its structure and content.
    #
6   #     Copyright 1998−2010 W3C (MIT, ERCIM, Keio)
    #
    #     Use and distribution of this code are permitted under the terms
    #     W3C Software Notice and License
    #     http://www.w3.org/Consortium/Legal/2002/copyright−software−20021231
11
    default namespace m = "http://www.w3.org/1998/Math/MathML"

    MathExpression |= PresentationExpression

16  ImpliedMrow = MathExpression*

    TableRowExpression = mtr|mlabeledtr

    TableCellExpression = mtd
21
    MstackExpression = MathExpression|mscarries|msline|msrow|msgroup

    MsrowExpression = MathExpression|none

26  MultiScriptExpression = (MathExpression|none),(MathExpression|none)

    mpadded−length = xsd:string {
      pattern = '\s*([\+\−]?[0−9]*(\.[0−9]*)?\s*((%?\s*(height|depth|width)?)|e[mx]|in|cm|mm|p[xtc]|((negative)?((very){0,2}thi(n|ck)|medium)m

31  linestyle  = "none" | "solid" | "dashed"

    verticalalign  =
          "top" |
          "bottom" |
36        "center" |
          "baseline" |
          "axis"

    columnalignstyle = "left" | "center" | "right"
41
    notationstyle =
        "longdiv" |
        "actuarial" |
        "radical" |
46      "box" |
        "roundedbox" |
        "circle" |
        "left" |
        "right" |
51      "top" |
        "bottom" |
        "updiagonalstrike" |
        "downdiagonalstrike" |
        "verticalstrike" |
56      "horizontalstrike" |
        "madruwb"
```

```
idref = text
unsigned-integer = xsd:unsignedLong
integer = xsd:integer
number = xsd:decimal

character = xsd:string {
  pattern = '\s*\S\s*'}

color = xsd:string {
  pattern = '\s*((#[0-9a-fA-F]{3}([0-9a-fA-F]{3})?)|[aA][qQ][uU][aA]|[bB][lL][aA][cC][kK]|[bB][lL][uU][eE]|[fF][uU][cC][hH][sS][iI][aA]|[gG]

group-alignment = "left" | "center" | "right" | "decimalpoint"
group-alignment-list = list {group-alignment+}
group-alignment-list-list = xsd:string {
  pattern = '(\s*\{\s*(left|center|right|decimalpoint)(\s+(left|center|right|decimalpoint))*\})*\s*' }
positive-integer = xsd:positiveInteger

TokenExpression = mi|mn|mo|mtext|mspace|ms

token.content = mglyph|malignmark|text

mi = element mi {mi.attributes, token.content*}
mi.attributes =
  CommonAtt,
  CommonPresAtt,
  TokenAtt

mn = element mn {mn.attributes, token.content*}
mn.attributes =
  CommonAtt,
  CommonPresAtt,
  TokenAtt

mo = element mo {mo.attributes, token.content*}
mo.attributes =
  CommonAtt,
  CommonPresAtt,
  TokenAtt,
  attribute form {"prefix" | "infix" | "postfix"}?,
  attribute fence {"true" | "false"}?,
  attribute separator {"true" | "false"}?,
  attribute lspace {length}?,
  attribute rspace {length}?,
  attribute stretchy {"true" | "false"}?,
  attribute symmetric {"true" | "false"}?,
  attribute maxsize {length | "infinity"}?,
  attribute minsize {length}?,
  attribute largeop {"true" | "false"}?,
  attribute movablelimits {"true" | "false"}?,
  attribute accent {"true" | "false"}?,
  attribute linebreak {"auto" | "newline" | "nobreak" | "goodbreak" | "badbreak"}?,
  attribute lineleading {length}?,
  attribute linebreakstyle {"before" | "after" | "duplicate" | "infixlinebreakstyle"}?,
  attribute linebreakmultchar {text}?,
  attribute indentalign {"left" | "center" | "right" | "auto" | "id"}?,
  attribute indentshift {length}?,
  attribute indenttarget {idref}?,
  attribute indentalignfirst {"left" | "center" | "right" | "auto" | "id" | "indentalign"}?,
  attribute indentshiftfirst {length | "indentshift"}?,
  attribute indentalignlast {"left" | "center" | "right" | "auto" | "id" | "indentalign"}?,
  attribute indentshiftlast {length | "indentshift"}?

mtext = element mtext {mtext.attributes, token.content*}
mtext.attributes =
  CommonAtt,
  CommonPresAtt,
  TokenAtt

mspace = element mspace {mspace.attributes, empty}
mspace.attributes =
  CommonAtt,
  CommonPresAtt,
  TokenAtt,
  attribute width {length}?,
```

```
    attribute height {length}?,
    attribute depth {length}?,
    attribute linebreak {"auto" | "newline" | "nobreak" | "goodbreak" | "badbreak" | "indentingnewline"}?

ms = element ms {ms.attributes, token.content*}
ms.attributes =
    CommonAtt,
    CommonPresAtt,
    TokenAtt,
    attribute lquote {text}?,
    attribute rquote {text}?

mglyph = element mglyph {mglyph.attributes,mglyph.deprecatedattributes,empty}
mglyph.attributes =
    CommonAtt, CommonPresAtt,
    attribute src {xsd:anyURI}?,
    attribute width {length}?,
    attribute height {length}?,
    attribute valign {length}?,
    attribute alt {text}?
mglyph.deprecatedattributes =
    attribute index {integer}?,
    attribute mathvariant {"normal" | "bold" | "italic" | "bold-italic" | "double-struck" | "bold-fraktur" | "script" | "bold-script" | "fraktu
    attribute mathsize {"small" | "normal" | "big" | length}?,
    DeprecatedTokenAtt

msline = element msline {msline.attributes,empty}
msline.attributes =
    CommonAtt, CommonPresAtt,
    attribute position {integer}?,
    attribute length {unsigned-integer}?,
    attribute leftoverhang {length}?,
    attribute rightoverhang {length}?,
    attribute mslinethickness {length | "thin" | "medium" | "thick"}?

none = element none {none.attributes,empty}
none.attributes =
    CommonAtt,
    CommonPresAtt

mprescripts = element mprescripts {mprescripts.attributes,empty}
mprescripts.attributes =
    CommonAtt,
    CommonPresAtt


CommonPresAtt =
    attribute mathcolor {color}?,
    attribute mathbackground {color | "transparent"}?

TokenAtt =
    attribute mathvariant {"normal" | "bold" | "italic" | "bold-italic" | "double-struck" | "bold-fraktur" | "script" | "bold-script" | "fraktu
    attribute mathsize {"small" | "normal" | "big" | length}?,
    attribute dir {"ltr" | "rtl"}?,
    DeprecatedTokenAtt

DeprecatedTokenAtt =
    attribute fontfamily {text}?,
    attribute fontweight {"normal" | "bold"}?,
    attribute fontstyle {"normal" | "italic"}?,
    attribute fontsize {length}?,
    attribute color {color}?,
    attribute background {color | "transparent"}?

MalignExpression = maligngroup|malignmark

malignmark = element malignmark {malignmark.attributes, empty}
malignmark.attributes =
    CommonAtt, CommonPresAtt,
    attribute edge {"left" | "right"}?

maligngroup = element maligngroup {maligngroup.attributes, empty}
maligngroup.attributes =
    CommonAtt, CommonPresAtt,
    attribute groupalign {"left" | "center" | "right" | "decimalpoint"}?
```

```
PresentationExpression = TokenExpression|MalignExpression|
                         mrow|mfrac|msqrt|mroot|mstyle|merror|mpadded|mphantom|
                         mfenced|menclose|msub|msup|msubsup|munder|mover|munderover|
                         mmultiscripts|mtable|mstack|mlongdiv|maction


mrow = element mrow {mrow.attributes, MathExpression*}
mrow.attributes =
  CommonAtt, CommonPresAtt,
  attribute dir {"ltr" | "rtl"}?

mfrac = element mfrac {mfrac.attributes, MathExpression, MathExpression}
mfrac.attributes =
  CommonAtt, CommonPresAtt,
  attribute linethickness {length | "thin" | "medium" | "thick"}?,
  attribute numalign {"left" | "center" | "right"}?,
  attribute denomalign {"left" | "center" | "right"}?,
  attribute bevelled {"true" | "false"}?

msqrt = element msqrt {msqrt.attributes, ImpliedMrow}
msqrt.attributes =
  CommonAtt, CommonPresAtt

mroot = element mroot {mroot.attributes, MathExpression, MathExpression}
mroot.attributes =
  CommonAtt, CommonPresAtt

mstyle = element mstyle {mstyle.attributes, ImpliedMrow}
mstyle.attributes =
  CommonAtt, CommonPresAtt,
  mstyle.specificattributes ,
  mstyle.generalattributes ,
  mstyle.deprecatedattributes

mstyle.specificattributes =
  attribute scriptlevel {integer}?,
  attribute displaystyle {"true" | "false"}?,
  attribute scriptsizemultiplier {number}?,
  attribute scriptminsize {length}?,
  attribute infixlinebreakstyle {"before" | "after" | "duplicate"}?,
  attribute decimalpoint {character}?

mstyle.generalattributes =
  attribute accent {"true" | "false"}?,
  attribute accentunder {"true" | "false"}?,
  attribute align {"left" | "right" | "center"}?,
  attribute alignmentscope {list {("true" | "false") +}}?,
  attribute bevelled {"true" | "false"}?,
  attribute charalign {"left" | "center" | "right"}?,
  attribute charspacing {length | "loose" | "medium" | "tight"}?,
  attribute close {text}?,
  attribute columnalign {list {columnalignstyle+} }?,
  attribute columnlines {list {linestyle +}}?,
  attribute columnspacing {list {(length) +}}?,
  attribute columnspan {positive-integer}?,
  attribute columnwidth {list {("auto" | length | "fit") +}}?,
  attribute crossout {list {("none" | "updiagonalstrike" | "downdiagonalstrike" | "verticalstrike" | "horizontalstrike")*}}?,
  attribute denomalign {"left" | "center" | "right"}?,
  attribute depth {length}?,
  attribute dir {"ltr" | "rtl"}?,
  attribute edge {"left" | "right"}?,
  attribute equalcolumns {"true" | "false"}?,
  attribute equalrows {"true" | "false"}?,
  attribute fence {"true" | "false"}?,
  attribute form {"prefix" | "infix" | "postfix"}?,
  attribute frame {linestyle}?,
  attribute framespacing {list {length, length}}?,
  attribute groupalign {group-alignment-list-list}?,
  attribute height {length}?,
  attribute indentalign {"left" | "center" | "right" | "auto" | "id"}?,
  attribute indentalignfirst {"left" | "center" | "right" | "auto" | "id" | "indentalign"}?,
  attribute indentalignlast {"left" | "center" | "right" | "auto" | "id" | "indentalign"}?,
  attribute indentshift {length}?,
  attribute indentshiftfirst {length | "indentshift"}?,
  attribute indentshiftlast {length | "indentshift"}?,
```

