Presenting Mathematical Content With Flexible Elisions

Michael Kohlhase, Christoph Lange, Florian Rabe
Computer Science, Jacobs University Bremen
D-28759 Bremen, Germany
http://kwarc.info

October 17, 2007

Mathematicians frequently elide brackets or symbols in formulae to concentrate on essential facts and to avoid distracting experienced mathematicians with notation that can easily be deduced from context. In this paper we propose a extension of the notation specification infrastructure in OMDoc by functionality for flexible elisions.

Contents

1 Introduction

Over the last three millennia, mathematics has developed a complicated two-dimensional format for communicating formulae (see e. g. [Caj93, Wol00] for details). Changes in notation have been influential in shaping the way we calculate and think about mathematical concepts, and understanding mathematical notations is an essential part of any mathematics education.

Content Markup formats for mathematics such as OpenMath [BCC+04] and MathML [ABC+03] concentrate on the functional structure of mathematical formulae, thus allowing mathematical software systems to exchange mathematical objects. For communication with humans, these formats rely on a "presentation process" (usually based on XSLT style sheets) that transforms the content objects into the usual two-dimensional form used in mathematical books and articles.

After a conceptual analysis of the presentation process and a survey of recent approaches to specify the notations of mathematical symbols and how to use these specifications to transform content-oriented mathematical documents to a presentation-oriented format like PDF or HTML, we introduce a revised specification of the presentation module of OMDoc that is enabled for elisions. We conclude by proposing how information about elisions that a presentation algorithm performed can be included in the presentation markup and used by a client application.

1.1 Presentation as Composition and Elision

Many such presentation processes have been proposed, and all have their strengths and weaknesses. In this paper, we conceptualize the presentation of mathematical formulae as consisting of two components: the two-dimensional **composition** of visual sub-presentations to larger ones and the **elision** of formula parts that can be deduced from context.

Most current presentation processes concentrate on the relatively well-understood composition aspect and implement only rather simple bracket elision algorithms. But the visual renderings of formulae in mathematical practice are not simple direct compositions of the concepts involved: mathematicians gloss over parts of the formulae, e. g. leaving out arguments, iff they are non-essential, conventionalized or can be deduced from the context. Indeed this is part of what makes mathematics so hard to read for beginners, but also what makes mathematical language so efficient for the initiates. A common example is the use of $\log(x)$ or even $\log x$ for $\log_{10}(x)$ or similarly $[\![t]\!]$ for $[\![t]\!]^{\phi}_{\mathcal{M}}$, if there is only one model $\mathcal{M}$ in the context and ϕ is the most salient variable assignment. Another example are the bracket elision rules in arithmetical expressions: $ax+y$ is actually $(ax)+y$, since multiplication "binds stronger" than addition. Note that we would not consider the "invisible times" operation as another elision, but as an alternative presentation. Finally, there are extreme examples of elision or substitution like Church's dot notation, where a dot stands for a left bracket, whose mate is as far to the right as consistent with the remaining (un-elided) brackets. For instance $(a \cdot .\, b + c) - d$ stands for $(a \cdot (b + c)) - d$, and $\forall x, y. \phi \wedge \psi$ stands for $\forall x, y(\phi \wedge \psi)$.

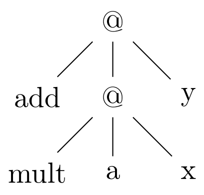
Now that we have convinced ourselves that the elision is an important component of generating high-quality presentations, let us reconsider the presentation process itself to see how composition and elision interact.

We will start from the observation that in content markup formalisms for mathematics formulae are represented as "formula trees". Concretely, we will concentrate on OpenMath objects, the conceptual data model of OpenMath representations, since it is sufficiently general, and work is currently under way to re-engineer content MathML representations based on this model. Furthermore, we observe that the target of the presentation process is also a tree expression: a layout tree made of layout primitives and glyphs, e. g. a presentation MathML expression. If we make examples with TEX/LATEX it is only since it is universally understood; here, the layout tree is the parse tree implicit in the linear TEX/LATEX string. This notwithstanding, we finally observe that even though formula presentations are two-dimensional in principle, large parts are more or less linear, and therefore mathematical notation relies on brackets to allow the reader to reconstruct the content structure from the presentation.

1.2 Characteristics of Mathematical Symbols

In a nutshell, OPENMATH objects are trees built up from **variables** and **symbols** at the leaves and **applications** and **binders** as internal nodes.

For our purposes, symbols and applications are the most important concepts, therefore let us fortify our intuition with the example on the right: the sum $ax + y$ above would be represented by a tree with an application at the root, whose first child is the symbol[1] for addition, the third child is the variable with the name y, and the second child is another application whose children are the symbols for multiplication and the variables with names a and x.



As part of the functional notation of mathematics popularized by Leibniz and Newton for algebra, the visual appearance of a subformula was largely determined by its head operator, which nowadays carries much of the notational conventions, which are usually phrased in terms of the following notational characteristics of a symbol:

fixity (for operators): Is the operator displayed as a prefix (like a function symbol), as a postfix (like the factorial symbol !) or as an infix (like most arithmetic operators)? Operators of higher arity can also have a "mixfix" style; consider the 4-ary typing judgment operator $\Gamma \vdash_\Sigma t\colon \alpha$, which states that a term t has type α in a variable context Γ and a signature Σ.

left and right brackets: If an operation is bracketed, are round, square, curly, angle, or other brackets used? In fact brackets in mathematical vernacular are mostly round brackets; technical notations sometimes use others, e.g. MATHEMATICA uses square ones throughout. Note that not all parts of a presentation that look like brackets are indeed. For instance in a half-open interval as $]a;b]$, is actually mixfix operator where the bracket-like glyphs "]" look and conventionally stretch like brackets, but do not match and cannot be elided like brackets. They are easy to mistake for brackets, since they (as all fully inhabited mixfix presentations) make brackets for the arguments unnecessary. We would consider it a semantic anomaly to present an "interval" operator as an infix ";" with obligatory left and right brackets "]" and "]".

precedence (for operators): If an operation O with a high precedence occurs as an argument of an operator with a lower precedence, the brackets around O can be elided.

associativity (for infix operators): To elide brackets for n-ary operators ($n > 2$), it must be known whether they are associative (i. e. $(a \circ b) \circ c = a \circ (b \circ c) =: a \circ b \circ c$) or obey "left-associative" or "right-associative" elision conventions (e.g. $\alpha \to \beta \to \gamma := \alpha \to (\beta \to \gamma)$ for right-associative function types).

The first two characteristics specify the composition of the visual representations, whereas the latter two are concerned with bracket elision. Furthermore, the **role** of a symbol in the formula context needs to be taken into account: Is it a simple *constant* like $\mathbb{R}$, a *function* that is applied to arguments, or a *binder* that binds variables like $\forall x.p(x)$. This is most pronounced for functions that can occur either applied to arguments or as the concept alone. Presentations differ for these occasions, e.g. for the absolute value function, we write something like $|\cdot|$ with a "dummy argument" $\cdot$ when it is not in an applied context.

[1] Note that a symbol is not a glyph — these are often confused, the symbol here stands for the mathematical concept of a sum, not the glyph +.

