Notations for Active Mathematical Documents

Michael Kohlhase, Christoph Lange, Christine Müller, Normen Müller, and Florian Rabe

Abstract. Notations are central for understanding mathematical discourse. Readers would like to read notations that transport the meaning well and prefer notations that are familiar to them. Therefore, authors optimize the choice of notations with respect to these two criteria, while at the same time trying to remain consistent throughout the document and with their own prior publications. In print media where notations are fixed at publication time, this is an over-constrained problem. In active documents notations can be adapted at reading time, taking reader preferences into account.

We present a representational infrastructure for notations in active mathematical documents. Mathematical notations can be defined declaratively. Author and reader can extensionally define the set of available notation definitions at arbitrary document levels, and they can guide the notation selection function via intensional annotations. We discuss the management of these declarative specifications with focus on the creation and maintenance of notations. Finally, we describe several interconnected implementations of the various notation-related aspects of active documents.

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). Structural properties of operators often result in special presentations, e.g., the scope of a radical expression is visualized by the length of its bar. Their mathematical properties give rise to placement (e.g., associative arithmetic operators are written infix), and their relative importance is expressed in terms of binding strength conventions for brackets. 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. All of these make it difficult to determine the functional structure of an expression from its presentation.

Content Markup formats for mathematics, such as OpenMath [BCC+04] and content 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. Many such presentation processes have been proposed, and all have their strengths and weaknesses. In this article, we conceptualize the presentation of mathematical formulae as consisting of three components: the context-dependent *selection* of appropriate rules ("notation definitions") for presentation, the two-dimensional *composition* or visual sub-presentations to larger ones, and the *interaction* with rendered content.

Most current presentation processes concentrate on the relatively well-understood composition aspect. They control the notation selection by relying on simple metadata, which do not fully represent the context of a formula. Their output is mostly static and meant to be read, not to be interacted with, except for some applications that allow for changing notations afterwards.

In this situation, we propose to encode the presentational characteristics of symbols (their context, their compositional behavior, and information accessible by interactive services) declaratively in **notation definitions**, which are part of the representational infrastructure and consist of "prototypes" (patterns that are matched against content representation trees) and "renderings" (that are used to construct the corresponding presentational trees). Note that since we have reified the notations, we can now devise a flexible management process for notations. For example, we can capture the notation preferences of authors, aggregators and readers and adapt documents to these. We propose an elaborated mechanism to collect notations from various sources and specify notation preferences. This brings the separation of *function* from *form* in mathematical objects and assertions in MKM formats to fruition on the document level. This is especially pronounced in the context of dynamic presentation media (e.g., on the screen), we can now realize "*active documents*", where we can interact with a document directly, e.g., instantiating a formula with concrete values or graphing a function to explore it or "*living/evolving documents*" which monitor the change of knowledge about a topic and adapt to a user's notation preferences consistently.

2. Understanding Notations

Mathematical notations denote mathematical concepts, i. e., the objects we talk and write about when we do mathematics: Rather simple objects like numbers, functions, triangles, matrices, and more complex ones such as vector spaces and infinite series. Mathematical notations are *no separate entities* but *highly interdependent*. In mathematics we speak of *notation systems*, i. e. collections of notations that depend on each other. Consequently, the choice of a specific notation for a

concept requires to use notations from the same system for all other concepts. For example, if we look at the notation for *subset* and *proper subset*, we can use $\subseteq$ and $\subset$ versus $\subset$ and $\subsetneq$. In the first combination, $\subset$ denotes the *proper subset*, while in the second combination it denotes *subset*. Consequently, when adapting the notation of subset from $\subseteq$ to $\subset$, we need to also change the notation for proper subset from $\subset$ to $\subsetneq$. Otherwise, we end up with the same notation for two different mathematical concepts, which eventually destroys the semantics of the mathematical formula.

Mathematical Communities and their Notations. We observe *mathematical communities*, which prefer different *notation systems*: For example, if we look at a *Russian* and *Western* mathematical journal we will find that two different notation systems are used. Partly, the notations between Russian and Western researchers differ as they build on different concepts. However, also the *overlapping concepts* used by both groups are denoted with very different sets of notations. Moreover, even if Western researchers used and defined concepts solely used by Russians, they would denote them very differently staying conform to the type of notations in their systems. Figure 1 provides an example of two notation systems in the area of *sentential logic*: On the right we see the core of Jan Łukasiewicz's notation for sentential logic [Łuk67], to the left we see the "conventional" notation, which was developed in the 1970s and 80s.

Mathematical Concept	**Conventional Notation**	**Polish Notation**
Negation	$\neg\varphi$	$N\varphi$
Conjunction	$\varphi \wedge \psi$	$K\varphi\psi$
Disjunction	$\varphi \vee \psi$	$A\varphi\psi$
Material conditional	$\varphi \rightarrow \psi$	$C\varphi\psi$
Biconditional	$\varphi \leftrightarrow \psi$	$E\varphi\psi$
Sheffer stroke	$\varphi \mid \psi$	$D\varphi\psi$
Possibility	$\Diamond\varphi$	$M\varphi$
Necessity	$\Box\varphi$	$L\varphi$
Universal Quantifier	$\forall\varphi$	$\Pi\varphi$
Existential Quantifier	$\exists\varphi$	$\Sigma\varphi$

Table 1. The Conventional and Polish Notation Systems

Mathematical areas are further divided into different *schools* that originally evolved based on individual styles of single mathematicians. For example, Calude/ Chaitin [Cha87] and Li/Vitani [LV97] use different notations to denote the same concepts (plain and prefix free complexity) in the field of *algorithmic information theory* (AIT): Chatin/Calude use $K(x)$ and $H(x)$, while Li/Vitany use $C(x)$ and $K(x)$.