```
    attribute indenttarget {idref}?,
    attribute largeop {"true" | "false"}?,
    attribute leftoverhang {length}?,
    attribute length {unsigned-integer}?,
    attribute linebreak {"auto" | "newline" | "nobreak" | "goodbreak" | "badbreak"}?,
    attribute linebreakmultchar {text}?,
    attribute linebreakstyle {"before" | "after" | "duplicate" | "infixlinebreakstyle"}?,
    attribute lineleading {length}?,
    attribute linethickness {length | "thin" | "medium" | "thick"}?,
    attribute location {"w" | "nw" | "n" | "ne" | "e" | "se" | "s" | "sw"}?,
    attribute longdivstyle {"lefttop" | "stackedrightright" | "mediumstackedrightright" | "shortstackedrightright" | "righttop" | "left/\right"
    attribute lquote {text}?,
    attribute lspace {length}?,
    attribute mathsize {"small" | "normal" | "big" | length}?,
    attribute mathvariant {"normal" | "bold" | "italic" | "bold-italic" | "double-struck" | "bold-fraktur" | "script" | "bold-script" | "fraktu
    attribute maxsize {length | "infinity"}?,
    attribute minlabelspacing {length}?,
    attribute minsize {length}?,
    attribute movablelimits {"true" | "false"}?,
    attribute mslinethickness {length | "thin" | "medium" | "thick"}?,
    attribute notation {text}?,
    attribute numalign {"left" | "center" | "right"}?,
    attribute open {text}?,
    attribute position {integer}?,
    attribute rightoverhang {length}?,
    attribute rowalign { list { verticalalign +} }?,
    attribute rowlines { list { linestyle +}}?,
    attribute rowspacing {list {(length) +}}?,
    attribute rowspan {positive-integer}?,
    attribute rquote {text}?,
    attribute rspace {length}?,
    attribute selection {positive-integer}?,
    attribute separator {"true" | "false"}?,
    attribute separators {text}?,
    attribute shift {integer}?,
    attribute side {"left" | "right" | "leftoverlap" | "rightoverlap"}?,
    attribute stackalign {"left" | "center" | "right" | "decimalpoint"}?,
    attribute stretchy {"true" | "false"}?,
    attribute subscriptshift {length}?,
    attribute superscriptshift {length}?,
    attribute symmetric {"true" | "false"}?,
    attribute valign {length}?,
    attribute width {length}?

mstyle.deprecatedattributes =
    DeprecatedTokenAtt,
    attribute veryverythinmathspace {length}?,
    attribute verythinmathspace {length}?,
    attribute thinmathspace {length}?,
    attribute mediummathspace {length}?,
    attribute thickmathspace {length}?,
    attribute verythickmathspace {length}?,
    attribute veryverythickmathspace {length}?

math.attributes &= CommonPresAtt
math.attributes &= mstyle.specificattributes
math.attributes &= mstyle.generalattributes



merror = element merror {merror.attributes, ImpliedMrow}
merror.attributes =
    CommonAtt, CommonPresAtt


mpadded = element mpadded {mpadded.attributes, ImpliedMrow}
mpadded.attributes =
    CommonAtt, CommonPresAtt,
    attribute height {mpadded-length}?,
    attribute depth {mpadded-length}?,
    attribute width {mpadded-length}?,
    attribute lspace {mpadded-length}?,
    attribute voffset {mpadded-length}?


mphantom = element mphantom {mphantom.attributes, ImpliedMrow}
mphantom.attributes =
    CommonAtt, CommonPresAtt
```

```
mfenced = element mfenced {mfenced.attributes, MathExpression*}
mfenced.attributes =
  CommonAtt, CommonPresAtt,
  attribute open {text}?,
  attribute close {text}?,
  attribute separators {text}?

menclose = element menclose {menclose.attributes, ImpliedMrow}
menclose.attributes =
  CommonAtt, CommonPresAtt,
  attribute notation {text}?

msub = element msub {msub.attributes, MathExpression, MathExpression}
msub.attributes =
  CommonAtt, CommonPresAtt,
  attribute subscriptshift {length}?

msup = element msup {msup.attributes, MathExpression, MathExpression}
msup.attributes =
  CommonAtt, CommonPresAtt,
  attribute superscriptshift {length}?

msubsup = element msubsup {msubsup.attributes, MathExpression, MathExpression, MathExpression}
msubsup.attributes =
  CommonAtt, CommonPresAtt,
  attribute subscriptshift {length}?,
  attribute superscriptshift {length}?

munder = element munder {munder.attributes, MathExpression, MathExpression}
munder.attributes =
  CommonAtt, CommonPresAtt,
  attribute accentunder {"true" | "false"}?,
  attribute align {"left" | "right" | "center"}?

mover = element mover {mover.attributes, MathExpression, MathExpression}
mover.attributes =
  CommonAtt, CommonPresAtt,
  attribute accent {"true" | "false"}?,
  attribute align {"left" | "right" | "center"}?

munderover = element munderover {munderover.attributes, MathExpression, MathExpression, MathExpression}
munderover.attributes =
  CommonAtt, CommonPresAtt,
  attribute accent {"true" | "false"}?,
  attribute accentunder {"true" | "false"}?,
  attribute align {"left" | "right" | "center"}?

mmultiscripts = element mmultiscripts {mmultiscripts.attributes, MathExpression,MultiScriptExpression*,(mprescripts,MultiScriptExpression*)
mmultiscripts.attributes =
  msubsup.attributes

mtable = element mtable {mtable.attributes, TableRowExpression*}
mtable.attributes =
  CommonAtt, CommonPresAtt,
  attribute align {xsd:string {
    pattern ='\s*(top|bottom|center|baseline|axis)\s*[0-9]*'}}?,
  attribute rowalign { list { verticalalign +} }?,
  attribute columnalign {list {columnalignstyle+} }?,
  attribute groupalign {group-alignment-list-list}?,
  attribute alignmentscope {list {("true" | "false") +}}?,
  attribute columnwidth {list {("auto" | length | "fit") +}}?,
  attribute width {"auto" | length}?,
  attribute rowspacing {list {(length) +}}?,
  attribute columnspacing {list {(length) +}}?,
  attribute rowlines { list { linestyle +}}?,
  attribute columnlines { list { linestyle +}}?,
  attribute frame { linestyle }?,
  attribute framespacing {list {length, length}}?,
  attribute equalrows {"true" | "false"}?,
  attribute equalcolumns {"true" | "false"}?,
```

```
    attribute displaystyle {"true" | "false"}?,
    attribute side {"left" | "right" | "leftoverlap" | "rightoverlap"}?,
    attribute minlabelspacing {length}?

mlabeledtr = element mlabeledtr {mlabeledtr.attributes, TableCellExpression+}
mlabeledtr.attributes =
    mtr.attributes

mtr = element mtr {mtr.attributes, TableCellExpression*}
mtr.attributes =
    CommonAtt, CommonPresAtt,
    attribute rowalign {"top" | "bottom" | "center" | "baseline" | "axis"}?,
    attribute columnalign {list {columnalignstyle+} }?,
    attribute groupalign {group-alignment-list-list}?

mtd = element mtd {mtd.attributes, ImpliedMrow}
mtd.attributes =
    CommonAtt, CommonPresAtt,
    attribute rowspan {positive-integer}?,
    attribute columnspan {positive-integer}?,
    attribute rowalign {"top" | "bottom" | "center" | "baseline" | "axis"}?,
    attribute columnalign {columnalignstyle}?,
    attribute groupalign {group-alignment-list}?

mstack = element mstack {mstack.attributes, MstackExpression*}
mstack.attributes =
    CommonAtt, CommonPresAtt,
    attribute align {xsd:string {
      pattern ='\s*(top|bottom|center|baseline|axis)\s*[0-9]*'}}?,
    attribute stackalign {"left" | "center" | "right" | "decimalpoint"}?,
    attribute charalign {"left" | "center" | "right"}?,
    attribute charspacing {length | "loose" | "medium" | "tight"}?

mlongdiv = element mlongdiv {mlongdiv.attributes, MstackExpression,MstackExpression,MstackExpression+}
mlongdiv.attributes =
    msgroup.attributes,
    attribute longdivstyle {"lefttop" | "stackedrightright" | "mediumstackedrightright" | "shortstackedrightright" | "righttop" | "left/\right"

msgroup = element msgroup {msgroup.attributes, MstackExpression*}
msgroup.attributes =
    CommonAtt, CommonPresAtt,
    attribute position {integer}?,
    attribute shift {integer}?

msrow = element msrow {msrow.attributes, MsrowExpression*}
msrow.attributes =
    CommonAtt, CommonPresAtt,
    attribute position {integer}?

mscarries = element mscarries {mscarries.attributes, (MsrowExpression|mscarry)*}
mscarries.attributes =
    CommonAtt, CommonPresAtt,
    attribute position {integer}?,
    attribute location {"w" | "nw" | "n" | "ne" | "e" | "se" | "s" | "sw"}?,
    attribute crossout { list {("none" | "updiagonalstrike" | "downdiagonalstrike" | "verticalstrike" | "horizontalstrike")*}}?,
    attribute scriptsizemultiplier {number}?

mscarry = element mscarry {mscarry.attributes, MsrowExpression*}
mscarry.attributes =
    CommonAtt, CommonPresAtt,
    attribute location {"w" | "nw" | "n" | "ne" | "e" | "se" | "s" | "sw"}?,
    attribute crossout { list {("none" | "updiagonalstrike" | "downdiagonalstrike" | "verticalstrike" | "horizontalstrike")*}}?

maction = element maction {maction.attributes, MathExpression+}
maction.attributes =
    CommonAtt, CommonPresAtt,
    attribute actiontype {text}?,
    attribute selection {positive-integer}?
```

Appendix T

Strict Content MathML

```
#     This is the Mathematical Markup Language (MathML) 3.0, an XML
#     application for describing mathematical notation and capturing
#     both its structure and content.
#
#     Copyright 1998−2009 W3C (MIT, ERCIM, Keio)
#
#     Use and distribution of this code are permitted under the terms
#     W3C Software Notice and License
#     http://www.w3.org/Consortium/Legal/2002/copyright−software−20021231


default namespace m = "http://www.w3.org/1998/Math/MathML"