1.3 Notation Definitions

Mathematical formulae given in the XML-based structural formats OpenMath and Content MathML are commonly presented by implementing one XSLT template [Kay06] per symbol, which specifies how to transform this symbol to the output format—another XML format like Presentation MathML, or a non-XML format like TeX, for example. Appropriate templates for the arguments are usually applied recursively.

The following listing shows a straightforward way to transform the exponentiation operator from OpenMath to TeX:

Listing 1: An XSLT presentation template

```
<xsl:template match="om:OMA[om:OMS[@cd='arith1' and @name='power']]">
  <xsl:apply-templates select="*[2]"/> <!-- first argument -->
  <xsl:text>^{</xsl:text>
  <xsl:apply-templates select="*[3]"/> <!-- second argument -->
  <xsl:text>}</xsl:text>
</xsl:template>
```

To save authors from the tedious and error-prone task of writing similar templates for every symbol and notation variant, different facilitations have been invented. The probably earliest examples were the presentation architecture in OMDoc 1.0 [Koh00] and Naylor's&Watt's OpenMath conversion [NW01]. Both supply XML-based **notation definitions** that can be transformed into a suitable XSLT conversion style sheet by a meta-stylesheet.

In OMDoc, a template like the one above would be generated from a `presentation` element of the form

```
<presentation for="power" theory="arith1" role="application" class="1">
  <style format="TeX">
    <recurse select="*[2]"/><text>^{</text><recurse select="*[3]"/><text>}</text>
  </style>
  ...
<presentation>
```

where the `match` attribute is generated from the attributes on the `presentation` element and the contents from the ones on the `use` element. In contrast to this, where notations definitions are thought as directed rules from content to presentation, Naylor's&Watt's framework views them as equivalences between representations. For the example above we would have

Listing 2: A Template-Based Notation Definition

```
<Notation>
  <version style="1">
    <tex>\arg{a1}{n}^{\arg{a2}{m}}</tex>
    ...
  </version>
  <semantic-template>
    <OMA><OMS cd="power" name="arith1"/>
      <OMV name="n" id="a1/>
      <OMV name="n" id="a2/>
    </OMA>
  </semantic-template>
</Notation>
```

We will call the first approach to notation specification **symbol-based**, as the notation is tied to a concrete symbol, and the second one **template-based**. Note that the template-based approach is in principle more directly invertible (e.g. for parsing), and more expressive (it would e.g. allow to present $log(e,x)$ as $\ln(x)$ and $log(b,x)$ as $\log_b(x)$). But this has not been realized in practice, since the presentation of mathematical formulae *is indeed* tied to symbols, so all templates are indeed one-level like the one in the example above. On this fragment, both approaches are equivalent. Note furthermore that many of the notational characteristics of a symbol mentioned in section 1.2 actually do not belong to the symbol but to its presentation. For example, the division operator, presented as $\cdot/\cdot$, has a higher precedence than $\frac{\cdot}{\cdot}$—consider $1/(a+b)$ vs. $\frac{1}{a+b}$.

In OMDoc 1.2 [Koh06], all of these characteristics can be specified as attributes to one out of potentially multiple `presentation` elements that refer to a `symbol`. For some aspects like mix-fixity, this declarative syntax is not yet expressive enough; in this case, embedded XSLT templates must be used, losing declarativity. In this paper, we rethink the notation definition system and propose a declarative system extending ideas from the notation definition system in the Isabelle theorem proving environment [Pau05].

2 A Mixfix Presentation Model for Flexary Terms

The ISABELLE presentation process allows to model all of the symbol characteristics discussed in section 1.2 by a single notation definition: the **mixfix declaration**. For an n-ary symbol, f is a declaration of the form $w_0\boxed{p_1|_{\pi_1}}w_1\boxed{p_2|_{\pi_2}}\ldots\boxed{p_n|_{\pi_n}}w_n : p$, where the w_i are strings in the output language and p is the **output precedence** of f. We call the boxes **argument specifications**, the p_i are **input precedences**. In accordance with Prolog, which, in contrast to ISABELLE, predefines precedences for many common mathematical operators, and staying compatible with previous OMDOC standards, we assign low precedence values to strongly binding operators and high precedence values to weakly binding operators.

In the presentation process, a term t is presented as a string $\overline{t}^q$, given an initial precedence q, according to the following definition:

$$\overline{f(t_1,\ldots,t_n)}^q = w_0\overline{t_{\pi_1}}^{p_1}w_1\ldots w_{n-1}\overline{t_{\pi_n}}^{p_n}w_n$$

Note that the w_i contain meta-tokens for brackets (actually pretty-printing blocks that also handle indentation in ISABELLE) that will be elided according to the "current precedence" at that point.

A declaration of a right-associative infix operator $\rightarrow$ with precedence p can be modeled by a mixfix declaration $\boxed{p-1|_1} \rightarrow \boxed{p|_2} : p$. That means, the first argument must be an atom or an operator that binds stronger than $\rightarrow$, or is bracketed otherwise, and the second argument must have a maximum output precedence of p, which would present an input of $a \rightarrow (b \rightarrow c)$ as $a \rightarrow b \rightarrow c$.

However, the ISABELLE model is not directly applicable to presentation of OPENMATH objects, as ISABELLE crucially depends on the assumption that terms have a fixed arity. In OPENMATH objects, applications can have flexible arities depending on the operator, i. e. application nodes can have different numbers of children, even if they coincide in the first child (the function). This is used to model a **flexary** addition function, i. e. a function that takes any number of arguments, e. g. $@(add,1,2,3,4)$. In ISABELLE, we would have to model addition as a binary, associative operator, and the term above as $plus(plus(1,2),plus(3,4))$ which would have the same presentation $1+2+3+4$, but a different structure. Similarly, ISABELLE only supports unary binding constructions.

We will now extend the presentation model to the flexary case, and the types of OPENMATH objects not covered by ISABELLE, and come back to flexible elision and abbreviation of non-brackets (we will come back to this in section 3).

The first part of the flexary extension is to generalize the idea of an argument vector, to a list of **components** of an OPENMATH object O as follows (where indices starts at 0):

- for applications and errors: the list of children of O, e. g., $(f,a_1,\ldots,a_n)$ for the object $@(f,a_1,\ldots,a_n)$,
- for bindings: the binder, followed by the list of variables, followed by the body, e. g., $(b,x_1,\ldots,x_n,a)$ for the binding $\beta(b;x_1,\ldots,x_n;a)$,
- for attributions: a three-element list consisting of the key and the value of the first attribution and the original object with the first attribution removed, e. g., $(k_1,v_1,\alpha(k_2 \mapsto v_2,\ldots,k_n \mapsto v_n;a))$ for $\alpha(k_1 \mapsto v_1,\ldots,k_n \mapsto v_n;a)$,
- otherwise: the object itself as a one-element list.

As a running example, consider the space of n-ary functions. We can model the function space constructor as a flexary function symbol nfuns whose last argument is the codomain and all previous ones the domains. We would like to present an OPENMATH object $O{:=}$

$@(\mathsf{nfuns}, A_1, \ldots, A_n, C)$ as $A_1 \times \cdots \times A_n \to C$. By the definition above, the set of components of the object O is the list $\mathsf{nfuns}, A_1, \ldots, A_n, C$.