Individual Styles. We can also observe individual author styles that differ within schools or communities. For example, some mathematical authors are more *formal*, while others prefer to include more *natural language terms*. For example, consider

the mathematical statement “Let n equal 2 times m square. Choose a natural number k so that k is less than or equal to n.” in constrast to the more compact and formal “Let $n = 2m^2$. Choose a number $k \leq n$.”. Some mathematicians feel that the latter is more *easier to read*, while others reject it, as they believe that symbols should not be part of the *prose text*. Consequently, mathematicians tend to prefer a specific *notation system* and *style* and often *reject* material with notations that differ to their own. In particular, different notation systems cause problems for the integration of (online) course materials from different authors as they cause inconsistencies and a tedious refactoring of the combined material.

2.1. Representation of Mathematical Notations

To provide automated services such as the adaptation of mathematical notations, we represent mathematical objects in formats like MathML [W3C03] a W3C recommended format for high-quality presentation of mathematical formulas on the Web or OpenMath [Ope07], a content-oriented format that concentrates on the meaning of objects[1].

OpenMath Representation

```
<om:OMOBJ>
  <om:OMA>
    <om:OMS cd="combinat1"
            name="binomial"/>
    <om:OMV name="n"/>
    <om:OMV name="k"/>
  </om:OMA>
</om:OMOBJ>
```

MathML Representation

```
<m:mrow>
  <m:mo fence="true">(</m:mo>
  <m:mfrac linethickness="0">
    <m:mi>n</m:mi>
    <m:mi>k</m:mi>
  </m:mfrac>
  <m:mo fence="true">)</m:mo>
</m:mrow>
```

Presentation

$\binom{n}{k}$

Figure 1. OpenMath and MathML representation of the binomial coefficient.

Figure 1 provides the OpenMath and MathML representations of $\binom{n}{k} = \frac{n!}{k!(n-k)!}$, the number of k-element subsets of an n-element set. The OpenMath expression on the left captures the functional structure of the expression by representing it as the application (using the `OMA` element) of the “binomial coefficient” function (represented by an `OMS` element) applied to two variables (`OMV`). Note that the `cd` and `name` attributes characterize the binomial function by pointing to a definition in a **content dictionary** (CD) [OMC08], a specialized document that specifies *commonly agreed* definitions of basic mathematical objects and allows machines to distinguish the meaning of included mathematical objects. In contrast to this, the presentation MathML expression in the middle marks up the

[1]In fact MathML has a sub-language that is equivalent to OpenMath, but we will concentrate on the presentational functionality of MathML for simplicity. Similarly, we will use OpenMath synonymously with “content markup”.

appearance of the formal when displayed visually (or read out aloud for vision-impaired readers): The formula is represented as a horizontal row (`mrow`) of two stretchy bracket operators (`mo`) with a special layout for fractions (`mfrac`, where the line is made invisible by giving it zero thickness) where the numerator and denominator are mathematical identifiers (`mi`). The aim and strengths of the two formats are complementary: OpenMath expressions are well-suited for information retrieval by functional structure and computation services, while MathML is used for display: MathML-aware browsers will present the middle expression in Figure 1 as $\binom{n}{k}$.

```
<m:semantics>
  <m:mrow id="top">
    <m:mo>(</m:mo>
    <m:mfrac linethickness="0">
      <m:mi id="left">n</m:mi>
      <m:mi id="right>k</m:mi>
    </m:mfrac>
    <m:mo>)</m:mo>
  </m:mrow>
  ...
  <m:annotation-xml>
    <om:OMOBJ>
      <om:OMA xref="top">
        <om:OMS cd="combinat1"
                name="binomial" />
        <om:OMV name="n" xref="left"/>
        <om:OMV name="k" xref="right"/>
      </om:OMA>
    </om:OMOBJ>
  </m:annotation-xml>
</m:semantics>
```

Figure 2. Parallel Markup: Combining OpenMath and MathML

In order to combine both markup aspects, MathML allows *parallel markup* [W3C03] with fine-grained cross-references of corresponding sub-expressions. Figure 2 provides the parallel markup for the example in Figure 1: The `semantics` element embeds a presentation MathML expression and an `annotation-xml` with the respective OpenMath expression. The `id` and `xref` attributes specify corresponding subterms. An application of this would be that a user can select a subterm in the presentation MathML rendered in a browser, so that a context menu option could send the corresponding OpenMath sub-expression to e. g. a computer algebra system for evaluation, simplification or graphing.

Parallel markup only provides a one-to-one mapping between OpenMath and MathML expressions. However, in mathematics with have to deal with multiple alternative notations that denote the same mathematical object, such as various presentations of the binomial coefficient C^k_n, C^n_k, or $\binom{n}{k}$. Moreover, we can also select a MathML representation that is more suited for further processing, such as by *braille* or *screen readers*. While the former reader supports visually impaired users, the latter is of use to any learner as it supports to *read* notations *out loud* and thus to foster the user's understanding. Figure 3 provides two alternative MathML representations for the binomial coefficient $\binom{n}{k}$. The expression to the right is also referred to as *canonical representation* [AM06] and can more easily be accessed by braille readers: A mathematical braille translator, such as [ASFM07], will recognize the MathML expression to the left as fraction $frac(n, k)$, since presentation attributes such as `linethickness` are ignored.

One could argue that braille translators should not rely on MathML (but rather OpenMath), as MathML only provides a layout tree and no insights on the semantics of the notation. We want to support existing implementations and thus provide an adaptable selection between alternative MathML expressions tailored to the needs and preferences of the users as well as further processing services.

```
<m:mrow>
  <m:mo>(</m:mo>
  <m:mfrac linethickness="0">
    <m:mi>n</m:mi>
    <m:mi>k</m:mi>
  </m:mfrac>
  <m:mo>)</m:mo>
</m:mrow>
```

```
<m:mrow>
 <m:mrow><m:mo>(</m:mo>
  <m:mrow>
   <m:mtable>
    <m:mtr><m:mtd><m:mi>n</m:mi></m:mtd></m:mtr>
    <m:mtr><m:mtd><m:mi>k</m:mi></m:mtd></m:mtr>
   </m:mtable>
  </m:mrow>
 <m:mo>)</m:mo></m:mrow>
</m:mrow>
```

Figure 3. Two Valid MathML Expressions.