ContExp = semantics−contexp | cn | ci | csymbol | apply | bind | share | cerror | cbytes | cs

cn = element cn {cn.attributes,cn.content}
cn.content = text
cn.attributes = attribute type {"integer" | "real" | "double" | "hexdouble"}

semantics−ci = element semantics {semantics.attributes,(ci|semantics−ci),
  (annotation|annotation−xml)*}

semantics−contexp = element semantics {semantics.attributes,ContExp,
  (annotation|annotation−xml)*}

ci = element ci {ci.attributes, ci.content}
ci.attributes = CommonAtt, ci.type?
ci.type = attribute type {"integer" | "rational" | "real" | "complex" | "complex−polar" | "complex−cartesian" | "constant" | "function" | "v
ci.content = text


csymbol = element csymbol {csymbol.attributes,csymbol.content}

SymbolName = xsd:NCName
csymbol.attributes = CommonAtt, cd
csymbol.content = SymbolName

BvarQ = bvar*
bvar = element bvar { ci | semantics−ci}

apply = element apply {CommonAtt,apply.content}
apply.content = ContExp+


bind = element bind {CommonAtt,bind.content}
bind.content = ContExp,bvar*,ContExp

share = element share {CommonAtt, src, empty}

cerror = element cerror {cerror.attributes, csymbol, ContExp*}
cerror.attributes = CommonAtt

cbytes = element cbytes {cbytes.attributes, base64}
cbytes.attributes = CommonAtt
base64 = xsd:base64Binary

cs = element cs {cs.attributes, text}
cs.attributes = CommonAtt
```

59
MathExpression |= ContExp

Appendix U

Pragmatic Content MathML

```
#     This is the Mathematical Markup Language (MathML) 3.0, an XML
#     application for describing mathematical notation and capturing
#     both its structure and content.
#
#     Copyright 1998−2010 W3C (MIT, ERCIM, Keio)
#
#     Use and distribution of this code are permitted under the terms
#     W3C Software Notice and License
#     http://www.w3.org/Consortium/Legal/2002/copyright−software−20021231

include "mathml3−strict−content.rnc"{
  cn.content = (text | mglyph | sep | PresentationExpression)*
  cn.attributes = CommonAtt, DefEncAtt, attribute type {text}?, base?

  ci.attributes = CommonAtt, DefEncAtt, ci.type?
  ci.type = attribute type {text}
  ci.content = (text | mglyph | PresentationExpression)*

  csymbol.attributes = CommonAtt, DefEncAtt, attribute type {text}?,cd?
  csymbol.content = (text | mglyph | PresentationExpression)*

  bvar = element bvar { (ci | semantics−ci) & degree?}

  cbytes.attributes = CommonAtt, DefEncAtt

  cs.attributes = CommonAtt, DefEncAtt

  apply.content = ContExp+ | (ContExp, BvarQ, Qualifier*, ContExp*)

  bind.content = apply.content
}

base = attribute base {text}


sep = element sep {empty}
PresentationExpression |= notAllowed


DomainQ = (domainofapplication|condition|interval|(lowlimit,uplimit?))*
domainofapplication = element domainofapplication {ContExp}
condition = element condition {ContExp}
uplimit = element uplimit {ContExp}
lowlimit = element lowlimit {ContExp}

Qualifier = DomainQ|degree|momentabout|logbase
degree = element degree {ContExp}
momentabout = element momentabout {ContExp}
logbase = element logbase {ContExp}

type = attribute type {text}
order = attribute order {"numeric" | "lexicographic"}
closure = attribute closure {text}


ContExp |= piecewise
```

```
piecewise = element piecewise {CommonAtt, DefEncAtt,(piece* & otherwise?)}

piece = element piece {CommonAtt, DefEncAtt, ContExp, ContExp}

otherwise = element otherwise {CommonAtt, DefEncAtt, ContExp}


DeprecatedContExp = reln | fn | declare
ContExp |= DeprecatedContExp

reln = element reln {ContExp*}
fn = element fn {ContExp}
declare = element declare {attribute type {xsd:string}?,
                           attribute scope {xsd:string}?,
                           attribute nargs {xsd:nonNegativeInteger}?,
                           attribute occurrence {"prefix"|"infix"|"function-model"}?,
                           DefEncAtt,
                           ContExp+}


interval.class = interval
ContExp |= interval.class


interval = element interval { CommonAtt, DefEncAtt,closure?, ContExp,ContExp}

unary-functional.class = inverse | ident | domain | codomain | image | ln | log | moment
ContExp |= unary-functional.class


inverse = element inverse { CommonAtt, DefEncAtt, empty}
ident = element ident { CommonAtt, DefEncAtt, empty}
domain = element domain { CommonAtt, DefEncAtt, empty}
codomain = element codomain { CommonAtt, DefEncAtt, empty}
image = element image { CommonAtt, DefEncAtt, empty}
ln = element ln { CommonAtt, DefEncAtt, empty}
log = element log { CommonAtt, DefEncAtt, empty}
moment = element moment { CommonAtt, DefEncAtt, empty}

lambda.class = lambda
ContExp |= lambda.class


lambda = element lambda { CommonAtt, DefEncAtt, BvarQ, DomainQ, ContExp}

nary-functional.class = compose
ContExp |= nary-functional.class


compose = element compose { CommonAtt, DefEncAtt, empty}

binary-arith.class = quotient | divide | minus | power | rem | root
ContExp |= binary-arith.class


quotient = element quotient { CommonAtt, DefEncAtt, empty}
divide = element divide { CommonAtt, DefEncAtt, empty}
minus = element minus { CommonAtt, DefEncAtt, empty}
power = element power { CommonAtt, DefEncAtt, empty}
rem = element rem { CommonAtt, DefEncAtt, empty}
root = element root { CommonAtt, DefEncAtt, empty}

unary-arith.class = factorial | minus | root | abs | conjugate | arg | real | imaginary | floor | ceiling | exp
ContExp |= unary-arith.class


factorial = element factorial { CommonAtt, DefEncAtt, empty}
abs = element abs { CommonAtt, DefEncAtt, empty}
conjugate = element conjugate { CommonAtt, DefEncAtt, empty}
arg = element arg { CommonAtt, DefEncAtt, empty}
real = element real { CommonAtt, DefEncAtt, empty}
imaginary = element imaginary { CommonAtt, DefEncAtt, empty}
floor = element floor { CommonAtt, DefEncAtt, empty}
ceiling = element ceiling { CommonAtt, DefEncAtt, empty}
exp = element exp { CommonAtt, DefEncAtt, empty}

nary-minmax.class = max | min
ContExp |= nary-minmax.class
```

```
max = element max { CommonAtt, DefEncAtt, empty}
min = element min { CommonAtt, DefEncAtt, empty}

nary-arith.class = plus | times | gcd | lcm
ContExp |= nary-arith.class

plus = element plus { CommonAtt, DefEncAtt, empty}
times = element times { CommonAtt, DefEncAtt, empty}
gcd = element gcd { CommonAtt, DefEncAtt, empty}
lcm = element lcm { CommonAtt, DefEncAtt, empty}

nary-logical.class = and | or | xor
ContExp |= nary-logical.class

and = element and { CommonAtt, DefEncAtt, empty}
or = element or { CommonAtt, DefEncAtt, empty}
xor = element xor { CommonAtt, DefEncAtt, empty}

unary-logical.class = not
ContExp |= unary-logical.class

not = element not { CommonAtt, DefEncAtt, empty}

binary-logical.class = implies | equivalent
ContExp |= binary-logical.class

implies = element implies { CommonAtt, DefEncAtt, empty}
equivalent = element equivalent { CommonAtt, DefEncAtt, empty}

quantifier.class = forall | exists
ContExp |= quantifier.class

forall = element forall { CommonAtt, DefEncAtt, empty}
exists = element exists { CommonAtt, DefEncAtt, empty}

nary-reln.class = eq | gt | lt | geq | leq
ContExp |= nary-reln.class

eq = element eq { CommonAtt, DefEncAtt, empty}
gt = element gt { CommonAtt, DefEncAtt, empty}
lt = element lt { CommonAtt, DefEncAtt, empty}
geq = element geq { CommonAtt, DefEncAtt, empty}
leq = element leq { CommonAtt, DefEncAtt, empty}

binary-reln.class = neq | approx | factorof | tendsto
ContExp |= binary-reln.class

neq = element neq { CommonAtt, DefEncAtt, empty}
approx = element approx { CommonAtt, DefEncAtt, empty}
factorof = element factorof { CommonAtt, DefEncAtt, empty}
tendsto = element tendsto { CommonAtt, DefEncAtt, type?, empty}

int.class = int
ContExp |= int.class

int = element int { CommonAtt, DefEncAtt, empty}

Differential-Operator.class = diff
ContExp |= Differential-Operator.class

diff = element diff { CommonAtt, DefEncAtt, empty}

partialdiff.class = partialdiff
ContExp |= partialdiff.class

partialdiff = element partialdiff { CommonAtt, DefEncAtt, empty}

unary-veccalc.class = divergence | grad | curl | laplacian
```

```
ContExp |= unary−veccalc.class

divergence = element divergence { CommonAtt, DefEncAtt, empty}
grad = element grad { CommonAtt, DefEncAtt, empty}
curl = element curl { CommonAtt, DefEncAtt, empty}
laplacian = element laplacian { CommonAtt, DefEncAtt, empty}

nary−setlist−constructor.class = set | \ list
ContExp |= nary−setlist−constructor.class


set = element set { CommonAtt, DefEncAtt, type?, BvarQ∗, DomainQ∗, ContExp∗}
\ list = element \list { CommonAtt, DefEncAtt, order?, BvarQ∗, DomainQ∗, ContExp∗}

nary−set.class = union | intersect | cartesianproduct
ContExp |= nary−set.class


union = element union { CommonAtt, DefEncAtt, empty}
intersect = element intersect { CommonAtt, DefEncAtt, empty}
cartesianproduct = element cartesianproduct { CommonAtt, DefEncAtt, empty}

binary−set.class = in | notin | notsubset | notprsubset | setdiff
ContExp |= binary−set.class


in = element in { CommonAtt, DefEncAtt, empty}
notin = element notin { CommonAtt, DefEncAtt, empty}
notsubset = element notsubset { CommonAtt, DefEncAtt, empty}
notprsubset = element notprsubset { CommonAtt, DefEncAtt, empty}
setdiff = element setdiff { CommonAtt, DefEncAtt, empty}

nary−set−reln.class = subset | prsubset
ContExp |= nary−set−reln.class


subset = element subset { CommonAtt, DefEncAtt, empty}
prsubset = element prsubset { CommonAtt, DefEncAtt, empty}

unary−set.class = card
ContExp |= unary−set.class

card = element card { CommonAtt, DefEncAtt, empty}

sum.class = sum
ContExp |= sum.class


sum = element sum { CommonAtt, DefEncAtt, empty}

product.class = product
ContExp |= product.class


product = element product { CommonAtt, DefEncAtt, empty}

limit . class = limit
ContExp |= limit.class


limit = element limit { CommonAtt, DefEncAtt, empty}

unary−elementary.class = sin | cos | tan | sec | csc | cot | sinh | cosh | tanh | sech | csch | coth | arcsin | arccos | arctan | arccosh | a
ContExp |= unary−elementary.class

sin = element sin { CommonAtt, DefEncAtt, empty}
cos = element cos { CommonAtt, DefEncAtt, empty}
tan = element tan { CommonAtt, DefEncAtt, empty}
sec = element sec { CommonAtt, DefEncAtt, empty}
csc = element csc { CommonAtt, DefEncAtt, empty}
cot = element cot { CommonAtt, DefEncAtt, empty}
sinh = element sinh { CommonAtt, DefEncAtt, empty}
cosh = element cosh { CommonAtt, DefEncAtt, empty}
tanh = element tanh { CommonAtt, DefEncAtt, empty}
sech = element sech { CommonAtt, DefEncAtt, empty}
csch = element csch { CommonAtt, DefEncAtt, empty}
coth = element coth { CommonAtt, DefEncAtt, empty}
```

