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 [**Kohlhase:OMDoc1.6spec**].

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.

Contents

Part I

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 I: 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 II 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 IV 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 V 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 VI 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 VII 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/`[1]

EdN:1

[1] EdNote: For the moment, this URI is not valid yet, use `https://svn.omdoc.org/repos/omdoc/trunk/examples/spec/`

Part II

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 [**Bourbaki:a74**], which we show in Figure 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 0 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 1 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 [**Kohlhase:OMDoc1.6projects**]), or the In2Math knowledge base [**Dahn:sbt01**; **BauBlo:adtmpd01**], 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 [**Kohlhase:OMDoc1.6projects**] can re-assemble the text in different orders.

In Chapter 2, 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), 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 3 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 [**Constable86**], Hol [**GoMe93**], Mizar [**Rudnicki:aomp92**] and OMEGA [**BenzmuellerEtAl:otama97**].

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 1: A fragment from Bourbaki's algebra [**Bourbaki:a74**]

Chapter 1

Minimal OMDoc Markup

It actually takes very little change to an existing document to make it a valid OMDoc document. We only need to wrap the text into the appropriate XML document tags. In Listing 1.1, 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[2] is the simplest way to create an OMDoc document. EdN:2

Listing 1.1: The outer part of the document

```
1  <?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"
6         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>
11     <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>
16       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>
21     <cc:license>
         <cc:permissions reproduction="permitted" distribution="permitted"
                         derivative_works="permitted"/>
         <cc:prohibitions commercial_use="permitted"/>
         <cc:requirements notice="required" copyleft="required" attribution="required"/>
26     </cc:license>
     </metadata>

     <omtext xml:id="all">
       <h:p xml:lang="en">
31       {\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>
36 </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.

[2] EdNote: redo numbers

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 [**Kohlhase:OMDoc1.6spec**]), 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.	[**Kohlhase:OMDoc1.6spec**]
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 [**Kohlhase:OMDoc1.6spec**]).	?? 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 2, but spurious namespace prefix declarations are not a problem in the XML world.	[**Kohlhase:OMDoc1.6spec**]
8	the namespace declaration for the document; if not prefixed, all elements live in the OMDoc namespace.	[**Kohlhase:OMDoc1.6spec**]
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	[**Kohlhase:OMDoc1.6spec**]
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 [**FreBor:MIME96**] 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 2

Marking up the text structure and statements

In the next step, we analyze and mark up the structure of the text of the further, and embed the paragraphs into markup for mathematical statements or text segments. Instead of lines 32–36 in Listing 1.1, we will now have the representation in Listing 2.1.

Listing 2.1: 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 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 3

Marking up the Formulae

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

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

Listing 3.1: 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 II), 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.	[**Kohlhase:OMDoc1.6spec**]
5	The start tag of a theory. We need this, since symbols and definitions can only appear inside `theory` elements.	[**Kohlhase:OMDoc1.6spec**]
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 3.1 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. **??**
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. **??**

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

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 3.1 is analogous. Thus we have treated the first definition in Figure 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 3.1 above.

Finally, we come to the examples at the end of our fragment. In the markup shown in Listing 3.2 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 3.1, 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 3.2: 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"/>.
```

```
46          </h:p>
        </example>

        <example xml:id="e2.magma" for="#law_of_composition" type="for">
          <h:p>
51          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 ...
56        </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 3.3: Examples embedded into a theory

```
1   <theory xml:id="magmas">
      <imports xml:id="imp3" from="background.omdoc#products"/>
      <imports from="http://mbc.mathweb.org/omstd/relation1.omdoc#relation1"/>
      ...
      <theory xml:id="magmas-examples"
6       <imports xml:id="imp4">
            from="http://www.omdoc.org/examples/omstd/fns1.omdoc#fns1"/>
        <imports xml:id="imp5" from="background.omdoc#nat"/>
        <imports xml:id="imp6" from="background.omdoc#functions"/>
        ...
11    </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 4

Full Formalization

The final step in the migration of the text fragment involves a transformation of the mathematical vernacular in the statements into some logical formalism. The main benefit of this is that we can verify the mathematical contents in theorem proving environments. We will start out by dividing the first definition into two parts. The first one defines the symbol `law_of_composition` (see Listing 4.1), and the second one `magma` (see Listing 4.2).

Listing 4.1: 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 4.2.

Listing 4.2: 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 [**Kohlhase:OMDoc1.6spec**]), 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 4.3: A formalized magma example

```
<example xml:id="e11.magma" for="#law_of_composition"
         type="for" assertion="e11.magma.ass">
```