In order to automatically adapt mathematical notations, we need to be able to vary the displayed presentation MathML. This is usually done by parameterizing the process by which presentations are generated from a given content representation. In our approach, we represent the *mappings* between an OpenMath expression and all alternative MathML representations. Conceptually, these mappings represent *mathematical notation practices* as they explicate the *choice of mathematical notations* of the user. In order to make adaptation *practice-* and *context-aware*, we need to provide a conversion workflow that takes notation practices and concrete context as input and adapts the respective notation for the user. Before we present our framework, we will review the state of the art.

2.2. Related Work

Naylor, Smirnova, and Watt [NW01, SW06b, SW06a] present an approach based on meta stylesheets that utilizes a MathML-based markup of arbitrary notations in terms of their content and presentation. Based on the manual selection of users, it generates user-specific XSLT style sheets [Kay06] for the adaptation of documents. Naylor and Watt [NW01] introduce a one-dimensional context annotation of content expressions to intensionally select an appropriate notation specification. The authors claim that users also want to delegate the styling decision to some defaulting mechanism and propose the following hierarchy of default notation specification (from high to low): command line control, input documents defaults, meta stylesheets defaults, and content dictionary defaults.

In [MLUM05], Manzoor et al. emphasize the need for maintaining uniform and appropriate notations in collaborative environments, in which various authors contribute mathematical material. They address the problem by providing authors with respective tools for editing notations as well as by developing a framework for a consistent presentation of symbols. In particular, they extend the approach

of Naylor and Watt by an explicit language markup of the content expression. Moreover, the authors propose the following prioritization of different notation styles (from high to low): individual style, group, book, author or collection, and system defaults.

In [KLR07] we have revised and improved the presentation specification of OMDoc1.2 [Koh06] by allowing a static well-formedness, i. e., the well-formedness of presentation specifications can be verified when writing the presentations rather than when presenting a document. We also addressed the issue of flexible elision. However, the approach does not facilitate to specify notations, which are not local tree transformations of the semantic markup.

In [KMM07] we initiated the redefinition of *documents* towards a more *dynamic* and *living* view. We explicated the narrative and content layer and extended the *document model* by a third dimension, i. e., the *presentation layer*. We proposed the *extensional* markup of the *notation context* of a document, which facilitates users to explicitly select suitable notations for document fragments. These extensional collections of notations can be inherited, extended, reused, and shared among users. For the system presented in this article, we have re-engineered and extended the latter two proposals.

[PZ06] provide a bidirectional conversion betwen a presentation-oriented encoding and a fully formal representation of their semantics. In consequence, the presented work includes research on disambiguating presentation markup during parsing, which will not be consider in this article. The authors describe an abstract syntax for notation as well as a rendering algorithms, both comparably powerful to our approach. However, the application of their work is an interactive theorem prover, while our work focuses on the management of mathematical notations on the web.

3. Defining Notations

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

3.1. Syntax of Notation Definitions

We will now present an abstract version of the presentation starting from the observation that in content markup formalisms for mathematics formulae are represented as "formula trees". Concretely, we will concentrate on OpenMath objects, the conceptual data model of OpenMath representations, since it is sufficiently

general, and work is currently under way to unify the semantics of MathML with the one of OpenMath. Furthermore, we observe that the target of the presentation process is also a tree expression: a layout tree made of layout primitives and glyphs, e. g., a presentation MathML or LaTeX expression.

To specify notation definitions, we use the one given by the abstract grammar from Figure 2. Here $|$, $[-]$, $-^*$, and $-^+$ denote alternative, bracketing, and non-empty and possibly empty repetition, respectively. The non-terminal symbol ω is used for patterns φ that do not contain jokers. Throughout this article, we will use the non-terminal symbols of the grammar as meta-variables for objects of the respective syntactic class.

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

Table 2. The Grammar for Notation Definitions

Intuitions. The intuitive meaning of a *notation definition* $ntn = \varphi_1, \ldots, \varphi_r \vdash (\lambda_1 : \rho_1)^{p_1}, \ldots, (\lambda_s : \rho_s)^{p_s}$ is the following: If an object matches one of the patterns φ_i, it is rendered by one of the renderings ρ_i. Which rendering is chosen, depends

on the active rendering context, which is matched against the context annotations λ_i; context annotations are usually lists of key-value pairs and their precise syntax is given in Section 4.2. The integer values p_i give the output precedences of the renderings, which are used to dynamically determine the placement of brackets.

The *patterns* φ_i are formed from a formal grammar for a subset of OpenMath objects extended with named jokers. The jokers $\underline{o}[\varphi]$ and $\underline{l}(\varphi)$ correspond to \(φ\) (or \(.\) if φ is omitted) and \(φ\)$^+$ in Posix regular expression syntax ([POS88]) – except that our patterns are matched against the list of children of an OpenMath object instead of against a list of characters. Here underlined variables denote names of jokers that can be referred to in the renderings ρ_i. We need two special jokers $\underline{s}$ and $\underline{v}$, which only match OpenMath symbols and variables, respectively. The *renderings* ρ_i are formed by a formal syntax for simplified XML extended with means to refer to the jokers used in the patterns. When referring to object jokers, input precedences are given that work together with the output precedences of renderings.

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

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

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

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

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

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

We can do that with the following notations:

$$@(int, \underline{\texttt{ranges}}(@(iv, \underline{\texttt{a}}, \underline{\texttt{b}})), \beta(lam, \underline{\texttt{vars}}(\underline{\texttt{x}}), \underline{\texttt{f}}))$$
$$\vdash\ ((format = latex):$$
$$\texttt{for}(\underline{\texttt{ranges}})\{\backslash\texttt{int_}\{\ \underline{\texttt{a}}^{\infty}\ \}\hat{}\{\ \underline{\texttt{b}}^{\infty}\ \}\}\ \underline{\texttt{f}}^{\infty}\ \texttt{for}(\underline{\texttt{vars}}, -1)\{\texttt{d}\ \underline{\texttt{x}}^{\infty}\})^{-\infty}$$

$$\alpha(\underline{a}, color \mapsto \underline{\texttt{col}})\ \vdash\ ((format = latex):\ \{\backslash color\{\ \underline{\texttt{col}}\ \}\ \underline{\texttt{a}}^{\infty}\ \})^{-\infty}$$

$$@(plus, \underline{\texttt{args}}(\underline{\texttt{arg}}))\ \vdash\ ((format = latex):\ \texttt{for}(\underline{\texttt{args}}, +)\{\underline{\texttt{arg}}\})^{10}$$

$$@(sin, \underline{\texttt{arg}})\ \vdash\ ((format = latex):\ \backslash sin\ \underline{\texttt{arg}})^{0}$$