```
arcsin = element arcsin { CommonAtt, DefEncAtt, empty}
arccos = element arccos { CommonAtt, DefEncAtt, empty}
arctan = element arctan { CommonAtt, DefEncAtt, empty}
arccosh = element arccosh { CommonAtt, DefEncAtt, empty}
arccot = element arccot { CommonAtt, DefEncAtt, empty}
arccoth = element arccoth { CommonAtt, DefEncAtt, empty}
arccsc = element arccsc { CommonAtt, DefEncAtt, empty}
arccsch = element arccsch { CommonAtt, DefEncAtt, empty}
arcsec = element arcsec { CommonAtt, DefEncAtt, empty}
arcsech = element arcsech { CommonAtt, DefEncAtt, empty}
arcsinh = element arcsinh { CommonAtt, DefEncAtt, empty}
arctanh = element arctanh { CommonAtt, DefEncAtt, empty}

nary-stats.class = mean | sdev | variance | median | mode
ContExp |= nary-stats.class


mean = element mean { CommonAtt, DefEncAtt, empty}
sdev = element sdev { CommonAtt, DefEncAtt, empty}
variance = element variance { CommonAtt, DefEncAtt, empty}
median = element median { CommonAtt, DefEncAtt, empty}
mode = element mode { CommonAtt, DefEncAtt, empty}

nary-constructor.class = vector | matrix | matrixrow
ContExp |= nary-constructor.class


vector = element vector { CommonAtt, DefEncAtt, BvarQ, DomainQ, ContExp*}
matrix = element matrix { CommonAtt, DefEncAtt, BvarQ, DomainQ, ContExp*}
matrixrow = element matrixrow { CommonAtt, DefEncAtt, BvarQ, DomainQ, ContExp*}

unary-linalg.class = determinant | transpose
ContExp |= unary-linalg.class


determinant = element determinant { CommonAtt, DefEncAtt, empty}
transpose = element transpose { CommonAtt, DefEncAtt, empty}

nary-linalg.class = selector
ContExp |= nary-linalg.class


selector = element selector { CommonAtt, DefEncAtt, empty}

binary-linalg.class = vectorproduct | scalarproduct | outerproduct
ContExp |= binary-linalg.class


vectorproduct = element vectorproduct { CommonAtt, DefEncAtt, empty}
scalarproduct = element scalarproduct { CommonAtt, DefEncAtt, empty}
outerproduct = element outerproduct { CommonAtt, DefEncAtt, empty}

constant-set.class = integers | reals | rationals | naturalnumbers | complexes | primes | emptyset
ContExp |= constant-set.class


integers = element integers { CommonAtt, DefEncAtt, empty}
reals = element reals { CommonAtt, DefEncAtt, empty}
rationals = element rationals { CommonAtt, DefEncAtt, empty}
naturalnumbers = element naturalnumbers { CommonAtt, DefEncAtt, empty}
complexes = element complexes { CommonAtt, DefEncAtt, empty}
primes = element primes { CommonAtt, DefEncAtt, empty}
emptyset = element emptyset { CommonAtt, DefEncAtt, empty}

constant-arith.class = exponentiale | imaginaryi | notanumber | true | false | pi | eulergamma | infinity
ContExp |= constant-arith.class