In an obvious generalization of the mixfix notations, we would like to represent the notation definition for nfuns as $\boxed{\times : p-1|_1^{n-1}} \to \boxed{p|_n} : p$. In this, we had to generalize the argument specification for the domains so it can deal with an argument list. The new **argument mapping specification** represented by the box here specifies the range of components it covers (the first $n-1$ arguments in our example), their input precedence[2], and finally, the **separator**, (the $\times$ operator in our example). The precedences are chosen so that the arrows associate to the right in our example.

The general form a of a component mapping specification is $\boxed{sep : N|_b^e}$, where sep is the separator, N is an arbitrary notation that may contain p to recurse into the arguments with input precedence p, and b/e are the **component sequence begin/end indices**.

Now let $w_0\boxed{s_1 : p_1|_{b_1}^{e_1}}w_1 \ldots w_{n-1}\boxed{s_n : p_n|_{b_n}^{e_n}}w_n : p$ be a notation declaration for a function f, then

$$\overline{@(f, t_1, \ldots, t_m)}^q = w_0\overline{t_{b_1}}^{p_1}s_1 \ldots s_1\overline{t_{e_1}}^{p_1}w_1 \ldots w_{n-1}\overline{t_{b_n}}^{p_n}s_n \ldots s_n\overline{t_{e_n}}^{p_n}w_n$$

Note that an ISABELLE-style argument specification $\boxed{p|_\pi}$ can be regained as $\boxed{\epsilon : p|_\pi^\pi}$, where ϵ is e. g. the empty string. This legitimizes our example above.

Note that again, the w_i can contain brackets and other material that can be elided taking precedence and other factors into account. We will describe the necessary infrastructure at a conceptual level in the next section.

[2]Note that the input precedences of all members of the component range is equal, since they are indistinguishable as list members.

3 Flexible Elisions

There are several situations in which it is desirable to display only some parts of the presentation:

- Brackets that are redundant due to operator precedences can be omitted.
- Arguments that are strictly necessary are omitted to simplify the notation, and the reader is trusted to fill them in from the context.
- Arguments are omitted because they have default values. For example $\log_{10} x$ is often written as $\log x$.
- Arguments whose values can be inferred from the other arguments are usually omitted. For example, matrix multiplication formally takes five arguments, namely the dimensions of the multiplied matrices and the matrices themselves, but only the latter two are displayed.

Typically, these elisions are confusing for readers who are getting acquainted with a topic, but become more and more helpful as the reader advances. For experienced readers more is elided to focus on relevant material, for beginners representations are more explicit. In the process of writing a mathematical document for traditional (print) media, an author has to decide on the intended audience and design the level of elision (which need not be constant over the document though). With electronic media we have new possibilities: we can make elisions flexible. The author still chooses the elision level for the initial presentation, but the reader can adapt it to her level of competence and comfort, making details more or less explicit.

To provide this functionality, we give each token in the w_i an integer **visibility level** and
P(1) group them into **elision groups**. High levels mean high elidability.

Brackets form a special elidability group of their own. Assume we calculate $\overline{O}^p$ for an OpenMath object O, and the notation definition chosen to present O has the output precedence q and returns the the token string B. Then every bracket token in B is given the visibility level $d := \max\{q-p, 0\}$. Thus brackets where the difference between input and output precedence is higher are more elidable because they are easier to reconstruct. Brackets with a visibility level 0 are necessary and cannot be elided.

In the special cases where p and q are both positive or both negative infinity, we put $p-q$ to be 0 resulting in these brackets being unelidable. If O is presented as a top-level object, i.e., not as the result of a recursive call to the presentation procedure, p is assumed to be negative
P(1) infinity, which means that top-level brackets have the highest elidability.

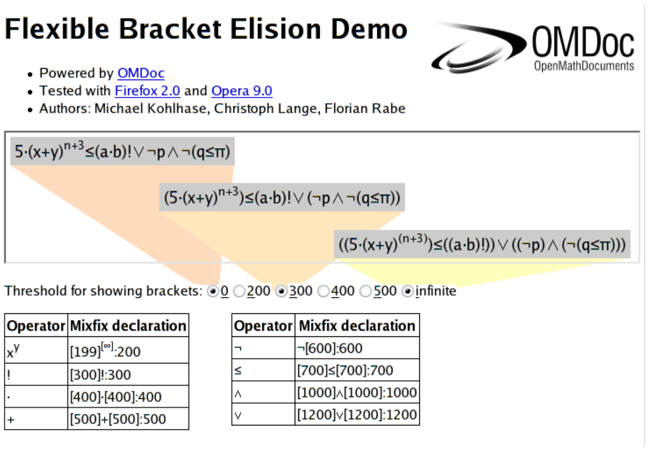
P(2) Elision can take various forms in print and digital media. In static media like traditional print on paper or the PostScript format, we have to fix the elision level, and can decide at presentation time which elidable tokens will be printed and which will not. In this case, the presentation algorithm will take visibility thresholds T_g for every elidability group g as a user parameter and then elide (i.e. not print) all tokens in visibility group g with level $l > T_g$.

In an output format that is capable of interactively changing its appearance, e.g. dynamic XHTML+MathML (i.e. XHTML with embedded Presentation MathML formulas, which can be manipulated via JavaScript in browsers), an application can export the the information about elision groups and levels to the target format, and can then dynamically change the visibility thresholds by user interaction.

We have implemented a prototypical flexible elision scheme for dynamic XHTML, by generating tokens for all elidable parts, wrapping them in `xhtml:span` elements and exporting the elision group and level information as XML attributes `omdoc:egroup` and `omdoc:elevel` on these elements. All `xhtml:span` elements with visibility group g and level $l > T_g$ are augmented with

[1] New Part: @FR/CL, please check, and adapt the examples. This is important stuff, we want people to understand it.

[2] New Part: @CL/FR: please check this



Flexible Bracket Elision Demo

- Powered by OMDoc
- Tested with Firefox 2.0 and Opera 9.0
- Authors: Michael Kohlhase, Christoph Lange, Florian Rabe

$5\cdot(x+y)^{n+3}\leq(a\cdot b)!\vee\neg p\wedge\neg(q\leq\pi)$

$(5\cdot(x+y)^{n+3})\leq(a\cdot b)!\vee(\neg p\wedge\neg(q\leq\pi))$

$((5\cdot(x+y)^{(n+3)})\leq((a\cdot b)!))\vee((\neg p)\wedge(\neg(q\leq\pi)))$

Threshold for showing brackets: [✓ 0] 200 [✓ 300] 400 500 [✓ infinite]

Operator	Mixfix declaration
x^y	$[199]^{[\infty]}$:200
!	[300]!:300
·	[400]·[400]:400
+	[500]+[500]:500

Operator	Mixfix declaration
¬	¬[600]:600
≤	[700]≤[700]:700
∧	[1000]∧[1000]:1000
∨	[1200]∨[1200]:1200

Figure 1: Flexible Elision of Brackets in XHTML

the attribute `style="display:none"`, which elides them. If the user changes the threshold, an embedded JavaScript function can be used to set the display property accordingly. In Figure 1, we show the system with three levels of bracketing thresholds. The same technique would work for XHTML+MathML, if the CSS `display` directive were implemented for MathML elements across browsers. EndNP(2)

4 The Flexary Mixfix Model in OMDoc2.0

We introduce extensions making the declarative OMDoc syntax of presentations more expressive. Thus we can drop the support for embedded XPath and XSLT expressions, which represented big hurdles for implementations.