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

The output precedence $-\infty$ of *int* makes sure that the integral as a whole is never bracketed. And the input precedences ∞ makes sure that the arguments of *int* are never bracketed. Both are reasonable because the integral notation provides its own fencing symbols, namely $\int$ and d. The output precedences of *plus* and *sin* are 10 and 0, which means that *sin* binds stronger; therefore, the expression $\sin x$ is not bracketed either. However, an inexperienced user may wish to display these brackets. Therefore, our rendering does not completely suppress them. Rather, we annotate them with an "elision level", which is computed as the difference of the two precedences. This information can then be used by active documents (see section 6.2).

Well-formed Notations. A notation definition $\varphi_1, \ldots, \varphi_r \vdash (\lambda_1 : \rho_1)^{p_1}, \ldots, (\lambda_s : \rho_s)^{p_s}$ is well-formed if all φ_i are well-formed patterns that induce the same empty match contexts, and all ρ_i are well-formed renderings with respect to that empty match context.

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

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

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

$$\texttt{ranges} \mapsto (_),\ \texttt{ranges/a} \mapsto (_),\ \texttt{ranges/b} \mapsto (_),\ \texttt{f} \mapsto _,$$

$$\texttt{vars} \mapsto (_),\ \texttt{vars/x} \mapsto (\text{``''})$$

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

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

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

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

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

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

3.2. Semantics of Notation Definitions

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

1. In a preprocessing step, *Doc* is scanned and notation definitions given inside *Doc* are collected. If *Doc* imports or includes other documents, these are retrieved and processed recursively. And if *Doc* references external notation definitions, these are retrieved as well. Together with the notations already present in the database, these notations form the notation context Π.

2. In a second preprocessing step, Π is normalized by grouping together renderings of the same patterns. Then for each pattern, the triples $(\lambda\colon \rho)^p$ pertaining to it are filtered and ordered according to how well λ matches Λ. This process can also scan for and store arbitrary other information in *Doc* or in *DB*: For example, *Doc* can contain what we call notation tags (see Section 4.1) that explicitly relate an OpenMath object and a rendering.
3. *Doc* is traversed, and the low-level algorithm is invoked on every OpenMath object found.

The preprocessing steps are described in detail in Sec. 4. In the following, we describe the low-level algorithm.

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

It computes its output in two steps.

1. ω is matched against the patterns in the notation definitions in Π until a matching pattern φ is found. The notation definition in which φ occurs induces a list $(\lambda_1\colon \rho_1)^{p_1}, \ldots, (\lambda_n\colon \rho_n)^{p_n}$ of context-annotations, renderings, and output precedences. The first one of them is chosen unless Π contains a notation tag that selects a different one for ω.
2. The output is ${\rho_j}^{M(\varphi,\omega)}$, the rendering of ρ_j in context $M(\varphi,\omega)$ as defined below. Additionally, if $p_j > p$, the output is enclosed in brackets.

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

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

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

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

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

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

- $\langle S\rangle\rho_1 \ \ldots \ \rho_r\langle/\rangle$ is rendered as an XML element with name S. The attributes are those ${\rho_i}^M$ that are rendered as attributes. The children are the concatenation of the remaining ${\rho_i}^M$ preserving their order.
- $S =$ "$\rho_1 \ \ldots \ \rho_r$" is rendered as an attribute with label S and value ${\rho_1}^M + \ldots + {\rho_n}^M$ (which has type text due to the well-formedness).
- S is rendered as the text S.
- $\underline{s}$ and $\underline{v}$ are rendered as the text $M(s)$ or $M(v)$, respectively.
- $\underline{o}^p$ is rendered by applying the rendering algorithm recursively to $M(o)$ and p.
- $\mathtt{for}(\underline{l}, I, \rho_1 \ \ldots \ \rho_r)\{\rho'_1 \ \ldots \ \rho'_s\}$ is rendered by the following algorithm:
  1. Let $sep := {\rho_1}^M + \ldots + {\rho_r}^M$ and t be the length of $M(l)$.
  2. For $i = 1, \ldots, t$, let $R_i := {\rho'_1}^{M_i^l} + \ldots + {\rho'_s}^{M_i^l}$.
  3. If $I = 0$, return nothing and stop. If I is negative, reverse the list R, and invert the sign of I.
  4. Return $R_I + sep + R_{2*I} \ldots + sep + R_T$ where T is the greatest multiple of I smaller than or equal to t.

Here the match context M_i^l arises from M as follows

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

Example. Consider the example introduced in Section 3.1. There we have

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

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

$$\mathtt{ranges} \mapsto (@(iv, a_1, b_1), \ldots, @(iv, a_n, b_n)),\ \mathtt{ranges/a} \mapsto (a_1, \ldots, a_n),$$
$$\mathtt{ranges/b} \mapsto (b_1, \ldots, b_n),\ \mathtt{f} \mapsto \alpha(@(plus, @(sin, \upsilon(x_1)), \upsilon(x_2)), color \mapsto red),$$
$$\mathtt{vars} \mapsto (\upsilon(x_1), \ldots, \upsilon(x_n)),\ \mathtt{vars/x} \mapsto (x_1, \ldots, x_n)$$

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

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

To render ρ in match context M, we have to render the three components and concatenate the results. Only the iterations are interesting. In both iterations,

the separator *sep* is empty; in the second case, the step size I is -1 to render the variables in reverse order.

4. Selecting Notations

After defining syntax and semantics of our notations, we will now discuss options for configuring the rendering algorithm by extending it with the ability to dynamically choose the proper notation.