the value of A, where all the occurrences of X have been replaced by B. See [**Barendregt80**; **Andrews02**] for an introduction.

```
  <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>
  <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 III

OpenMath Content Dictionaries

Content Dictionaries are structured documents used by the OpenMath standard [**BusCapCar:2oms04**] 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 [**URL:omcd-core**] 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 4.4: 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 4.4 is the OMDoc document in Listing 4.5.

The first 25 lines in Listing 4.4 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 [**Kohlhase:OMDoc1.6spec**] since it suffices in this case, this is indicated by the `modules` attribute on the `omdoc` element.

Listing 4.5: 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"/>
```

```
100             <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"/>
105                   <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>
110               </OMBIND>
                </OMA>
              </OMA>
            </OMBIND>
          </FMP>
115     </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 4.1: A human-oriented presentation of the OMDoc CD

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

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 4.2: A human-oriented presentation in German

Part IV

Documented Ontologies

OMDoc can be used for marking up ontologies. In particular, we use this for marking up the metadata ontologies, e.g. [**Kohlhase:OMDoc1.6spec**]. Until we put a revised version here, please see [**LK:MathOntoAuthDoc09**].

Part V

Structured and Parametrized Theories

In Part II 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 [**CoFI:2004:CASL-RM**]) standard to the Pvs theorem prover [**OwRu92**], since it uses most language elements without becoming too unwieldy to present.

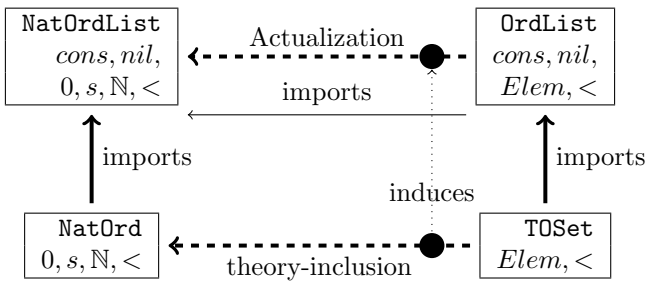

Figure 4.3: A Structured Specification of Lists (of Natural Numbers)

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 4.3. 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 [**Kohlhase:OMDoc1.6spec**] 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 **??**). 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 V 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 [**Kohlhase:OMDoc1.6spec**]) 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>
```

[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="TOSet" name="ord"/>
      <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 [**Kohlhase:OMDoc1.6spec**] 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 [**Kohlhase:OMDoc1.6spec**] 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 VI

A Development Graph for Elementary Algebra

We will now use the technique presented in the last chapter for the elementary algebraic hierarchy. Figure 4.4 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.

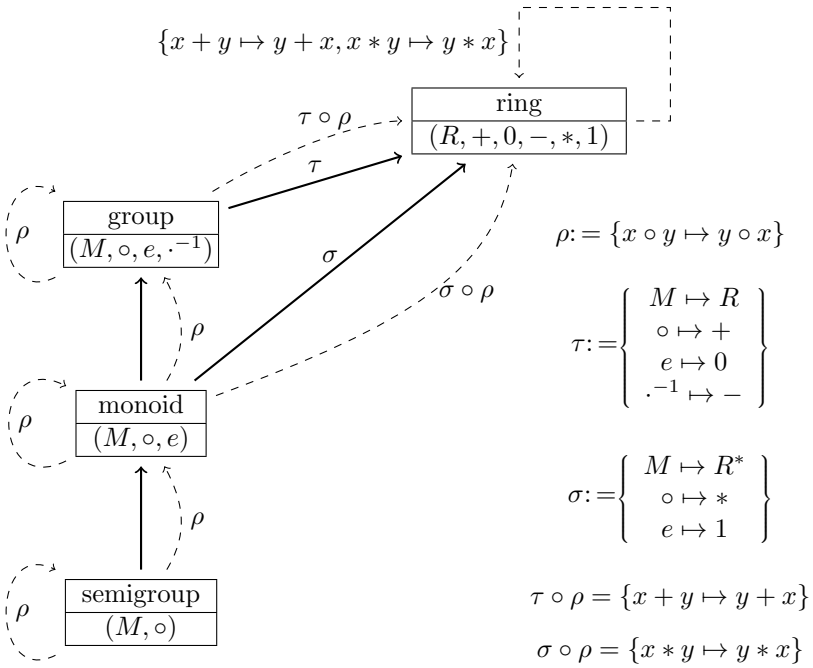

Figure 4.4: A Development Graph for Elementary Algebra

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 4.4), 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 4.4). 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 VII

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 [**Kohlhase:OMDoc1.6projects**]) and can be produced e.g. by CPoint (see [**Kohlhase:OMDoc1.6projects**]).