exponentiale = element exponentiale { CommonAtt, DefEncAtt, empty}
imaginaryi = element imaginaryi { CommonAtt, DefEncAtt, empty}
notanumber = element notanumber { CommonAtt, DefEncAtt, empty}
true = element true { CommonAtt, DefEncAtt, empty}
false = element false { CommonAtt, DefEncAtt, empty}
pi = element pi { CommonAtt, DefEncAtt, empty}
eulergamma = element eulergamma { CommonAtt, DefEncAtt, empty}
infinity = element infinity { CommonAtt, DefEncAtt, empty}
```

Bibliography

[] OpenMath. web page at http://www.openmath.org. URL: http://www.openmath.org.

[] *Metadata Commons Worldwide*. web page at http://creativecommons.org/learn/technology/metadata. URL: http://creativecommons.org/learn/technology/metadata.

[] *Document Object Model DOM*. web page at http://www.w3.org/DOM/. URL: http://www.w3.org/DOM/.

[] *Root-Zone Whois Information*. http://www.iana.org/cctld/cctld-whois.htm. URL: http://www.iana.org/cctld/cctld-whois.htm.

[] *JavaServer Pages*. web page at http://java.sun.com/products/jsp. URL: http://java.sun.com/products/jsp.

[] *MathPlayer ¡Display MathML in your browser¿*. web page at http://www.dessci.com/en/products/mathplayer. URL: http://www.dessci.com/en/products/mathplayer.

[] *Maxima – A GPL CAS based on DOE-MACSYMA*. web page at http://maxima.sourceforge.net. URL: http://maxima.sourceforge.net.

[] *MBase*. http://mbase.mathweb.org:8000. URL: http://mbase.mathweb.org:8000.

[] *Mizar Mathematical Library*. Web Page at http://www.mizar.org/library. seen May 2008. URL: http://www.mizar.org/library.

[] *MONET – Mathematics on the net*. web page at http://monet.nag.co.uk. URL: http://monet.nag.co.uk.

[] *The Mozart Programming System*. http://www.mozart-oz.org. URL: http://www.mozart-oz.org.

[] OpenMath *Content Dictionaries*. web page at http://www.openmath.org/cd/. seen June2008. URL: http://www.openmath.org/cd/.

[] *The OMDoc Subversion Repository*. Repository at https://svn.omdoc.org/repos/omdoc. URL: https://svn.omdoc.org/repos/omdoc.

[] *The OMDoc Wiki*. http://www.mathweb.org/omdoc/wiki/. URL: http://www.mathweb.org/omdoc/wiki/.

[] *ROML, The RIACA* OpenMath *Library*. web page at http://crystal.win.tue.nl/download/. URL: http://crystal.win.tue.nl/download/.

[] *WIRIS CAS*. web page at http://www.wiris.com/overview/products/wiris-cas.html. URL: http://www.wiris.com/overview/products/wiris-cas.html.

[] *XML Schema*. Web page at http://www.w3.org/XML/Schema. URL: http://www.w3.org/XML/Schema.

[] *The Yacas computer algebra system*. web page at http://yacas.sourceforge.net. URL: http://yacas.sourceforge.net/.

[03] *MARC code list for Relators, Sources, Description Conventions*. 2003. URL: http://www.loc.gov/marc/relators.

[05a] *Bugzilla*. web page at http://www.bugzilla.org. seen 2005. URL: http://www.bugzilla.org.

[05b] *DeskZilla*. web page at http://www.deskzilla.com. seen 2005. URL: http://www.deskzilla.com.

[06] *Mizar Language*. web page at http://mizar.org/language. seen III 2006. URL: http://mizar.org/language.

[88] *IEEE POSIX*. ISO/IEC 9945. 1988.

[96] *The QED Project*. http://www-unix.mcs.anl.gov/qed/. 1996. URL: http://www-unix.mcs.anl.gov/qed/.

[ABD03] Andrea Asperti, Bruno Buchberger, and James Harold Davenport, eds. *Mathematical Knowledge Management, MKM'03*. LNCS 2594. Springer Verlag, 2003.

[Abn96] S. Abney. *Partial parsing via finite-state cascades*. http://citeseer.ifi.unizh.ch/abney96partial.html. 1996. URL: http://citeseer.ifi.unizh.ch/abney96partial.html.

[ABT04] Andrea Asperti, Grzegorz Bancerek, and Andrej Trybulec, eds. *Mathematical Knowledge Management, MKM'04*. LNAI 3119. Springer Verlag, 2004.

[Adi+08] Ben Adida et al. *RDFa in XHTML: Syntax and Processing*. W3C Recommendation. World Wide Web Consortium (W3C), Oct. 2008. URL: http://www.w3.org/TR/2008/REC-rdfa-syntax-20081014/.

[AF06] Serge Autexier and Armin Fiedler. "Textbook Proofs Meet Formal Logic – The Problem of Underspecification and Granularity". In: *Mathematical Knowledge Management, MKM'05*. Ed. by Michael Kohlhase. LNAI 3863. Springer Verlag, 2006.

[AH05] Serge Autexier and Dieter Hutter. "Formal Software Development in Maya". In: *Festschrift in Honor of J. Siekmann*. Ed. by Dieter Hutter and Werner Stephan. LNAI 2605. Springer, Feb. 2005.

[AK02] Andrea Asperti and Michael Kohlhase. "MathML in the MoWGLI Project". In: *Second International Conference on MathML and Technologies for Math on the Web*. Chicago, USA, 2002. URL: http://www.mathmlconference.org/2002/presentations/asperti/.

[AKS03] Andrea Asperti, Michael Kohlhase, and Claudio Sacerdoti Coen. *Prototype n. D2.b Document Type Descriptors: OMDoc Proofs*. MoWGLI Deliverable. The MoWGLI Project, 2003.

[Alt+10] *Modularization of XHTML*. W3C Recommendation. World Wide Web Consortium (W3C), 2010. URL: http://www.w3.org/TR/xhtml-modularization.

[AM02] Serge Autexier and Till Mossakowski. "Integrating HOLCASL into the Development Graph Manager MAYA". In: *Frontiers of Combinning Systems (FROCOS'02)*. Ed. by Alessandro Armando. LNAI 2309. Springer Verlag, 2002, pp. 2–17.

[And+96] Peter B. Andrews et al. "TPS: A Theorem-Proving System for Classical Type Theory". In: *Journal of Automated Reasoning* 16 (1996), pp. 321–353.

[And02] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. second. Kluwer Academic Publishers, 2002.

[Asp+01] Andrea Asperti et al. "HELM and the Semantic Math-Web". In: *Theorem Proving in Higher Order Logics: TPHOLs'01*. Ed. by Richard. J. Boulton and Paul B. Jackson. LNCS 2152. Springer Verlag, 2001, pp. 59–74.

[Aus+03a] Ron Ausbrooks et al. *Mathematical Markup Language (MathML) Version 2.0 (second edition)*. Tech. rep. World Wide Web Consortium (W3C), 2003. URL: `http://www.w3.org/TR/MathML2`.

[Aus+03b] Ron Ausbrooks et al. *Mathematical Markup Language (MathML) Version 2.0 (second edition)*. W3C Recommendation. World Wide Web Consortium (W3C), 2003. URL: `http://www.w3.org/TR/MathML2`.

[Aut+00a] Serge Autexier et al. "Towards an Evolutionary Formal Software-Development Using CASL". In: *Proceedings Workshop on Algebraic Development Techniques, WADT-99*. Ed. by C. Choppy and D. Bert. LNCS 1827. Springer Verlag, 2000, pp. 73–88.

[Aut+00b] S. Autexier et al. "VSE: Formal methods meet industrial needs". In: *International Journal on Software Tools for Technology Transfer, Special issue on Mechanized Theorem Proving for Technology* 3.1 (Sept. 2000).

[Aut+03] Serge Autexier et al. "Assertion Level Proof Representation with Underspecification". In: *Proceedings of MKM Symposium*. Ed. by Fairouz Kamareddine. Heriot-Watt, Edinburgh, Nov. 2003.

[Aut+06] Serge Autexier et al. "Integrating Proof Assistants as Reasoning and Verification Tools into a Scientific WYSIWYG Editor". In: *Proceedings of UITP'05*. ENTCS (2006). Ed. by David Aspinall and Christoph Lüth.

[Aut+99] S. Autexier et al. "System description: INKA 5.0 – a logical voyager". In: *16^{th} International Conference on Automated Deduction, CADE-16*. Ed. by H. Ganzinger. LNAI 1732. Trento: Springer, 1999.

[Aut05] Serge Autexier. "The CORE Calculus". In: *Proceedings of the 20^{th} International Conference on Automated Deduction (CADE-20)*. Ed. by Robert Nieuwenhuis. LNAI 3632. Tallinn, Estonia: Springer, July 2005.

[Bar80] Hendrik P. Barendregt. *The Lambda-Calculus: Its Syntax and Semantics*. North-Holland, 1980.

[Bau99] Judith Baur. "Syntax und Semantik mathematischer Texte – ein Prototyp". MA thesis. SaarbrückenGermany: Fachrichtung Computerlinguistik, Universität des Saarlandes, 1999.

[BB01] P. Baumgartner and A. Blohm. "Automated deduction techniques for the management of personalized documents". In: *Electronic Proceedings of the First International Workshop on Mathematical Knowledge Management: MKM'2001*. Ed. by Bruno Buchberger and Olga Caprotti. 2001. URL: `http://www.risc.uni-linz.ac.at/institute/conferences/MKM2001/Proceedings/`.

[BC01a] Henk Barendregt and Arjeh M. Cohen. "Electronic communication of mathematics and the interaction of computer algebra systems and proof assistants". In: *Journal of Symbolic Computation* 32 (2001), pp. 3–22.

[BC01b] Bruno Buchberger and Olga Caprotti, eds. *Electronic Proceedings of the First International Workshop on Mathematical Knowledge Management: MKM'2001*. 2001. URL: `http://www.risc.uni-linz.ac.at/institute/conferences/MKM2001/Proceedings/`.

[Ben+97] Christoph Benzmüller et al. "ΩMEGA: Towards a mathematical assistant". In: *Proceedings of the 14^{th} Conference on Automated Deduction*. Ed. by William McCune. LNAI 1249. Townsville, Australia: Springer Verlag, 1997, pp. 252–255. URL: `http://kwarc.info/kohlhase/papers/Omega97-CADE.pdf`.

[Ber91] Paul Bernays. *Axiomatic Set Theory*. Dover Publications, 1991.

[Ber98] Tim Berners-Lee. *The Semantic Web*. W3C Architecture Note. 1998. URL: `http://www.w3.org/DesignIssues/Semantic.html`.

[BF06] Jon Borwein and William M. Farmer, eds. *Mathematical Knowledge Management (MKM)*. LNAI 4108. Springer Verlag, 2006.

[BFM98] Tim Berners-Lee, Roy T. Fielding, and Larry. Masinter. *Uniform Resource Identifiers (URI), Generic Syntax*. RFC 2717. Internet Engineering Task Force (IETF), 1998. URL: `http://www.ietf.org/rfc/rfc2717.txt`.

[BHL99] *Namespaces in XML*. W3C Recommendation. World Wide Web Consortium (W3C), 1999. URL: `http://www.w3.org/TR/REC-xml-names`.

[Blo56] B. S. Bloom, ed. *Taxonomy of educational objectives: The classification of educational goals: Handbook I, cognitive domain*. New York, Toronto: Longmans, Green, 1956.

[BM04] Paul V. Biron and Ashok Malhotra. *XML Schema Part 2: Datatypes Second Edition*. Tech. rep. World Wide Web Consortium (W3C), Oct. 28, 2004. URL: `http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/`.

[Bos+98] *Cascading Style Sheets, level 2; CSS2 Specification*. W3C Recommendation. World Wide Web Consortium (W3C), 1998. URL: `http://www.w3.org/TR/1998/REC-CSS2-19980512`.

[Bou74] Nicolas Bourbaki. *Algebra I*. Elements of Mathematics. Springer Verlag, 1974.

[Bra+02] R. Bradford et al. "Reasoning About the Elementary Functions of Complex Analysis." In: *Annals of Mathematics and Artificial Intelligence* 36 (2002), pp. 303–318.