4.1. Extensional Selection of Notations

In Section 3, we have seen how collections of notation definitions induce rendering functions. Now we permit users to define the set Π of available notations extensionally. In the following, we discuss the collection of notations from various sources and the construction of Π_ω for a concrete mathematical object ω.

Collecting Notation Definitions. The algorithm for the collection of notations takes as input a tree-structured document, e. g., an XML document, an object ω within this document, and a totally ordered set $\mathcal{S}^N$ of source names. Based on the hierarchy proposed in [NW01], we use the source names EC, F, Doc, CD, SD, and T explained below. The user can change their priorities by ordering them.

The collection algorithm consists of two steps: The collection of notation definitions and their reorganization. In the first step the notation definitions are collected from the input sources according to the order in $\mathcal{S}^N$. The respective input sources are treated as follows:

- EC denotes the **extensional context**, which associates a list of renderings, notation definitions, or containers of notation definitions to every node of the input document. The effective extensional context is computed according to the position of ω in the input document (see a concrete example below). EC is used by authors to reference their individual notation context.
- F denotes an **external notation document** from which notation definitions are collected. F can be used to overwrite the author's extensional context declarations.
- Doc denotes the **input document**. As an alternative to EC, Doc permits authors to embed notation definitions into the input document.
- CD denotes the **content dictionaries** defining symbols occurring in ω. These are searched in the order in which the symbols occur in ω. Content dictionaries may include or reference default notation definitions for their symbols.
- SD denotes the **system default** notation document, which typically occurs last in $\mathcal{S}^N$ as a fallback if no other notation definitions are given.
- T denotes **notation tags**, which provide the same functionality as the EC option without changing the input document. They allow readers full control of the eventually selected rendering elements as they explicitly point to the appropriate notations. Please note that T has to be used in combination

with either F (provided in an external file) or Doc (embedded in the input document).

In the second step the obtained notation context Π is reorganized: All occurrences of a pattern φ in notation definitions in Π are merged into a single notation definition. All EC and T references to rendering elements are applied to filter and prioritize the $(\lambda\colon \rho)^p$ pairs (see a concrete example below).

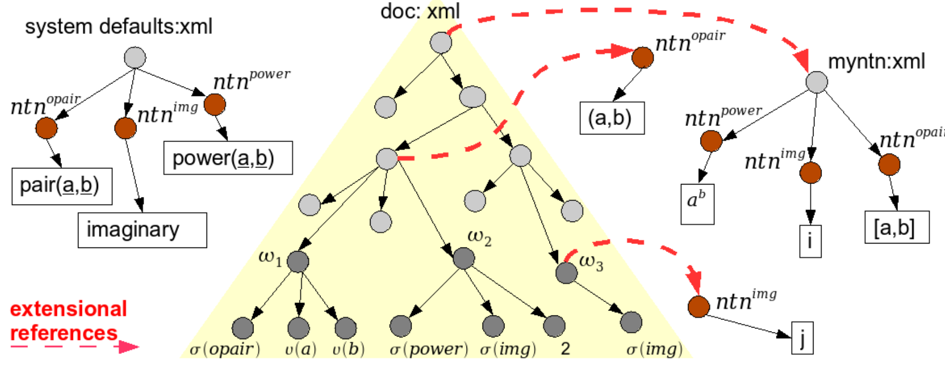

FIGURE 4. Collection Example

We base our further illustration on the input document in Fig. 4, which includes three mathematical objects. For simplicity, we omit the cdbase and cd attributes of symbols.

$$\omega_1 : @(\sigma(opair), \upsilon(a), \upsilon(b)) \leadsto (a,b) \qquad \omega_2 : @(\sigma(power), \sigma(img), 2) \leadsto i^2 \qquad \omega_3 : \sigma(img) \leadsto j$$

The dashed arrows in the figure represent extensional references: For example, the ec attribute of the document root *doc* references the notation document "*myntn*", which is interpreted as a container of notation definitions.

We apply the algorithm above with the input object ω_3 and $\mathcal{S}^N = (EC, SD)$ and receive Π_{ω_3} in return. For simplicity, we do not display context annotations and precedences.

---

1. We collect all notation definitions yielding Π_{ω_3}
1.1 We collect notation definitions from EC
1.1.1 We compute the effective extensional context based on the position of ω_3 in the input document: $ec(\omega_3) = (ntn^{img}, myntn)$
1.1.2 We collect all notation definitions based on the references in $ec(\omega_3)$:
$\Pi_{\omega_3} = (ntn^{img}, ntn^{power}, ntn^{img}, ntn^{opair})$
1.2. We collect notation definitions from SD and append them to Π_{ω_3}
$\Pi_{\omega_3} = (ntn^{img}, ntn^{power}, ntn^{img}, ntn^{opair}, ntn^{opair}, ntn^{img}, ntn^{power})$
1.3. The collected notation definition form the notation context Π_{ω_3}
$\Pi_{\omega_3} = (\ \varphi_1 \vdash j, \varphi_2 \vdash \underline{a}^{\underline{b}}, \varphi_1 \vdash i, \varphi_3 \vdash [\underline{a}, \underline{b}], \varphi_3 \vdash pair(\underline{a}, \underline{b}), \varphi_1 \vdash imaginary,$

$\varphi_2 \vdash power(\underline{a}, \underline{b})$)
2. We reorganize Π_{ω_3} yielding Π'_{ω_3}
$\Pi'_{\omega_3} = (\ \varphi_1 \vdash j, i, imaginary; \varphi_2 \vdash \underline{a}^{\underline{b}}, power(\underline{a}, \underline{b}); \varphi_3 \vdash [\underline{a}, \underline{b}], pair(\underline{a}, \underline{b})\ )$

To implement EC in arbitrary XML-based document formats, we propose an `ec` attribute in a namespace for notation definitions, which may occur on any element. The value of the `ec` attribute is a whitespace-separated list of URIs of either rendering elements, notation definitions, or any other document. The latter is interpreted as a container, from which notation definitions are collected. The `ec` attribute is empty by default. When computing the effective extensional context of an element, the values of the `ec` attributes of itself and all parents are concatenated, starting with the innermost.