4.1 An XML Encoding for Flexary Mixfix Declarations

The presentation of mathematical objects is specified by `presentation` elements as follows:

Element	Attributes		Content
	Required	Optional	
presentation	for, role, format	precedence	⟪*item*⟫*
map	begin, end	egroup, elevel	separator? ⟪*item*⟫*
arg	pos	egroup, elevel	EMPTY
separator		egroup, elevel	⟪*item*⟫*
recurse		egroup, elevel, precedence	EMPTY
attribution	cd, name	egroup, elevel, cdbase	⟪*item*⟫*
text	value	crossref	#PCDATA
element	name	ns, egroup, elevel, crossref	attribute* ⟪*item*⟫*
attribute	name	ns	⟪*item*⟫*
name			EMPTY
⟪*item*⟫ $\hat{=}$ (text | element | map | attribution | recurse)			

The attributes of `presentation` elements have the following meaning:

- `for`: a URI reference to a symbol to which the presentation applies, or to the theory for which a default presentation is given.
- `role`: the role of the presented symbol to which the presentation applies. The same symbol can be used in different roles[3], and for each role a different presentation may be necessary. Possible values are `binder`, `application`, `constant`, `variable`, `key`, and `error`, mirroring the possible values of the corresponding attribute of the presented symbol (cf. [Koh06, chapter 15.2.1]).
- `format`: A whitespace-separated list of target formats for which the presentation element can be used. Typical values are `ascii`, `html`, `pmathml`, `latex`, and `default`. The latter acts as a backup that can be applied if there is no specific `presentation` element for a given format.
- `precedence`: the output precedence of the term, which is used for bracket elimination. It must be an integer, or the string "infinity", and defaults to 0.[3]

e(3)

The exact syntax of the children of the `presentation` elements is introduced when we specify their presentation behavior. The two optional attributes `egroup` and `elevel`, which may occur on almost all elements that relate to presentation are used to specify the elision group and elision level. If `egroup` is omitted, it defaults to the home theory that the containing `presentation` element applies to.

[3]Do not confuse the symbols' role in the notational context specified in the `role` attribute here with the role of a symbol in the OpenMath2 standard

[3]EdNote: IMHO reicht $[0,\infty)$; $(-\infty,+\infty)$ braucht man nicht. –CL

4.2 Determining Notation Definitions

An application that displays a mathematical object O must process O recursively looking up the needed `presentation` element once in the beginning and then whenever a `recurse` element is encountered. This lookup is guided by the attributes `for`, `role`, and `format`, which determine when a `presentation` element is eligible. If these attributes uniquely determines a `presentation` element, it must be used by the application. Otherwise, the application is free to choose the most appropriate one. This choice is non-trivial and is the topic of ongoing research [KMM07].

More precisely, to present an object O, a `presentation` element is chosen as follows.

- If O is a binding, application, or error object with a symbol as the first child, a presentation for this symbol with the appropriate role is chosen. If none exists or if the first child is not a symbol, a presentation for the home theory of O with the appropriate role is chosen. [4] EdNote(4)
- If O is an attributed object, a presentation for the key of the first attribution with role `key` is chosen. If none exists, a presentation for the home theory of O with that role is chosen.
- If O is a symbol, a presentation for O with role `constant`, or, if none is found, a presentation for the home theory of O with that role is chosen.
- If O is a variable, a presentation for it that is attributed to it in its binder as specified in OMDoc 1.2 is chosen. If none exists, a presentation for the home theory of the binder with role `variable` is chosen.
- The presentation of constants and foreign objects must be determined independently by the application.

Of course, in all these cases only `presentation` elements with the intended target format are eligible. Also, if the above algorithm fails, the presentation system may fail or fall back first to a target format-specific presentation with the correct role, then to a general presentation with the correct role, and finally to a presentation defined by the system.[5] EdNote(5)

4.3 Generating Presentations for OpenMath Objects

Once a `presentation` element, say with a body B, has been chosen for O, the presentation is obtained as a function of O and B. We say that B is rendered in context O. Let n be the number of components of O; furthermore, let $\pi_n(x) := x \ mod \ n$, if $x \in [-n; n-1]$, and fail otherwise. Each child of B is rendered separately as defined in the following, and the results are concatenated. The rendering recurses over the structure of O, and the recursive calls additionally pass the output precedence as a parameter P; for the initial call, this parameter is set to negative infinity.[6] EdNote(6)

An `arg` element A is interpreted by selecting and rendering a component of O. , and let i be the value of the required attribute `pos` and let p be the value of the (optional) attribute `precedence` of A defaulting to P if it is missing. Then `arg` recursively calls the presenation procedure with precedence p on the $\pi_n(i)$-th component of O. In particular, for applications, $i = 0$ renders the operator and, e.g., $i = 2$ renders the second argument. Note that $\pi_n(i) \in [0, \dots, n-1]$; this allows for conveniently accessing, e. g., the last component by the index -1.

For example the presentation of a typing judgment $\Gamma \vdash_\Sigma t : T$ in LaTeX (expressing that t has type T in context Γ over the signature Σ) can be specified as follows:

Listing 3: Presenting a Typing Judgment

```
<symbol name="typing−judgment" role="application"/>
```

[4] EdNote: in between, fall back to the meta-theory, but we leave out meta-theories here

[5] EdNote: @MK: I made this optional since we do not implement it yet.

[6] EdNote: FR@FR: or was it positive?

```
<presentation for="#typing–judgment" role="application" format="latex">
  <arg pos="1"/>
  <text>\vdash_{</text><arg pos="2"/><text>}</text>
  <arg pos="3"/>
  <text>:</text>
  <ar pos="4"/>
</presentation>
```

The only other element needed here is the `text` element, which literally inserts its content into the output string. If the output format is XML-based the `text` generates an XML text node.

A `map` element M is interpreted as a component mapping specification $\boxed{\langle\!\langle sep \rangle\!\rangle : p|^{\pi_n(e)}_{\pi_n(b)}}$, where

1. n is the number of children n of O.
2. b (e) is the value of the attribute `begin` (`end`).
3. p is the value of the `precedence` attribute of the `recurse` element or P if it that is missing.
4. $\langle\!\langle sep \rangle\!\rangle$ is the result of recursively rendering the `separator` child of M in context O. If there is no such child, assume $\langle\!\langle sep \rangle\!\rangle$ to be the empty token sequence.

If M is empty, assume its content to be a child `<recurse/>`.

Note that `<map begin="`n`" end="`n`"><recurse precedence="`p`"/></map>` is equivalent to `<arg pos="`n`" precedence="`p`"/>`.