[Bra+04] Tim Bray et al. *Extensible Markup Language (XML) 1.1*. W3C Recommendation REC-xml11-20040204. World Wide Web Consortium (W3C), 2004. URL: `http://www.w3.org/TR/2004/REC-xml11-20040204/`.

[Bru80] Nicolaas Govert de Bruijn. "A Survey of the Project AUTOMATH". In: *To H.B. Curry: Essays in Combinator Logic, Lambda Calculus and Formalisms*. Ed. by R. Hindley and J. Seldin. Academic Press, 1980, pp. 579–606.

[Bru94] Nicolaas Govert de Bruijn. "The Mathematical Vernacular, A Language for Mathematics with Typed Sets". In: *Selected Papers on Automath*. Ed. by R. P Nederpelt, J. H. Geuvers, and R. C. de Vrijer. Vol. 133. Studies in Logic and the Foundations of Mathematics. Elsevier, 1994, pp. 865–935.

[Bus+04] Stephen Buswell et al. *The Open Math Standard, Version 2.0*. Tech. rep. The OpenMath Society, 2004. URL: `http://www.openmath.org/standard/om20`.

[Bus+99] Stephen Buswell et al. *Mathematical Markup Language (MathML) 1.01 Specification*. W3C Recommendation. World Wide Web Consortium (W3C), 1999. URL: `http://www.w3.org/TR/REC-MathML`.

[Cap+00] Olga Caprotti et al. "Using OPENMATH Servers for Distributing Mathematical Computations". In: *ATCM 2000: Proceedings of the Fifth Asian Technology Conference in Mathematics*. Ed. by Wei Chi Yang, Sung-Chi Chu, and Jen-Chung Chuan. Chiang-Mai, Thailand: ATCM, Inc., 2000, pp. 325–336.

[CCR00] Olga Caprotti, Arjeh M. Cohen, and Manfred Riem. "Java Phrasebooks for Computer Algebra and Automated Deduction". In: *Bulletin of the ACM Special Interest Group on Symbolic and Automated Mathematics (SIGSAM)* 34.2 (2000), pp. 43–48.

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

[Cha+92] Bruce W. Char et al. *First leaves: a tutorial introduction to Maple V*. Berlin: Springer Verlag, 1992.

[Cla+03] Edmund Clarke et al. "System Description: Analytica 2". In: *11^{th} Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus 2003)*. Ed. by Thérèse Hardin and Renaud Rioboo. Rome, Italy, Sept. 2003, pp. 69–74. URL: `http://kwarc.info/kohlhase/papers/calculemus03.pdf`.

[Cla05] James Clark. *nXML mode*. web page at `http://www.thaiopensource.com/nxml-mode/`. seen 2005. URL: `http://www.thaiopensource.com/nxml-mode/`.

[Cla97] James Clark. *Comparison of SGML and XML*. World Wide Web Consortium Note. 1997. URL: `http://www.w3.org/TR/NOTE-sgml-xml.html`.

[Cla99a] *Associating Style Sheets with XML Documents Version 1.0*. W3C Recommendation. World Wide Web Consortium (W3C), 1999. URL: `http://www.w3.org/TR/xml-stylesheet`.

[Cla99b] *XSL Transformations (XSLT) Version 1.0*. W3C Recommendation. World Wide Web Consortium (W3C), 1999. URL: `http://www.w3.org/TR/xslt`.

[CM98] A. M. Cohen and L. Meertens. "The ACELA project: Aims and Plans". In: *Computer-Human interaction in Symbolic Computation*. Ed. by N. Kajler. Texts and Monographs in Symbolic Computation. Springer Verlag, 1998, pp. 7–23.

[Com] Userland Com. *XML Remote Procedure Call Specification*. web page at `http://www.xmlrpc.com/`. URL: `http://www.xmlrpc.com/`.

[Con+04] R. Conejo et al. "SIETTE: A Web-Based Tool for Adaptive Teaching". In: *International Journal of Artificial Intelligence in Education (IJAIED 2004)* 14 (2004), pp. 29–61.

[Con+86] Robert L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System*. Englewood Cliffs, NJUSA: Prentice-Hall, 1986.

[Con01] IMS Global Learning Consortium. *Learnig Resource Metadata Specification*. 2001. URL: `http://www.imsglobal.org/metadata/`.

[Cor] Microsoft Corp. *Microsoft Internet Explorer*. web page at `http://www.microsoft.com/windows/ie/`. URL: `http://www.microsoft.com/windows/ie/`.

[CT04] *XML Information Set (Second Edition)*. W3C Recommendation. World Wide Web Consortium (W3C), Feb. 4, 2004. URL: `http://www.w3.org/TR/2004/REC-xml-infoset-20040204`.

[Dah01] Ingo Dahn. "Slicing Book Technology – Providing Online Support for Textbooks". In: *The 20^{th} ICDE World Conference on Open Learning and Distance Education*. 2001.

[DCM03a] The DCMI Usage Board. *DCMI Metadata Terms*. DCMI Recommendation. Dublin Core Metadata Initiative, 2003. URL: `http://dublincore.org/documents/dcmi-terms/`.

[DCM03b] The DCMI Usage Board. *DCMI Type Vocabulary*. DCMI Recommendation. Dublin Core Metadata Initiative, 2003. URL: `http://dublincore.org/documents/dcmi-type-vocabulary/`.

[DeR+01] Steve DeRose et al. *XML Linking Language (XLink Version 1.0)*. W3C Recommendation. World Wide Web Consortium (W3C), 2001. URL: `http://www.w3.org/TR/2000/REC-xlink-20010627/`.

[Duc98] Denys Duchier. "The NeGra tree bank". Private communication. 1998.

[DW05] Mark Davis and Ken Whistler. *Unicode Collation Algorithm*. Unicode Technical Standard #10. 2005. URL: `http://www.unicode.org/reports/tr10/`.

[Far93] William M. Farmer. "Theory Interpretation in Simple Type Theory". In: *HOA'93, an International Workshop on Higher-order Algebra, Logic and Term Rewriting*. LNCS 816. Amsterdam, The Netherlands: Springer Verlag, 1993.

[FB96] N. Freed and N. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. RFC 2046: `http://www.faqs.org/rfcs/rfc2046.html`. 1996. URL: `http://www.faqs.org/rfcs/rfc2046.html`.

[FGT93] William M. Farmer, Joshua D. Guttman, and F. Javier Thayer. "IMPS: An Interactive Mathematical Proof System". In: *Journal of Automated Reasoning* 11.2 (Oct. 1993), pp. 213–248.

[FH01] Armin Fiedler and Helmut Horacek. "Argumentation in Explanations to Logical Problems". In: *Computational Science — ICCS 2001*. Ed. by Vassil N. Alexandrov et al. LNCS 2074. San Francisco, CAUSA: Springer Verlag, 2001, pp. 969–978.

[FH97] Amy P. Felty and Douglas J. Howe. "Hybrid Interactive Theorem Proving using NuPRL and HOL". In: *Proceedings of the 14th Conference on Automated Deduction*. Ed. by William McCune. LNAI 1249. Townsville, Australia: Springer Verlag, 1997, pp. 351–365.

[Fie01a] Armin Fiedler. "Dialog-driven Adaptation of Explanations of Proofs". In: *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*. Ed. by Bernhard Nebel. Seattle, WAUSA: Morgan Kaufmann, 2001, pp. 1295–1300.

[Fie01b] Armin Fiedler. "User-adaptive Proof Explanation". PhD Thesis. Saarbrücken, Germany: Naturwissenschaftlich-Technische Fakultät I, Universität des Saarlandes, 2001.

[Fie97] Armin Fiedler. "Towards a Proof Explainer". In: *Proceedings of the First International Workshop on Proof Transformation and Presentation*. Ed. by J. Siekmann, F. Pfenning, and X. Huang. Schloss DagstuhlGermany, 1997, pp. 53–54.

[Fie99] Armin Fiedler. "Using a Cognitive Architecture to Plan Dialogs for the Adaptive Explanation of Proofs". In: *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*. Ed. by Thomas Dean. Stockholm: Morgan Kaufmann, 1999, pp. 358–363.

[FK99] Andreas Franke and Michael Kohlhase. "System Description: MathWeb, an Agent-Based Communication Layer for Distributed Automated Theorem Proving". In: *Automated Deduction — CADE-16*. Ed. by Harald Ganzinger. LNAI 1632. Springer Verlag, 1999, pp. 217–221. URL: `http://kwarc.info/kohlhase/papers/cade99.pdf`.

[Fra+99] Andreas Franke et al. "Agent-Oriented Integration of Distributed Mathematical Services". In: *Journal of Universal Computer Science* 5 (1999), pp. 156–187. URL: `http://kwarc.info/kohlhase/papers/jucs.pdf`.

[Fre91] Free Software Foundation. *GNU General Public License*. `http://www.gnu.org/copyleft/gpl.html`. Software License. 1991. URL: `http://www.gnu.org/copyleft/gpl.html`.

[Fre99] Free Software Foundation. *GNU Lesser General Public License*. `http://www.gnu.org/copyleft/lesser.html`. Software License. 1999. URL: `http://www.gnu.org/copyleft/lesser.html`.

[G S95] G. Smolka. "The Oz Programming Model". In: *Computer Science Today*. Ed. by Jan van Leeuwen. LNCS 1000. Berlin: Springer-Verlag, 1995, pp. 324–343.

[GB92] J. A. Goguen and R. M. Burstall. In: *Journal of the Association for Computing Machinery* 39 (1992). Predecessor in: LNCS 164, 221–256, 1984., pp. 95–146.

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

[GG03] 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. Nov. 2003. URL: `http://www.ags.uni-sb.de/~alberto/publications/authoring.ps`.

[GGM05] Georgi Goguadze, Alberto González Palomo, and Erica Melis. "Interactivity of Exercises in ActiveMath". In: *International Conference on Computers in Education (ICCE)*. Singapore, 2005.

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

[Gog+03] George Goguadze et al. "Problems and Solutions for Markup for Mathematical Examples and Exercises". In: *Mathematical Knowledge Management, MKM'03*. Ed. by Andrea Asperti, Bruno Buchberger, and James Harold Davenport. LNCS 2594. Springer Verlag, 2003, pp. 80–93.

[Gog+04] George Goguadze et al. *LeActiveMath Structure and Metadata Model*. Deliverable D6. LeActiveMath Consortium, 2004. URL: `http://www.activemath.org/pubs/LeAM-D6.pdf`.

[Gona] Alberto González Palomo. *Algebra*. `http://www.mathweb.org/algebra/index.en.html`. URL: `http://www.mathweb.org/algebra/index.en.html`.

[Gonb] Alberto González Palomo. *QMath History*. `http://www.matracas.org/qmath/history.html`. URL: `http://www.matracas.org/qmath/history.html`.

[GR02] J. Goguen and G. Rosu. "Institution morphisms". In: *Formal aspects of computing* 13 (2002), pp. 274–307.

[Gra96] Peter Graf. *Term Indexing*. LNCS 1053. Springer Verlag, 1996.

[Gro+03] Paul Grosso et al. *W3C XPointer Framework*. W3C Recommendation. World Wide Web Consortium (W3C), Mar. 25, 2003. URL: `http://www.w3.org/TR/2003/REC-xptr-framework-20030325/`.

[Gro+07] Tudor Groza et al. "SALT – Semantically Annotated LaTeX for Scientific Publications". In: *The Semantic Web: Research and Applications*. 4th European Semantic Web Conference (ESWC). (Innsbruck, Austria, June 3–7, 2007). Ed. by Enrico Franconi, Michael Kifer, and Wolfgang May. LNCS 4519. Springer Verlag, 2007, pp. 518–532. ISBN: 9783540726661.

[Gro99] The Open eBook Group. *Open eBook[tm] Publication Structure 1.0*. Draft Recommendation. The OpenEBook Initiative, 1999. URL: `http://www.openEbook.org`.

[Gud+03] Martin Gudgin et al. *SOAP 1.2 Part 2: Adjuncts*. W3C Recommendation. 2003. URL: `http://www.w3.org/TR/2003/REC-soap12-part2-20030624`.