Discussion of Collection Strategies. In the following, we illustrate the advantages and drawbacks for collecting notations from EC, F, Doc, CD, SD, or T.

1. Authors can write documents which only include content markup and do not need to provide any notation definitions. The notation definitions are then collected from CD and SD.
2. The external document F permits authors to store their notation definitions centrally, facilitating the maintenance of notational preferences. However, authors may not specify alternative notations for the same symbol on granular document levels.
3. Authors may use the content dictionary defaults or overwrite them by providing F or Doc.
4. Authors may embed notation definitions inside their documents (Doc). However, this causes redundancy inside the document and complicates the maintenance of notation definitions.
5. Users can overwrite the specification inside the document with F. However, that can destroy the meaning of the text, since the granular notation contexts of the authors are replaced by only one alternative declaration in F.
6. Collecting notation definitions from F or Doc has benefits and drawbacks. Since users want to easily maintain and change notation definitions but also use alternative notations on granular document levels, we provide EC. This permits a more controlled and more granular specification of notations.
7. Besides the T option, users cannot change the granular extensional declarations without modifying the input document. They can only overwrite the author's granular specifications with their individual styles F or default specification CD, which may reduce the understandability of the document.

Besides T and EC, none of the extensional notation context declarations does support users to select between alternative renderings inside one notation definition. Moreover, attached to the output expression, T and EC also support to capture, which rendering elements have been applied during the conversion. This is particularly useful for interactive features (see Section 6) as well as implicit user modeling (see [MK08] for details).

4.2. Choosing Renderings Intensionally

Extensional reference enforce users to explicate the notation choice. They provide authors and readers with full control of the applied rendering elements, but also require additional efforts. In contrast, a *context-sensitive* selection of renderings lets users guide the selection of alternative renderings with less efforts; instead of providing explict rendering references users only need to describe the context properties of the intendent output. We use an intensional rendering context Λ, which is matched against the context annotations in the notation definitions. In the following, we discuss the collection of contextual information from various sources and the construction of Λ_ω for a concrete mathematical object ω.

Collecting Contextual Information. We represent contextual information by contextual key-value pairs, denoted by $(d_i = v_i)$. The key represents a *context dimension*, such as *language*, *level of expertise*, *area of application*, or *individual preference*. The value represents a *context value* for a specific context dimension. The algorithm for the context-sensitive selection takes as input an object ω, a list L of elements of the form $(\lambda\colon \rho)^p$, and a totally ordered set $\mathcal{S}^C$ of source names. We allow the names GC, CCF, IC, and MD. The algorithm returns a pair (ρ, p).

The selection algorithm consists of two steps: The collection of contextual information Λ_ω and the selection of a rendering. In the first step Λ_ω is computed by processing the input sources in the order given by $\mathcal{S}^C$. The respective input sources are treated as follows:

- GC denotes the **global context** which provides contextual information during *rendering time* and overwrites the author's intensional context declarations. The respective $(d_i = v_i)$ can be collected from a user model or are explicitly entered. GC typically occurs first in $\mathcal{S}^C$.
- CCF denotes the **cascading context files**, which permit the contextualization analogously to cascading stylesheets [Cas99].
- IC denotes the **intensional context**, which associates a list of contextual key-value pairs $(d_i = v_i)$ to any node of the input document. These express the author's intensional context. For the implementation in XML formats, we use an `ic` attribute similar to the `ec` attribute above, i. e., the effective intensional context depends on the position of ω in the input document (see a concrete example below).
- MD denotes **metadata**, which typically occurs last in $\mathcal{S}^C$.

In the second step, the rendering context Λ_ω is matched against the context annotations in L. We select the pair (ρ, p) whose corresponding context annotation satisfies the intensional declaration best.

In Fig. 5, we continue our illustration on the given input document. The dashed arrows represent extensional references, the dashed-dotted arrows represent intensional references, i. e., implicit relations between the `ic` attributes of the input document and the context-annotations in the notation document. A global context is declared, which specifies the language and course dimension of the document. We apply the algorithm above with the input object ω_3, $\mathcal{S}^C = (GC, IC)$, and a list

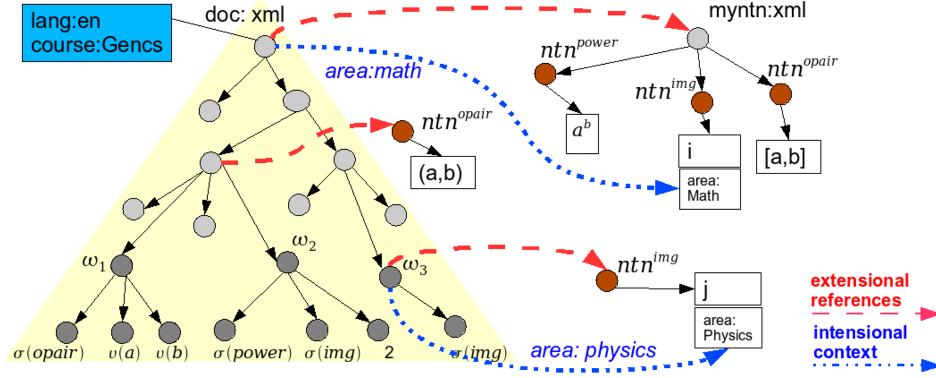

FIGURE 5. Rendering Example

of context annotations and rendering pairs based on the formerly created notation context Π_{ω_3}. For convenience, we do not display the system's default notation document and the precedences.