Listing 4: A Notation Definition for Multiplication

```
<symbol name="times" role="application"/>
<presentation for="#times" role="constant" format="ascii"><text>∗</text></presentation>
<presentation for="#times" role="constant" format="latex"><text>\ast</text></presentation>
<presentation for="#times" role="application" format="ascii latex">
  <text egroup="lbrack">(</text>
  <map begin="1" end="−1">
    <separator><arg pos="0"/></separator>
    <recurse/>
  </map>
  <text egroup="rbrack">)</text>
</presentation>
```

Here we have used two format variants for the constant role to keep the notation definition of the applied form as general as possible: we use `<arg pos="0"/>` to call the presentation of the function symbol as a constant in the presentation of multiplication. We rely on the recursive rendering of the arguments. Note that we have used the `egroup` attribute to classify the bracket characters as such.[7] e(7)

`<attribution cd="`$\langle\!\langle cd \rangle\!\rangle$`" name="`$\langle\!\langle name \rangle\!\rangle$`">A</attribution>` is rendered by rendering A in the context of the value for the key specified by the symbol with name $\langle\!\langle name \rangle\!\rangle$ from the theory $\langle\!\langle cd \rangle\!\rangle$ of the attribution of O. If this attribution is not present, it is rendered as the empty string. Consider for instance the presentation of the universal quantifier with a flexible number of possibly typed variables:

```
<symbol name="univ−quant" role="binder"/>
<presentation for="#univ−quant" role="binder" format="latex">
  <text value="\forall"/>
  <map begin="1" end="−2">
    <separator><text>,</text></separator>
    <recurse/>
    <attribution cd="types" name="type"><text>:</text><recurse/></attribution>
  </map>
  <text egroup="lbrack">.(</text><arg pos="−1"/><text egroup="rbrack">)</text>
</presentation>
```

[7] EdNote: MK: possible inconsistency to what we said earlier, note that we could also use the lbrack and rbrack elements here, maybe?
Unless we hard-code that e.g. an lbrack in PMML is "<mo>(</mo>" and "(" in TEX, I like @egroup better, as, depending on the output format, we need either <text> or <element> to generate brackets. –CL

Note how the bound variables appear with the indices 1 to -2 and the body with index -1. Also note how the colon is only printed if a type ascription is present.

Finally, we can use the `name` element to render the value of the name attribute of O if O is a symbol or variable. For other O, it is not defined. Therefore, `<name/>` may only occur in a `presentation` element with role `constant` or `variable`. This is useful for specifying the default presentation mentioned above, for instance the following presentations can be used for a logic L so that symbols declared in logical signatures of L can be presented without giving separate presentation elements.

```
<presentation for="L" role="constant" format="ascii"><name/></presentation>
<presentation for="L" role="variable" format="ascii"><name/></presentation>
<presentation for="L" role="application" format="ascii">
  <arg pos="0"/>
  <text egroup="lbrack">(</text>
  <map begin="1" end="−1"><separator><text>,</text></separator></map>
  <text egroup="rbrack">)</text>
</presentation>
```

We have only used the ASCII-based LaTeX as an example format above; for XML-based formats, we need extra infrastructure: An `element` element is rendered as an XML element in the obvious way. And an `attribute` element produces an attribute of the governing XML element, the presentation of the body of the element suitably XML-escaped yields the value of the generated attribute. For example the presentation of a symbol that attributes a certain color in presentation MathML could be given as follows. (The value of such an attribution must be an `OMFOREIGN` element that the application can present as text.)

```
<symbol name="color" role="attribution"/>
<presentation for="#color" role="key" format="pmathml">
  <element name="mstyle">
    <attribute name="fontcolor"><arg pos="1"/></attribute>
    <arg pos="2"/>
  </element>
</presentation>
```

To get an intuition for the full power of the elision capability of the new notation definition format, consider the following (contrived) example: assume an object O with component vector (A_1, A_2) which are rendered recursively as `<a1/>` and `<a2 omdoc:egroup="group0" omdoc:elevel="0"/>`. Then in context O, the presentation

```
<presentation>
  <map egroup="group1" level="1">
     <separator egroup="group2" elevel="2">
        <element name="s1" egroup="group3" elevel="3">
          <element name="s2" egroup="group4" elevel="4"/>
        </element>
     </separator>
     <recurse egroup="group5" elevel="5"/>
  </map>
</presentation>
```

is rendered as

```
<a1 omdoc:egroup="group5" omdoc:elevel="5"/>
<s1 omdoc:egroup="group1" omdoc:elevel="1">
  <s2 omdoc:egroup="group4" omdoc:elevel="4"/>
</s1>
<a2 omdoc:egroup="group5" omdoc:elevel="5"/>
```

Note how the visibility information of the outermost element overrides the information of inner elements if the outer element is a structuring element like `map`, `separator,` or `recurse`. We only mention this for the sake of definiteness though. In practice it is unreasonable to provide so much visibility information that it leads to such conflicts.

Cross-References In principle, the treatment of cross-references is left as in OMDoc1.2. In output formats that are capable of cross-references, references from parts of the display to the definition of the symbol may be generated. The `presentation` elements can specify which parts of the presentation this reference is affixed to by using the optional `crossref` attribute. This has values `true` and `false`, the latter being the default. Cross-references are to be attached to all elements for which the value is `true`.

5 Compatibility

Here we show that the proposed infrastructure for notation definitions is a superset of the functionality of other systems. We first show that we can re-gain the OMDoc 1.2 syntax by specifying two new elements that get their meaning as abbreviations of the new functionality. In a second step we develop a template-based variant of our presentation technique to be compatible to (a functional superset of) the ActiveMath notation definition system.

5.1 Compatibility to OMDoc 1.2

The proposed presentation syntax has one disadvantage: Information about the fixity or associativity of a symbol is not marked up explicitly. Therefore, we specify `use` elements in the next section which are very similar to the corresponding OMDoc 1.2 syntax.

The `style`- and `xslt`-based infrastructure for notation definitions in OMDoc 1.2 will not be supported. They have rightfully been described as non-declarative, therefore ill-suited as a definitional mechanism and have tied the presentation procedure to the XSLT programming language. The extended expressivity of the proposed notation definition format seems to cover all the cases, where these two have been needed in practice. We have re-formulated[8] all the notation definitions in the OpenMath standard content dictionaries — which cover the K-14 fragment of mathematics — and have not found need for a procedural fall-back solution. EdNote(8)

5.2 Direct Specification of Symbol Characteristics

We define a presentation element `use` which like its counterpart in OMDoc 1.2 is based on the direct specification of symbol characteristics. But here we define its meaning via abbreviations for certain presentations in the above syntax.

Element	Attributes		Content
	Required	Optional	
use		egroup, elevel, precedence, bracket-style, fixity	rbrack? lbrack? operator? separator?
operator		egroup, elevel	⟪*item*⟫*
lbrack		egroup, elevel	⟪*item*⟫*
rbrack		egroup, elevel	⟪*item*⟫*
⟪*item*⟫ $\hat{=}$ (text | element | map | attribution | recurse)			

A `use` may have the attributes

- `fixity` with values `prefix`, `postfix`, `infixl`, `infixr`, and `infixlr`,
- If fixity is `prefix` or `postfix`: `bracket-style` with values `math` and `lisp`[9], EdNote(9)
- `precedence`.