[Har03] Eliotte Rusty Harold. "Effective XML". In: Addison Wesley, 2003. Chap. 15.

[HF96] Xiaorong Huang and Armin Fiedler. "Presenting Machine-Found Proofs". In: *Proceedings of the 13th Conference on Automated Deduction*. Ed. by M. A. McRobbie and J. K. Slaney. LNAI 1104. New Brunswick, NJ, USA: Springer Verlag, 1996, pp. 221–225.

[HF97] Xiaorong Huang and Armin Fiedler. "Proof Verbalization in *PROVERB*". In: *Proceedings of the First International Workshop on Proof Transformation and Presentation*. Ed. by J. Siekmann, F. Pfenning, and X. Huang. Schloss DagstuhlGermany, 1997, pp. 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.

[Hoe01] Joris Van der Hoeven. "GNU TeXMacs: A free, structured, wysiwyg and technical text editor". In: *Cahiers GUTenberg* (May 2001), pp. 39–40.

[HS96] Dieter Hutter and Claus Sengler. "INKA – The Next Generation". In: *Proceedings of the 13th Conference on Automated Deduction*. Ed. by M. A. McRobbie and J. K. Slaney. LNAI 1104. New Brunswick, NJ, USA: Springer Verlag, 1996, pp. 288–292.

[Hua96] Xiaorong Huang. *Human Oriented Proof Presentation: A Reconstructive Approach*. DISKI 112. Sankt Augustin, Germany: Infix, 1996.

[Hut00] Dieter Hutter. "Management of Change in Verification Systems". In: *Proceedings 15th IEEE International Conference on Automated Software Engineering, ASE-2000*. IEEE Computer Society, 2000, pp. 23–34.

[IEE02] IEEE Learning Technology Standards Committee. *Standard for Learning Object Metadata*. Tech. rep. 1484.12.1. IEEE, 2002.

[Inc03] Unicode Inc., ed. *The Unicode Standard, Version 4.0*. Addison-Wesley, 2003.

[JFF02] Dean Jackson, Jon Ferraiolo, and Jun Fujisawa. *Scalable Vector Graphics (SVG) 1.1 Specification*. W3C Candidate Recommendation. World Wide Web Consortium (W3C), Apr. 2002. URL: http://www.w3.org/TR/2002/CR-SVG11-20020430.

[KA03] Michael Kohlhase and Romeo Anghelache. "Towards Collaborative Content Management And Version Control For Structured Mathematical Knowledge". In: *Mathematical Knowledge Management, MKM'03*. Ed. by Andrea Asperti, Bruno Buchberger, and James Harold Davenport. LNCS 2594. Springer Verlag, 2003, pp. 147–161. URL: http://kwarc.info/kohlhase/papers/mkm03.pdf.

[Kay] Michael Kay. *SAXON, The XSLT and XQuery Processor*. Web page at saxon.sf.net. URL: saxon.sf.net.

[KD03a] Michael Kohlhase and Stan Devitt. *Bound Variables in MathML*. W3C Working Group Note. 2003. URL: http://www.w3.org/TR/mathml-bvar/.

[KD03b] Michael Kohlhase and Stan Devitt. *Structured Types in MathML 2.0*. W3C Note. 2003. URL: http://www.w3.org/TR/mathml-types/.

[KF01] Michael Kohlhase and Andreas Franke. "MBase: Representing Knowledge and Context for the Integration of Mathematical Software Systems". In: *Journal of Symbolic Computation; Special Issue on the Integration of Computer Algebra and Deduction Systems* 32.4 (2001), pp. 365–402. DOI: 10.1006/jsco.2000.0468. URL: http://kwarc.info/kohlhase/papers/jsc.pdf.

[KK04] Andrea Kohlhase and Michael Kohlhase. "CPoint: Dissolving the Author's Dilemma". In: *Mathematical Knowledge Management, MKM'04*. Ed. by Andrea Asperti, Grzegorz Bancerek, and Andrej Trybulec. LNAI 3119. Springer Verlag, 2004, pp. 175–189. URL: http://kwarc.info/kohlhase/papers/mkm04.pdf.

[KK06] Andrea Kohlhase and Michael Kohlhase. "An Exploration in the Space of Mathematical Knowledge". In: *Mathematical Knowledge Management, MKM'05*. Ed. by Michael Kohlhase. LNAI 3863. Springer Verlag, 2006, pp. 17–32. URL: http://kwarc.info/kohlhase/papers/mkm05.pdf.

[Kli+04] E. Klieme et al. *The development of national educational standards – an expertise*. Tech. rep. Bundesministerium für Bildung und Forschung / German Federal Ministry of Education and Research, 2004.

[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. URL: https://svn.omdoc.org/repos/codeml/doc/spec/codeml.pdf.

[Kohb] Michael Kohlhase. *OMDoc Mailing Lists*. http://omdoc.org/resources/mailing-lists.html. seen May 2008. URL: http://omdoc.org/resources/mailing-lists.html.

[Kohc] Michael Kohlhase. *The OMDoc Document Type Definition*. http://omdoc.org/dtd/omdoc.dtd. URL: http://omdoc.org/dtd/omdoc.dtd.

[Kohd] Michael Kohlhase. *The OMDoc RelaxNG Schema*. http://omdoc.org/rnc/omdoc.rnc. URL: http://omdoc.org/rnc/omdoc.rnc.

[Kohe] Michael Kohlhase. *The OMDoc XML Schema*. `http://omdoc.org/rnc/omdoc.xsd`. URL: `http://omdoc.org/rnc/omdoc.xsd`.

[Kohf] Michael Kohlhase. *XSL Style Sheets for OMDoc*. `http://omdoc.org/xsl/`. URL: `http://omdoc.org/xsl/`.

[Koh05] Michael Kohlhase. *Inference Rules*. OMDoc Content Dictionary at `https://svn.omdoc.org/repos/omdoc/trunk/examples/logics/inference-rules.omdoc`. seen Jan 2005. URL: `https://svn.omdoc.org/repos/omdoc/trunk/examples/logics/inference-rules.omdoc`.

[Koh06a] Andrea Kohlhase. "What if PowerPoint became emPowerPoint (through CPoint)?" In: *Society for Information Technology and Teacher Education, 17th International Conference SITE 2006*. Ed. by Caroline M. Crawford. Orlando (USA), 2006-03-20/24. SITE. AACE, 2006, pp. 2934–2939.

[Koh06b] Michael Kohlhase, ed. *Mathematical Knowledge Management, MKM'05*. LNAI 3863. Springer Verlag, 2006.

[Koh06c] Michael Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2]*. LNAI 4180. Springer Verlag, Aug. 2006. URL: `http://omdoc.org/pubs/omdoc1.2.pdf`.

[KR93] Hans Kamp and Uwe Reyle. *From Discourse to Logic: Introduction to Model-Theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Dordrecht: Kluwer, 1993.

[Kri+03] B. Krieg-Brückner et al. "MultiMedia Instruction in Safe and Secure Systems". In: *Recent Trends in Algebraic Development Techniques*. LNCS 2755. Springer Verlag, 2003, pp. 82–117.

[Kri+04a] Bernd Krieg-Brückner et al. "Semantic Interrelation of Documents via an Ontology". In: *DeLFI 2004*. Ed. by G. Engels and S. Seehusen. Vol. P-52. LNI. Springer-Verlag, 2004, pp. 271–282. ISBN: 3885793814. URL: `http://www.mmiss.de/papers/semantic_interrelation.pdf`.

[Kri+04b] B. Krieg-Brückner et al. *MultiMedia Instruction in Safe and Secure Systems*. Abschlussbericht. BMBF project 01NM070, 2001-2004. Universität Bremen, 2004.

[Lag+02] Carl Lagoze et al., eds. *The Open Archives Initiative Protocol for Metadata Harvesting*. June 2002. URL: `http://www.openarchives.org/OAI/openarchivesprotocol.html`.

[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. ISBN: 020171499X.

[Len04] Richard Lennox. *Development of an RDF/XML based data model for bibliographic data*. Dissertation for Bachelor of Science in Computer Science. http://richardlennox.net/dissertation.pdf 2004.

[Lib04] P. Libbrecht. "Authoring Web Content in ActiveMath: From Developer Tools and Further". In: *Proceedings of the Second International Workshop on Authoring Adaptive and Adaptable Educational Hypermedia, AH-2004: Workshop Proceedings, Part II, CS-Report 04-19*. Ed. by Alexandra Christea and Franca Garzotto. Technische Universiteit Eindhoven, 2004, pp. 455–460.

[LK09] Christoph Lange and Michael Kohlhase. "A Mathematical Approach to Ontology Authoring and Documentation". In: *MKM/Calculemus Proceedings*. Ed. by Jacques Carette et al. LNAI 5625. Springer Verlag, July 2009, pp. 389–404. ISBN: 9783642026133. URL: `http://kwarc.info/kohlhase/papers/mkm09-omdoc4onto.pdf`.

[Lom05] Cyprien Lomas. *7 things you should know about social bookmarking*. `http://www.educause.edu/ir/library/pdf/ELI7001.pdf`. Seen March 2006. EDUCAUSE Learning Initiative, 2005. URL: `http://www.educause.edu/ir/library/pdf/ELI7001.pdf`.

[LS99] Ora Lassila and Ralph R. Swick. *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation. World Wide Web Consortium (W3C), 1999. URL: `http://www.w3.org/TR/1999/REC-rdf-syntax`.

[M+04] Robert Meersman, Zahir Tari, Angelo Corsaro, et al., eds. *On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops*. LNCS 3292. Springer Verlag, 2004.

[MAH06] Till Mossakowski, Serge Autexier, and Dieter Hutter. "Development Graphs – Proof Management for Structured Specifications". In: *Journal of Logic and Algebraic Programming* 67.1–2 (2006), pp. 114–145.

[Man+06] Shahid Manzoor et al. "Authoring Presentation for OpenMath". In: *Mathematical Knowledge Management, MKM'05*. Ed. by Michael Kohlhase. LNAI 3863. Springer Verlag, 2006, pp. 33–48.

[McC97] William McCune, ed. *Proceedings of the 14^{th} Conference on Automated Deduction*. LNAI 1249. Townsville, Australia: Springer Verlag, 1997.

[Mei00] Andreas Meier. "System Description: Tramp: Transformation of Machine-Found Proofs into ND-Proofs at the Assertion Level". In: *Automated Deduction – CADE-17*. Ed. by David McAllester. LNAI 1831. Springer Verlag, 2000, pp. 460–464.

[Mel+01] E. Melis et al. "The ActiveMath Learning Environment". In: *Artificial Intelligence and Education* 12.4 (2001).

[Mel+03] Erica Melis et al. "Knowledge Representation and Management in ActiveMath". In: *Annals of Mathematics and Artificial Intelligence* 38.1–3 (2003), pp. 47–64.

[Mel+06] Erica Melis et al. "Semantic-aware components and services of ActiveMath". In: *British Journal of Educational Technology* 37.3 (May 2006), pp. 405–423.

[Mes89] J. Meseguer. "General Logics". In: *Logic Colloquium 87*. North Holland, 1989, pp. 275–329.

[MG04a] M. Mavrikis and A. González Palomo. "Mathematical, Interactive Exercise Generation from Static Documents". In: *Electronic Notes in Computer Science* 93 (2004), pp. 183–201.

[MG04b] E. Melis and G. Goguadze. "Towards Adaptive Generation of Faded Examples". In: *International Conference on Intelligent Tutoring Systems*. Ed. by J. Lester, R. Vicari, and F. Paraguacu. LNCS 3220. Springer-Verlag, 2004, pp. 762–771. ISBN: 3540229485. URL: `Melisetal-FadedEx-ITS04-2004.pdf`.

[Mil] Bruce Miller. *LaTeXML: A LaTeX to XML Converter*. Web Manual at `http://dlmf.nist.gov/LaTeXML/`. seen September 2011. URL: `http://dlmf.nist.gov/LaTeXML/`.

[Mit03] Nilo Mitra. *SOAP 1.2 Part 0: Primer*. W3C Recommendation. 2003. URL: `http://www.w3.org/TR/2003/REC-soap12-part0-20030624`.

[MKH05] E. Melis, P. Kärger, and M. Homik. "Interactive Concept Mapping in ActiveMath (iCMap)". In: *Delfi 2005: 3. Deutsche eLearning Fachtagung Informatik*. Ed. by Jörg M. Haake, Ulrich Lucke, and Djamshid Tavangarian. Vol. 66. LNI. Gesellschaft für Informatik e.V. (GI). 2005, pp. 247–258. ISBN: 3885793954.