---

1. We compute the intensional rendering context
1.1. We collect contextual information from GC
$\Lambda_{\omega_3} = (lang = en, course = GenCS)$
1.2. We collect contextual information from IC an append them to Λ_{ω_3}
$\Lambda_{\omega_3} = (lang = en, course = GenCS, area = physics, area = math)$

2. We match the rendering context against the context annotations of the input list L and return the rendering with the best matching context annotation:
$L = [\ (\lambda_0 = j), (\lambda_1 = i), (\lambda_2 = imaginary)\ ]$
$\lambda_0 = (area = physics)$, $\lambda_1 = (area = maths)$, and $\lambda_2 = \emptyset$

For simplicity, we compute the similarity between Λ_{ω_3} and λ_i based on the number of similar $(d_i = v_i)$: λ_0 includes $(area = physics)$. λ_1 includes $(area = math)$. λ_2 is empty. λ_0 and λ_1 both satisfy one $(d_i = v_i)$ of Λ_{ω_3}. However, since $(\lambda_0 = \rho_0)$ occurs first in Π_{ω_3}, the algorithm returns ρ_0.

Discussion of Context-Sensitive Selection Strategies. In the following, we illustrate the advantages and drawbacks for collecting contextual information from GC, CCF, IC and MD.

1. The declaration of a *global context* provides a more intelligent intensional selection between alternative $(\lambda\colon \rho)^p$ triples inside one notation definition: The globally defined $(d_i = v_i)$ are matched against the context-annotations λ to select an appropriate rendering ρ. However, the approach does not let users specify intensional contexts on granular levels.
2. Considering *metadata* is a more granular approach than the global context declaration. However, metadata may not be associated to every node in the

input document and cannot be overwritten without modifying the input document. But note that modern metadata formats are quite flexible. For example, metadata in can only be put into the header of a document and the vocabulary is limited, but the recent RDFa extension to (X)HTML allows for adding metadata to almost any fragment of a document, without being limited to a particular metadata vocabulary [ABMP08]. The same holds for OMDoc, the language in which we have implemented our findings (see section 5). – Still, metadata cannot be assigned in that way *inside* mathematical objects.

3. The *intensional context* can be associated to any node of the input document and thus supports a fine-granular selection of renderings. However, the intensional references cannot be overwritten on granular document levels by readers.
4. *Cascading Context Files* permit a granular overwriting of contexts.

5. Authoring Notations

So far we have illustrated our notation framework, in particular, the rendering algorithm with its various input options for notation definitions and contexts. Notation definitions are an essential part of our framework and inscribe context-dependent notation preferences. Being such central objects, *how and where should we create, maintain, and preserve notation definitions*? To answer these questions, it is essential to understand the structure of mathematical knowledge as well as its communication. This section introduces the *layers of mathematical knowledge* as well as a workflow for the *creation and maintenance of notations.*

5.1. Layers of Mathematical Knowledge

So far, we have based our framework on the level of mathematical *objects* and their notations. Our framework relies on the markup language OpenMath for representing the functional structure of mathematical objects, whereas MathML represents their layout.

For the way mathematicians work, this level of support is insufficient, because definitions, theorems, their proofs, whole theories, and even networks of theories are the primary focus of mathematical communication. Beyond formulae, there are large-scale structures of mathematical knowledge; mathematicians navigate the space of mathematical knowledge along them and use them to anchor their communication and new developments. Symbols and their notations do not float in a vacuum but are introduced in the context of theories: groups of mathematical statements, e. g. those in a chapter or section of a textbook. Thus, the theory context is crucial to understanding a mathematical text. It can usually be inferred from a document – be it stated explicitly (e. g. by the title of a book) or implicitly (e. g. by the fact that the e-mail comes from a person that we know works on finite groups).

For managing notations in this wider context, we need a representation of these higher layers, and systems that allow users to manage notations within the context of them. The *Open Mathematical Document Format* OMDoc [Koh06] can represent mathematical knowledge above the object layer. The format extends OpenMath and MathML with markup primitives for the structure and interrelations of mathematical objects expressed as **mathematical statements**, i. e. definitions, theorems, and proofs. In OpenMath, content dictionaries (CDs) serve as an explicitly represented context for definitions of mathematical symbols (see Section 2.1). OMDoc allows for modeling CDs as **theories** containing mathematical statements, but extends this functionality with a very expressive and extensible infrastructure for relations within and among theories that facilitate concept inheritance, parametric reuse, and multiple views on mathematical objects and statements. Finally, **documents** can be written that consist of narrative and content layers. Content layers contain statements or theories, whereas narrative layers arrange snippets from content layers into a sequential order. This facilitates the reuse of content from a shared knowledge base (also called "content commons") in documents that are actually consumed by humans: scientific articles, books, or slide shows. Altogether, the statement and theory level and the support for reusable documents make OMDoc an ideal representation format for sophisticated notation support. In Sections **??** and **??**, we present OMDoc-based systems that make the higher layers of mathematical knowledge usable in practice.

5.2. Creation and Maintenance of Notations

Having introduced *where* notations can be defined and maintained, we now need to have a look at *how* this should be done. This does not just mean making individual notation definitions editable, but supporting the whole maintenance *workflow* – from offering assistance in all situations where a user would want to create or modify a notation or a set of related notations to previewing and finally committing the changes the user made.

The need for creating or modifying a notation definition for a particular mathematical symbol usually arises from dealing either with its declaration or definition, or its occurrence in a formula. Typical settings include the initial addition of that symbol to a content dictionary and subsequent revisions of the latter. One such revision task can be fixing a faulty notation definition, or adding an alternative one. This task is not only of interest to the original content dictionary authors, but also to authors of documents *using* these symbols, and ultimately to any reader who is not satisfied with the existing selection of notations that are applicable to the formulae she is looking at.

In section 5.3, we will introduce several concrete representations for notation definitions, each of which should be supported by a dedicated editing mode. While we recommend a text editor for our ASCII syntax, a form-based user interface would be more appropriate for the declarative syntax. But an editor should not only be available, it should also be ready to use whenever the user is in one of the situations outlined above. There should be an easy way to get from a declaration

or occurrence of a symbol to the notations defined for it, and to get an overview of available notation definitions – e. g. all notation definitions for a particular symbol, all notation definitions for a particular value of an intensional context dimension (e. g. "language=French"), etc. For the storage backend, this means that there has to be an efficient index of available notation definitions, which is kept up to date whenever a notation definition is imported or edited.