Its children are at most one each of `lbrack`, `rbrack`, `operator`, and `separator`, all with ⟪*item*⟫* content. [10] Such a presentation specification can be translated easily into the above syntax. For prefix operators, EdNote(10)

[8]EdNote: MK: complete this

[9]EdNote: discuss that! –CL

[10]EdNote: FR: This should be improved: It should be possible that a use element gives only, e.g., fixity and precedence with the rest being inherited from default presentations. The motivation is that fixity and precedence typically depend on the symbol but not on the output format, whereas the other attributes typically depend on the home theory and the output format.

```
<use fixity="prefix" bracket−style="math" precedence="p">
  <lbrack>⟪l⟫</lbrack>
  <rbrack>⟪r⟫</rbrack>
  <operator>⟪op⟫</operator>
  <separator>⟪s⟫</separator>
</use>
```

abbreviates

```
⟪op⟫
⟪l⟫
<map begin="1" end="−1">
  <separator>⟪s⟫</separator>
  <recurse precedence="p"/>
</map>
⟪r⟫
```

And

```
<use fixity="prefix" bracket−style="lisp" precedence="p">
  <lbrack>⟪l⟫</lbrack>
  <rbrack>⟪r⟫</rbrack>
  <operator>⟪op⟫</operator>
  <separator>⟪s⟫</separator>
</use>
```

abbreviates

```
⟪l⟫^b
⟪op⟫
⟪s⟫
<map begin="1" end="−1">
  <separator>⟪s⟫</separator>
  <recurse precedence="p"/>
</map>
⟪r⟫^b
```

where the superscript b denotes that the attribute `level="bracket"` has been added to the elements l and r. Note how in the mathematical bracket style brackets are always displayed, whereas they can be elided in the Lisp style[11]. Postfix operators are treated correspondingly. e(11)

If omitted `lbrack`, `rbrack`, and `separator` default to the empty elements[12], and `operator` defaults to `map begin="0"/>`. `fixity` defaults to `prefix`, `bracket-style` to `math`, and `precedence` to `0`. e(12)

Infix operators require some care: An infix operator which is applied to n arguments is inserted $n-1$ times between the arguments. `infixlr` is used for associative operators for which no inner brackets are needed. `infixl` and `infixr` refer to left and right-associative operators, respectively. In none of these cases do we generate inner brackets, not even elidable ones. This permits us to distinguish, e. g., the n-ary application of addition from successive application of binary addition.

Elidable or unelidable brackets are only generated when the several applications of the operator are nested. This is achieved by different input precedences when expanding. Associative, right-associative, and left-associative infix specifications expand to, respectively,

```
⟪l⟫^b
<map begin="1" end="−1">
  <separator>⟪op⟫</separator>
  <recurse precedence="p"/>
</map>
⟪r⟫^b
```

```
⟪l⟫^b
<map begin="1">
  <separator/>
  <recurse precedence="p − 1"/>
</map>
<map begin="2" end="−1">
  <separator/>
  ⟪op⟫
  <recurse precedence="p"/>
</map>
⟪r⟫^b
```

```
⟪l⟫^b
<map begin="1" end="−2">
  <separator/>
  <recurse precedence="p"/>
  ⟪op⟫
</map>
<map begin="−1">
  <separator/>
  <recurse precedence="p − 1"/>
</map>
⟪r⟫^b
```

5.3 Compatibility to ActiveMath

The ActiveMath system [Act07] uses a pattern-based approach that synthesizes ideas from the OMDoc 1.0 infrastructure and [NW01] for XSLT template generation.

```
<symbolpresentation xmlns="..." id="comb1bk" xref=".../bk">
  <notation format="html|pmathml|TeX" language="de">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <mrow><mo>(</mo><mfrac linethickness="0"><mi>n</mi><mi>k</mi></mfrac><mo>)</mo></mrow>
    </math>
    <OMOBJ xmlns="http://www.openmath.org/OpenMath">
      <OMA><OMS cd="comb1" name="bk"/><OMV name="n"><OMV name="k"></OMA>
    </OMOBJ>
  </notation>
</symbolpresentation>
```

[11] EdNote: @FR: Not in general! For postfix (e. g. factorial) and infix (most ordinary operators), yes, but for prefix? –CL

[12] EdNote: I thought in OMDoc 1.2 they defaulted to (and)? –CL

The presentation approach outlined in [MLUM05] directly uses the names of OpenMath variables and the content of MathML `mi` elements for the necessary cross-referencing between formats, confusing the role of meta-variables and object variables. Conceptually, this is a step backwards from [NW01] (where the distinction was somewhat tentative) and makes it impossible to exclude variables from the cross-referencing (and therefore recursion) of the presentation procedure. The main contribution here is the provision of the *jEditOQMath* [jEd07] developed by P. Libbrecht [Lib04] with a OpenMath Presentation Editor (OMPE) plug-in [OQM07] for the visual authoring of `symbolpresentation` elements. BegNP(13)

5.4 A Pattern-Based Approach to Flexary Mixfix Notations

Template-based approaches are often simpler to understand for authors, moreover, they are the only intuitive way to capture notations like $\sin^2(x)$ for the square of the value of the sine function at x. Here we propose to directly follow the lead of the Naylor&Watt system described in Section 1.3, re-using as much of the infrastructure we have built up until now as possible. Furthermore, we will drop the distinction between a semantic template and the notation generators. We claim that template-based notation definitions should only contain information about "corresponding" representations, and that presentation generation is only one direction of transformation between representations (from machine-oriented to human-oriented ones). The the symmetricity of the "correspondence relation" is immediately perspicuous if we have templates for more than one machine-oriented format, e. g. OpenMath and content MathML. But parsing of human-oriented formats should in principle be able to use the information in the "notation correpsondence" in the opposite of the "presentation direction". We should be able to derive parsing-oriented grammars taking into account precedences and the extended elision specifications developed above. In this paper though, we will not make use of this generalization and only concentrate on the presentation direction. Concretely, we assume that the presentation algorithm either directly uses the OpenMath template for pattern matching internally, or generates XPath expression for the `match` attribute in the `xsl:template` element as in Listing 1.

Finally, we keep the set of corresponding templates loosely coupled, rather than enging them in a `Notation` element as in Listing **??**: Two `presentation` elements are in correspondence, if they reference the same symbol in the `for` element and have the same `role` value.

Consider for instance the case of the typing judgment from Listing 3, here we would just add the following template.

```
<presentation format="OM" for="#typing−judgment">
  <OMA><OMS cd="types" name="typing−judgment"/>
    <arg name="context"/><arg name="sig"/><arg name="term"/><arg name="type"/>
  </OMA>
</presentation>
```

Unfortunately, we cannot directly write this in our grammar (nor would we like allow this, since we want to support an open-ended set of XML-based formats). Therefore we write instead

```
<presentation format="OM" role="application" for="#typing−judgment">
  <element name="OMA">
    <element name="OMS">
      <attribute name="cd"><text>types</text></attribute>
      <attribute name="name"><text>typing−judgment</text></attribute>
    </element>
    <arg name="context"/><arg name="sig"/><map name="term"/><map name="type"/>
  </element>
</presentation>
```

This template is assembled using the OMDoc `element` and `attribute` elements, rather than the OpenMath elements themselves. We consider this representation above to be equivalent to the one that uses the target elements literally.