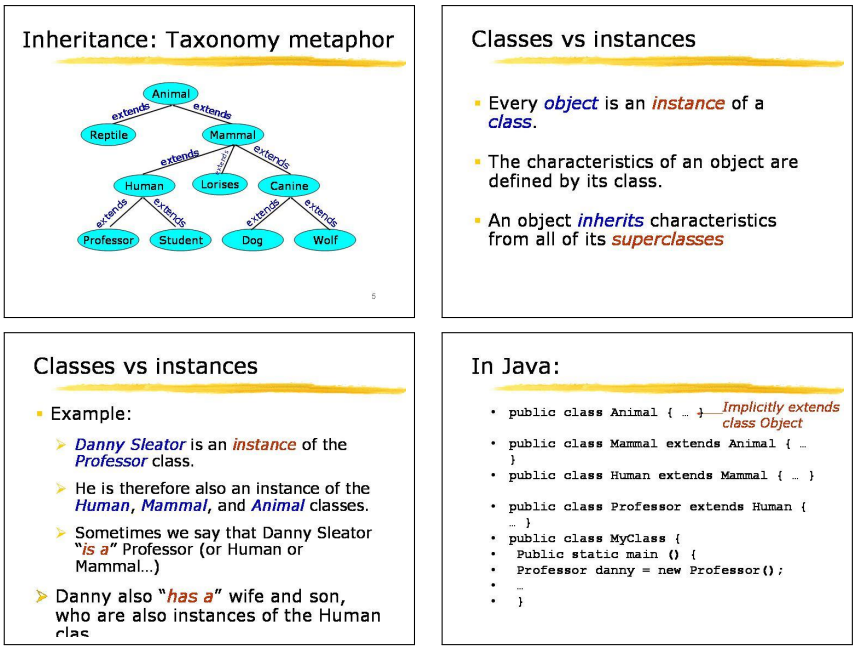

Figure 4.5: Three slides from 15-211

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 4.5. 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 5.

[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 5

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 4.5 and its structure explicit so that a knowledge management system like MBase (see [**Kohlhase:OMDoc1.6projects**]) or knowledge presentation system like ActiveMath (see [**Kohlhase:OMDoc1.6projects**]) 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 5.1. 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 5.1: The OMDoc Representation for Slide 1 from Figure 4.5

```
<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 6.1.)

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 5.2: The OMDoc Representation for Slide 2 from Figure 4.5

```
<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 4.5. 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 5.3 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 5.3. 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 5[4].

Listing 5.3: The OMDoc Representation for Slide 3 from Figure 4.5

```
<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
           &#x201C;<h:span style="font-style:italic;color:red">is a</h:span>&#x201D;
31         Professor (or Human or Mammal&#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 5.4, 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 5.4 — 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 5.4. 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)[3] EdN:3
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 4.5.

Listing 5.4: 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>
```

[3] EdNote: MiKo: this needs more discussion, here we have an introduction of a new concept/notation/verbalization by way of an example. This should probably also be discussed in the statements module.

```
</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 5.5. 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 5.5: 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 6

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 4.5 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 5.2). 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 6.1: The Narrative OMDoc for Figure 4.5

```
...
<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>
...
```

BOP:4

[4] Old Part: part of this also used (copied to) language design in the spec

Chapter 7

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 7.1. 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 [**Kohlhase:OMDoc1.6spec**]. The arrows represent the theory inheritance structure, e.g. Theory $\mathcal{F}$ imports theory $\mathcal{N}$.

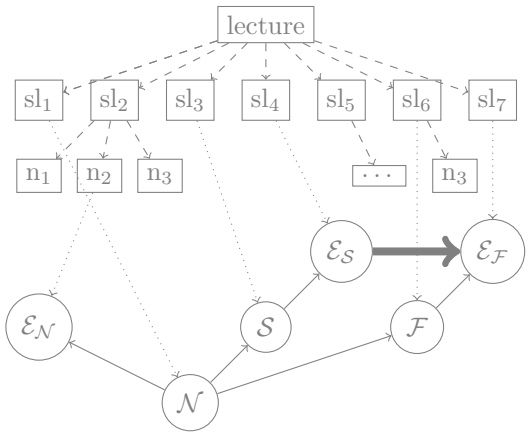

Figure 7.1: An Introduction of a Concept via a Straw-Man Theory

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 7.1, 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 [**MelBue:krma03**] generates a simple narrative structure (the presentation; called a personalized book) from the underlying content
EOP:4 structure (given in OMDoc) and a user model.

Chapter 8

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 [**Kohlhase:OMDoc1.6projects**] 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 VIII

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 [**GudHad:soapad03**] (see [**Mitra:soapPrimer03**] 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 [**Weidenbach:sv97**] via the HTTP GET request in Listing 8.1 to a MathWeb-WS broker node.

Listing 8.1: 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 8.2 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 [**FraKoh:mabdl99**; **ZimKoh:tmsbdmr02**] ("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) [**xmlrpc**]). 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 8.2: 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 8.3[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 8.2).

Listing 8.3: 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 8.3 depends on the information given in the theory `lovelife` referenced in the `theory` attribute in the `assertion` element (see [**Kohlhase:OMDoc1.6spec**] 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 [**Kohlhase:OMDoc1.6projects**] 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 8.4 to the MBASE service at `http://mbase.mathweb.org:8080`.

Listing 8.4: 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 8.5. Here, the SOAP envelope contains the OMDoc representation of the requested theory (irrespective of what the internal representation of MBASE was).

Listing 8.5: The Background Theory for Message 8.3

```
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>∀x.loves(mary, x) ⇒ 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 8.3; and the SPASS service might reply to the request with the message in Listing 8.6 which contains an OMDoc representation of a proof (see [**Kohlhase:OMDoc1.6spec**] for details). Note that the `for` attribute in the `proof` element points to the original assertion from Listing 8.3.

Listing 8.6: 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 8.5 (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 [**Kohlhase:OMDoc1.6spec**].

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 [**Meier:sdttom00**] that transforms resolution proofs (the native format of the SPASS prover) to natural deduction proofs.

Index