[Mos+05] Till Mossakowski et al. "What is a Logic?" In: *Logica Universalis*. Ed. by Jean-Yves Beziau. Birkhäuser, 2005, pp. 113–133. URL: `http://www.tzi.de/~till/papers/nel05.pdf`.

[Mos02] Till Mossakowski. "Heterogeneous development graphs and heterogeneous borrowing". In: *Foundations of Software Science and Computation Structures (FOSSACS02)*. Ed. by Mogens Nielsen and Uffe Engberg. LNCS 2303. Springer Verlag, 2002, pp. 310–325.

[Mos04a] T. Mossakowski. *HetCASL – Heterogeneous Specification. Language Summary*. 2004. URL: `http://www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/CoFI/HetCASL/HetCASL-Summary.pdf`.

[Mos04b] P. D. Mosses, ed. CASL *Reference Manual*. LNCS 2960 (IFIP Series). Springer Verlag, 2004.

[Mos05] T. Mossakowski. "Heterogeneous specification and the heterogeneous tool set". Habilitation thesis, University of Bremen. 2005.

[MS96] M. A. McRobbie and J. K. Slaney, eds. *Proceedings of the 13th Conference on Automated Deduction*. LNAI 1104. New Brunswick, NJ, USA: Springer Verlag, 1996.

[MSK01] M. Murata, S. St. Laurent, and D. Kohn. *XML Media Types*. RFC 3023. Jan. 2001. URL: `ftp://ftp.isi.edu/in-notes/rfc3023.txt`.

[Mül05] Normen Müller. "OMDoc-Repräsentation von Programmen und Beweisen in VeriFun". MA thesis. Programmiermethodik, Technische Universität Darmstadt, 2005. URL: `http://kwarc.info/nmueller/papers/dt.pdf`.

[Mur+] Peter Murray-Rust et al. *Chemical Markup Language (CML)*. `http://cml.sourceforge.net/`. seen January 2007. URL: `http://cml.sourceforge.net/`.

[MVW05] Jonathan Marsh, Daniel Veillard, and Norman Walsh. `xml:id` *Version 1.0*. Tech. rep. World Wide Web Consortium (W3C), Sept. 9, 2005. URL: `http://www.w3.org/TR/2005/REC-xml-id-20050909/`.

[NS81] Alan Newell and Herbert A. Simon. "Computer Science as empirical inquiry: Symbols and search". In: *Communications of the Association for Computing Machinery* 19 (1981), pp. 113–126.

[Odl95] A. M. Odlyzko. "Tragic loss or good riddance? The impending demise of traditional scholarly journals". In: *International Journal of Human-Computer Studies* 42 (1995), pp. 71–122.

[OMDoc] Michael Kohlhase. OMDOC*: An open markup format for mathematical documents (latest released version)*. Specification, `http://www.omdoc.org/pubs/spec.pdf`. URL: `http://www.omdoc.org/pubs/spec.pdf`.

[Org] The Mozilla Organization. *Mozilla*. web page at `http://www.mozilla.org`. URL: `http://www.mozilla.org`.

[ORS92] S. Owre, J. M. Rushby, and N. Shankar. "PVS: A Prototype Verification System". In: *Proceedings of the 11th Conference on Automated Deduction*. Ed. by D. Kapur. LNCS 607. Saratoga Springs, NY, USA: Springer Verlag, 1992, pp. 748–752.

[Pau94] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*. LNCS 828. Springer Verlag, 1994.

[PB04] F. Piroi and B. Buchberger. *An environment for building mathematical knowledge libraries*. `http://citeseer.ifi.unizh.ch/piroi04environment.html`. 2004. URL: `citeseer.ifi.unizh.ch/piroi04environment.html`.

[Pfe01] Frank Pfenning. "Logical Frameworks". In: *Handbook of Automated Reasoning*. Ed. by Alan Robinson and Andrei Voronkov. Vol. I and II. Elsevier Science and MIT Press, 2001.

[Pfe91] Frank Pfenning. "Logic Programming in the LF Logical Framework". In: *Logical Frameworks*. Ed. by Gérard P. Huet and Gordon D. Plotkin. Cambridge University Press, 1991.

[Pie80] John R. Pierce. *An Introduction to Information Theory. Symbols, Signals and Noise.* Dover Publications Inc., 1980.

[Pin+04] Manfred Pinkal et al. "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. URL: `http://www.ags.uni-sb.de/~chris/papers/R30.pdf`.

[PN90] Lawrence C. Paulson and Tobias Nipkow. *Isabelle Tutorial and User's Manual.* Tech. rep. 189. Computer Laboratory, University of Cambridge, Jan. 1990.

[RSG98] Julian D. C. Richardson, Alan Smaill, and Ian M. Green. "System description: Proof planning in higher-order logic with *λClam*". In: *Proceedings of the 15th Conference on Automated Deduction.* Ed. by Claude Kirchner and Hélène Kirchner. LNAI 1421. Springer Verlag, 1998.

[Rud92] Piotr Rudnicki. "An Overview of the MIZAR Project". In: *Proceedings of the 1992 Workshop on Types and Proofs as Programs.* 1992, pp. 311–332.

[Sac06] Claudio Sacerdoti Coen. "Explanation in Natural Language of $\overline{\lambda}\mu\overline{\mu}$-terms". In: *Mathematical Knowledge Management, MKM'05.* Ed. by Michael Kohlhase. LNAI 3863. Springer Verlag, 2006.

[SBA05] Jörg Siekmann, Christoph Benzmüller, and Serge Autexier. "Computer Supported Mathematics with OMEGA". In: *Journal of Applied Logic, special issue on Mathematics Assistance Systems* (Dec. 2005). Ed. by Christoph Benzmüller.

[Sch+06] Sebastian Schaffert et al. "Learning with Semantic Wikis". In: *Proceedings of the 1st Workshop on Semantic Wikis, European Semantic Web Conference.* (Budva, Montenegro, June 12, 2006). Ed. by Max Völkel, Sebastian Schaffert, and Stefan Decker. CEUR Workshop Proceedings 206. Aachen, 2006. URL: `http://www.wastl.net/download/paper/Schaffert06_SemWikiLearning.pdf`.

[Sch04] Klaus Schneider. *Verification of Reactive Systems.* Springer Verlag, 2004.

[Sch06] Sebastian Schaffert. *IkeWiki: A Semantic Wiki for Collaborative Knowledge Management.* Tech. rep. Salzburg Research Forschungsgesellschaft, 2006. URL: `http://www.wastl.net/download/paper/schaffert06_ikewiki.pdf`.

[Sie+00] Jörg Siekmann et al. "Adaptive Course Generation and Presentation". In: *Proceedings of ITS-2000 workshop on Adaptive and Intelligent Web-Based Education Systems.* Ed. by P. Brusilovski and Chrisoph Peylo. Montreal, 2000.

[Sie+99] Jörg Siekmann et al. "LΩUI: Lovely ΩMEGA User Interface". In: *Formal Aspects of Computing* 3.11 (1999), pp. 326–342. URL: `http://kwarc.info/kohlhase/papers/foascomp.pdf`.

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

[SSY94] Geoff Sutcliffe, Christian Suttner, and Theodor Yemenis. "The TPTP Problem Library". In: *Proceedings of the 12th Conference on Automated Deduction.* Ed. by Alan Bundy. LNAI 814. Nancy, France: Springer Verlag, 1994.

[SZS04] G. Sutcliffe, J. Zimmer, and S. Schulz. "TSTP Data-Exchange Formats for Automated Theorem Proving Tools". In: *Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems.* Ed. by W. Zhang and V. Sorge. Frontiers in Artificial Intelligence and Applications 112. IOS Press, 2004, pp. 201–215.

[Tea] Coq Development Team. *The Coq Proof Assistant: Reference Manual.* INRIA. URL: `https://coq.inria.fr/refman/`.

[The] The Apache Software Foundation. *Xalan-Java.* Web page at `http://xml.apache.org/xalan-j`. URL: `http://xml.apache.org/xalan-j`.

[The02] The W3C HTML Working Group. *XHTML 1.0 The Extensible HyperText Markup Language (Second Edition) – A Reformulation of HTML 4 in XML 1.0*. W3C Recommendation. World Wide Web Consortium (W3C), Aug. 1, 2002. URL: http://www.w3.org/TR/2002/REC-xhtml1-20020801.

[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. URL: http://www.cogsci.ed.ac.uk/~richard/rxp.html.

[TS06] Robert Tolksdorf and Elena Paslaru Bontas Simperl. "Towards Wikis as Semantic Hypermedia". In: *International Symposium on Wikis (WikiSym)*. (Odense, Denmark, Aug. 21–23, 2006). Ed. by Dirk Riehle and James Noble. ACM Press. 2006. URL: http://www.wikisym.org/ws2006/proceedings/p79.pdf.

[Ull+04] C. Ullrich et al. "A Flexible and Efficient Presentation-Architecture for Adaptive Hypermedia: Description and Technical Evaluation". In: *Proceedings of the 4th IEEE International Conference on Advanced Learning Technologies (ICALT 2004)*. Ed. by Kinshuk et al. IEEE Computer Society, 2004, pp. 21–25.

[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*. Hiroshima, Japan, 2004, pp. 93–94.

[Ull05] C. Ullrich. "Tutorial Planning: Adapting Course Generation to Today's Needs". In: *Young Researcher Track Proceedings of 12th International Conference on Artificial Intelligence in Education*. Ed. by M. Grandbastian. Amsterdam, 2005, pp. 155–160.

[Vat] Irène Vatton. *Welcome to Amaya*. web page at http://www.w3.org/Amaya. URL: http://www.w3.org/Amaya.

[Veia] Daniel Veillard. *The XML C parser and toolkit of Gnome; libxml*. System Home page at http://xmlsoft.org. URL: http://xmlsoft.org.

[Veib] Daniel Veillard. *The XSLT C library for Gnome: libxslt*. Web page at http://xmlsoft.org/XSLT/. URL: http://xmlsoft.org/XSLT/.

[Vli03] Eric van der Vlist. *Relax NG*. O'Reilly, 2003.

[Völ+] Max Völkel et al. *Semantic Wiki State of The Art Paper – Ontoworld*. Ed. by Semantic Wiki Interest Group/AIFB Institute, Karlsruhe. http://ontoworld.org/index.php/Semantic_Wiki_State_of_The_Art_Paper. draft. seen January 2007. URL: http://ontoworld.org/index.php/Semantic_Wiki_State_of_The_Art_Paper.

[Völ+06] Max Völkel et al. "Semantic Wikipedia". In: *Proceedings of the 15th international World Wide Web conference (WWW)*. (Edinburgh, Scotland). ACM Press, 2006, pp. 585–594. URL: http://www.aifb.uni-karlsruhe.de/WBS/hha/papers/SemanticWikipedia.pdf.

[Wei97] Christoph Weidenbach. "SPASS: Version 0.49". In: *Journal of Automated Reasoning* 18.2 (1997). Special Issue on the CADE-13 Automated Theorem Proving System Competition, pp. 247–252.

[Wol02] Stephen Wolfram. *The Mathematica Book*. Cambridge University Press, 2002.

[WR10] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*. 2nd ed. Vol. I. Cambridge, UK: Cambridge University Press, 1910.

[WS02] Christoph Walther and Stephan Schweitzer. *The VeriFun Tutorial*. Tech. rep. VFR 02/04. Programmiermethodik, Technische Universität Darmstadt, 2002.

[Zim04] Jürgen Zimmer. “A Framework for Agent-based Brokering of Reasoning Services”. In: *Proceedings of the Mexican International Conference on Artificial Intelligence 2004*. Ed. by Raul Monroy, Gustavo Arroyo Figueroa, and L. Enrique Sucar. to appear. Springer-Verlag, 2004.

[ZK02] Jürgen Zimmer and Michael Kohlhase. “System Description: The MathWeb Software Bus for Distributed Mathematical Reasoning”. In: *Automated Deduction — CADE-18*. Ed. by Andrei Voronkov. LNAI 2392. Springer Verlag, 2002, pp. 247–252. URL: `http://kwarc.info/kohlhase/papers/cade02.pdf`.