[13]New Part: MK: we have to talk about this

Following XSLT terminology, we call we speak of the **native** encoding for the former representation and the **literal inclusion** encoding for the latter. Note that the native encoding has great advantages for validation, while the literal inclusion encoding is simpler to read for humans. They are obviously identical in expressivity and trivially isomorphic. We therefore assume that native form is used for storage and a pre-process or a tool like OMPE will present equivalent literal inclusion syntax for editing and user interaction. In particular, we will write our examples in literal inclusion encoding. P(14)

```
<presentation format="pmathml" for="#typing−judgment">
  <mrow>
    <arg name="context"/>
    <msub><mo>⊢</mo><arg name="sig"/></msub>
    <arg name="term"/>
    <mo>:</mo>
    <arg name="type"/>
  </mrow>
</presentation>
```

When for the same symbol there is a presentation pattern p_{in} for the given input format (e. g. OpenMath) and p_{out} for the desired output format (e. g. Presentation MathML), the input would be matched against p_{in}, assigning the meta-variables named by the `name` attribute. Then, the output would be rendered according to p_{out}, filling in the results of recursively rendering the values of the meta-variables at the indicated positions. P(14)

Note that the native encoding also clarifies the role of the `map` elements as meta-variables, and the contribution of the other presentation elements.

```
<presentation for="#plus" role="constant" format="OM"><OMS cd="arith1" name="plus"/></presentation>
<presentation for="#plus" role="application" format="OM">
  <OMA><OMS cd="arith1" name="plus"/><map begin="1" end="−1" name="summands"/></OMA>
</presentation>
<presentation for="#plus" role="application" format="ascii latex">
  <element name="mo" egroup="lbrack"><text>(</text></element>
  <map name="summands"><separator><arg pos="0"/></separator><recurse/></map>
  <element name="mo" egroup="lbrack"><text>(</text></element>
</presentation>
```

e(15) [15]

P(13)

[14]New Part: @MK: I only understood this stuff when you gave the additional PMML pattern and told me that one way of the presentation could be "pattern matching OM" → "output PMML". Therefore, I want an PMML example here and an explanation; please adapt this draft. –CL

[15]EdNote: MK: say something about the referencing with @name and @begin/end/pos, and where what is redundant. But @name is always a good documentation of the intended behavior.

6 Conclusion and Outlook

The advantage of content-oriented representation formats for mathematical formulae is that they are independent or a specific output format, and that human-oriented presentations can be generated taking into account user preferences, device constraints, etc. To realize this potential, we need presentation algorithms that are

knowledge-based The knowledge about notations and their use is an integral part of our mathematical expertise. If a presentation algorithm is not, then it becomes unmaintainable.

extensible Just as mathematics itself, mathematical notation is never finished; new notations are always invented, and they are introduced and discarded on the fly.

adaptive They should allow to take the user's preferences into account. In the end, the purpose of mathematical notation is to support a reader in understanding mathematics as effortlessly as possible — and people differ in their cognitive styles.

mathematical They should be able to model as many mathematical practices as possible, so that mathematicians are not hindered by system restrictions in the available notations.

efficient Presentation generation should not bog down servers, and should run on all kinds of clients with little lag. The time of the working mathematician is precious!

The first three of these aspects call for an meta-mathematical language for *notation definitions* which can be managed by MKM tools. A corollary of the manageability postulate is that the notation definition language should be *declarative*.

In this paper, we have presented an infrastructure for declarative notation definitions building on ideas from OMDoc1.2 and the presentation system of the Isabelle theorem prover. We have incorporated functionality for flexary applications and binders necessary to apply these ideas to OpenMath and content MathML, and we have extended the Isabelle's precedence-based elision algorithm with general flexible elision functionality, which covers most mathematical elision practices. We conjecture that others, like Church's dot notation can be specified by allowing more values for the `egroup` attribute.

Actually, the dot notation is a representative of a more general, user-adaptive practice that we do not cover in this paper: *abbreviation*. If a formula is too large or complex to be digested in one go, mathematicians often help their audience by abbreviating parts, which are explained in isolation or expanded when the general picture has been grasped. We feel that the abbreviation generation problem shares many aspects with elisions, but is less structured, and more dependent on modeling the abilities and cognitive load of the reader. Therefore we expect the elision techniques presented in this paper to constitute a first step into the right direction, but also that we need more insights to solve the abbreviation generation problem.

Another problem we have not addressed in this paper even though it is very related is the problem of reversing generation in the parsing process for mathematical formulae[16]: In some situations, the symbol characteristics are sufficient to allow a mathematical software system to reconstruct the content structure from its presentation. In the case of the Isabelle system, this property is essential to ensure unambiguous input. In this paper, we will not pursue the question of parseability, since the flexible elision of arguments is not sufficiently understood to warrant a restriction to an invertible presentation process. In fact, we have argued elsewhere[Koh04] that the process of parsing mathematical formula is AI-hard[4] in the worst case, since it pre-supposes a mathematical understanding of the communicated concepts. EdNote(16)

We have implemented the approach presented in this paper in a Scala class[17] to evaluate the EdNote(17)

[16]EdNote: I guess we did in the compatibility section. –CL

[4]i.e. we cannot solve the problem short of succeeding at Artificial Intelligence

[17]EdNote: @CL/FR: cite, is that correct?

expressivity and efficiency of our approach. First results are very promising; we will release the finished implementation under an open source license as a basis for practical presentation systems.

After a thorough evaluation we want to incorporate the presentation infrastructure presented in this paper into the OMDoc 2.0 format currently under development, and we propose it as part of the content dictionary mechanism of the upcoming MathML3 recommendation.

7 References

[ABC⁺03] Ron Ausbrooks, Stephen Buswell, David Carlisle, Stéphane Dalmas, Stan Devitt, Angel Diaz, Max Froumentin, Roger Hunter, Patrick Ion, Michael Kohlhase, Robert Miner, Nico Poppelier, Bruce Smith, Neil Soiffer, Robert Sutor, and Stephen Watt. Mathematical Markup Language (MathML) version 2.0 (second edition). W3C recommendation, World Wide Web Consortium, 2003. Available at `http://www.w3.org/TR/MathML2`.

[Act07] ActiveMath, seen February 2007. available at `http://www.activemath.org/`.

[BCC⁺04] Stephen Buswell, Olga Caprotti, David P. Carlisle, Michael C. Dewar, Marc Gaetano, and Michael Kohlhase. The Open Math standard, version 2.0. Technical report, The Open Math Society, 2004. `http://www.openmath.org/standard/om20`.

[Caj93] Florian Cajori. *A History of Mathematical Notations.* Courier Dover Publications, 1993. Originally published in 1929.

[jEd07] ***jEditOQMath***, seen May 2007. homepage at `http://www.activemath.org/projects/jEditOQMath/`.

[Kay06] Michael Kay. XSL Transformations (XSLT) Version 2.0. W3C Candidate Recommendation, World Wide Web Consortium (W3C), June 2006. Available at `http://www.w3.org/TR/2006/CR-xslt20-20060608/`.

[KMM07] Michael Kohlhase, Christine Müller, and Normen Müller. Documents with flexible notation contexts as interfaces to mathematical knowledge. In Paul Libbrecht, editor, *Mathematical User Interfaces Workshop 2007*, 2007.

[Koh00] Michael Kohlhase. OMDoc: Towards an openmath representation of mathematical documents. Seki Report SR-00-02, Fachbereich Informatik, Universität des Saarlandes, 2000. `http://www.mathweb.org/omdoc`.

[Koh04] Michael Kohlhase. Semantic markup for TEX/LATEX. In Paul Libbrecht, editor, *Mathematical User Interfaces*, 2004.

[Koh06] Michael Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2].* Number 4180 in LNAI. Springer Verlag, 2006.

[Lib04] P. Libbrecht. Authoring web content in activemath. In *Proceedings of the Second International Workshop on Authoring Adaptive and Adaptable Educational Hypermedia*, pages 445–460, 2004.

[MLUM05] Shahid Manzoor, Paul Libbrecht, Carsten Ullrich, and Erica Melis. Authoring Presentation for OpenMath. In Michael Kohlhase, editor, *Mathematical Knowledge Management, MKM'05*, number 3863 in LNAI, pages 33–48. Springer Verlag, 2005.

[NW01] Bill Naylor and Stephen M. Watt. Meta style sheets for the conversion of mathematical documents into multiple forms. In *Proceedings of the International Workshop on Mathematical Knowledge Management*, 2001.

[OQM07] ***OQMath***, seen May 2007. homepage at `http://www.activemath.org/projects/OQMath/`.

[Pau05] Lawrence C. Paulson. Isabelle reference manual. Technical report, Computer Laboratory, University of Cambridge, October 2005.

[Wol00] Stephen Wolfram. Mathematical notation: Past and future. In *MathML and Math on the Web: MathML International Conference*, Urbana Champaign, USA, October 2000. http://www.stephenwolfram.com/publications/talks/mathml/.

A A first RelaxNG Grammar

```
# A RelaxNG for Open Mathematical documents (OMDoc 2 alpha) Module PRES
# $Id: pres.rnc 6561 2007−06−24 06:30:57Z kohlhase $
# $HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc−1.2/projects/omdoc−2.0/OpenMath−paper/pres.rnc $
# See the documentation and examples at http://www.omdoc.org
# Copyright (c) 2007 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace omdoc = "http://www.mathweb.org/omdoc"

omdoc.class |=  presentation  |omstyle

 elision . attrib  = attribute egroup {xsd:anyURI}?, attribute elevel {xsd:integer}?
 crossref . attrib  = attribute crossref−symbol
                         {"no" | "yes" | "brackets" | "separator" | "lbrack" | "rbrack" | "all"}
format.attrib  = attribute format {xsd:string}, attribute requires {omdocref}?, xml.lang.attrib
precedence.attrib  = attribute precedence {xsd:string {pattern = "0|[1−9][0−9]∗"}}
role . attrib  = attribute role {"applied" | "binding" | "key"}

pres.items = elt  |  txt  |  recurse  |  map | keyval

presentation  = element presentation {omdoc.toplevel.attribs,
                                      format.attrib ,
                                      attribute for {omdocref},
                                      role . attrib ?,
                                      precedence.attrib ?,
                                      crossref . attrib ?,
                                      (pres.items)∗}

 elt  = element element {attribute name {xsd:NMTOKEN},
                         attribute crid {xsd:string}?,
                         attribute cr {"yes" |"no"}?,
                         attribute ns {xsd:anyURI}?,
                          elision . attrib ,
                         (attrb  |  pres.items)∗}

attrb  = element attribute {attribute name {xsd:NMTOKEN},
                            attribute ns {xsd:anyURI}?,
                            text}

map = element map {attribute begin {xsd:integer},
                   attribute end {xsd:integer},
                   precedence.attrib ?,
                    elision . attrib ?,
                   lbrack?,rbrack?,(pres.items)∗}

recurse  = element recurse {attribute precedence {xsd:integer}?,
                             elision . attrib}
keyval  = element keyval {attribute cd {xsd:NMTOKEN},
                          attribute name {xsd:NMTOKEN},
                           elision . attrib}

txt  = element text {elision.attrib,text}
```

B Complex Examples

```
<!−− Multiple integral example −−>
<OMOBJ version="3.0">
  <OMBIND>
    <OMA><OMS cd="newint" name="int"/><OMV name="S1"/><OMV name="S2"/><OMV name="S3"/></OMA>
    <OMBVAR><OMV name="x1"/><OMV name="x2"/><OMV name="x3"/></OMBVAR>
    <OMC>
      <OMA>
        <OMS cd="logic1" name="and"/>
        <OMA><OMS cd="relation1" name="gt"/><OMV name="x1"/><OMI>1</OMI></OMA>
        <OMA><OMS cd="relation1" name="gt"/><OMV name="x2"/><OMI>1</OMI></OMA>
        <OMA><OMS cd="relation1" name="gt"/><OMV name="x3"/><OMI>1</OMI></OMA>
      </OMA>
    </OMC>
    <OMA>
      <OMS cd="arith1" name="plus"/>
      <OMV name="x1"/>
      <OMA><OMS cd="arith1" name="times"/>
        <OMI>2</OMI>
        <OMA><OMS cd="arith1" name="power"/><OMV name="x2"/><OMI>2</OMI></OMA>
      </OMA>
      <OMA><OMS cd="arith1" name="times"/>
        <OMI>3</OMI>
        <OMA><OMS cd="arith1" name="power"/><OMV name="x3"/><OMI>3</OMI></OMA>
      </OMA>
    </OMA>
  </OMBIND>
</OMOBJ>
```

```
<omdoc>
<presentation for="newint" role="applied" class="multi">
  <map begin="1" end="−1">
    <element name="frac">
      <attribute name='linethickness'>
        <text>0</text>
      </attribute>
      <element name="mo">&int;</element>
      <recurse/>
    </element>
  </map>
</presentation>
<presentation for="newint" role="binder" class="multi">
  <element name="frac">
    <attribute name='linethickness'>
      <text>0</text>
    </attribute>
    <arg pos="0"/>
    <arg pos="−2"/>
  </element>
  <arg pos="−1"/>
  <map begin="−3" end="1">
    <text>d</text>
    <recurse/>
  </map>
</presentation>
<presentation for="newint" role="applied" class="mono">
  <element name="frac">
    <attribute name='linethickness'>
      <text>0</text>
    </attribute>
    <element name="mo">&int;</element>
    <map begin="1" end="−1">
      <separator>
        <element name="mo">&times;</element>
      </separator>
      <recurse/>
    </map>
  </element>
</presentation>
<presentation for="newint" role="binder" class="mono">
  <element name="frac">
    <attribute name='linethickness'>
      <text>0</text>
    </attribute>
    <arg pos="0"/>
    <arg pos="−2"/>
  </element>
  <arg pos="−1"/>
  <map begin="1" end="−3">
    <text>d</text>
    <recurse/>
  </map>
</presentation>
</omdoc>
```

makes $\genfrac{}{}{0pt}{}{\int_{S1}\int_{S2}\int_{S2}}{x_1>1\wedge x_2>1\wedge x_3>1} x_1 + 2x_2^2 + 3x_3^3 dx_3 dx_2 dx_1$