Once an author has *changed* the notation definition, the effect of this change has to be determined. In applications that do not give the reader full influence on notation choice, it is possible to know from a document Doc and from the notation database DB (see section 3.2) what notation definitions are needed for rendering the respective document. (As a trivial example, consider a knowledge base with exactly one notation per symbol.) That allows for caching rendered documents and only re-rendering them if either the source code of the document or one required notation definition has changed.

For making this workflow of revising a dissatisfying notation and reviewing the re-rendered document(s) and then, possibly, switching back to the notation editor again, more smoothly, we suggest an integrated *preview* – at least displaying the effect of the user's edit on the formula from which the user reached the notation editor. The effect on similar formulae – either taken from the documents in the knowledge base or generated on the fly – could be shown as well. Note that, if a formula is isolated from its parent document for the purpose of previewing, the context that it inherited from its container, has to be preserved for rendering it correctly. Consider an English document about binomial coefficients containing the following sentence: "We write the binomial coefficient of n over k as $\binom{n}{k}$, but a Russian would write $\mathcal{C}_n^k$." Suppose our notation definitions have language context annotations, and let the notation choice for the first binomial coefficient be determined by the language of the whole document, English in our example. Let the second notation choice be determined by a local intensional context "language=Russian". If an author then wants to edit the English notation, the document snippet rendered for previewing needs to contain the information that its language is English.

So far, we have oversimplified the notion of "the" notation definition to be edited. Our notation definitions map a [set of] pattern[s] to a [set of] rendering[s], unless we assume a declarative syntax, as will be introduced in section 5.3. For every symbol to be rendered, from all renderings that are specified for patterns that match the occurrence of the symbol, the one is applied that matches the effective rendering context best. That is, for every symbol that has been rendered, we know a unique pattern/rendering pair $n = \varphi \vdash \rho$ that the renderer used[2]. Note that, in most cases, the author will only need to change the rendering; therefore, we will focus on that case in the following.

The prototype/rendering pair $n = \varphi \vdash \rho$ does not necessarily exist exactly like this in the knowledge base; a superset like $n' = \varphi, \varphi', \ldots \vdash \rho, \rho'', \ldots$ is more likely.

[2]For clarity, we omitted the context and precedence annotations of the rendering ρ.

The editor should not bluntly display the notation definitions as they happen to be stored, but as they are applied in the concrete case of interest. There are cases when it is of interest to edit more than a single pattern/rendering pair at once, which ones, besides φ and ρ, has to be decided from the concrete occurrence of the symbol and the given rendering context. When editing $n = \varphi \vdash (\lambda\colon \rho)^p$, other renderings of interest are those n' whose context annotations λ' do not contradict λ (i. e. do not have contradicting values for the same context dimensions). This is often the case when there are are prototypes for the same symbol occurring in different *roles*. For example, in trigonometry, we usually *apply* the tangent function to an argument, i. e. an angle. The prototype that matches this is $@(tan, x)$. But if we talk about vector spaces of functions and talk about the tangent function *itself*, we use it in the *constant* role and match it just by the prototype tan. Clearly, there are notational changes that should be applied to the renderings for both roles, e. g. changing the spelling of the function from tan to tg.

5.3. Input Formats

To suit different editing needs, we developed several concrete representations for our abstract grammar of notation definition, as given in section 3.1. The XML syntax has the full expressivity, at the expense of being verbose. As a more concise syntax for practical cases, we then present a declarative syntax, also in XML. Finally, we introduce an ASCII representation corresponding to the full XML syntax, suitable for text-mode environments. Both the full and the declarative XML syntax will be incorporated into the upcoming version 1.6 of OMDoc, where they replace the old presentation module of version 1.2 [Koh06].

XML Syntax. We choose XML as the framework for our concrete syntax. This choice is based in our objective to provide scalable distributed services to handle notations. Therefore, we defined an encoding function $E(-)$ that maps the grammar of Section 3.1 to XML elements. The encoding of notations and renderings is given in Fig. 3. Fig. 4 gives the encodings of patterns where we use OpenMath [BCC+04] as our specific XML pattern languages. An encoding using content MathML can be defined similarly.

We assume that the following namespace bindings are in effect:

xmlns="http://omdoc.org/ns"
xmlns:**om**="http://www.openmath.org/OpenMath"
xmlns:**m**="http://www.w3.org/1998/Math/MathML"

As an example, Figure 6 shows a notation specification for the binomial coefficient. The prototype pattern matches OpenMath expressions such as the one in Figure 1. Except for jokers, the syntax is the same as the one of an OpenMath expression. This allows for easily starting the prototype part of a notation specification by copying an expression from a document. The concrete presentation for the expression is induced from the `rendering` elements. Note that there can be multiple renderings for a pattern, which are distinguished by the context attribute, which associates them with specific context parameters. In the example, the nationality of the respective notations are added. This allows to distinguish

$E(\varphi_1, \ldots, \varphi_r \vdash (\lambda_1 : \rho_1)^{p_1}, \ldots, (\lambda_s : \rho_s)^{p_s})$	<**notation**> <**prototype**>$E(\varphi_1)$</**prototype**> $\vdots$ <**prototype**>$E(\varphi_r)$</**prototype**> <**rendering** **context**="λ_1" precedence="p_1"> $E(\rho_s)$</**rendering**> $\vdots$ <**rendering** **context**="λ_s" precedence="p_s"> $E(\rho_s)$</**rendering**> </**notation**>
$E(\rho_1, \ldots, \rho_r)$	$E(\rho_1) \ldots E(\rho_r)$
$E(\langle S \rangle \rho \langle / \rangle)$	<**element** **name**="S"> $E(\rho)$ </**element**>
$E(S = "\rho")$	<**attribute** **name**="S"> $E(\rho)$ </**attribute**>
$E(S)$	<**text**>S</**text**>
$E(\underline{p})$	<**render** **name**="p"/>
$E(\underline{p}^q)$	<**render** **name**="p" precedence="q"/>
$E(\texttt{for}(\underline{l}, I, \rho')\{\rho\})$	<**iterate** **name**="l" step="I"> <**separator**> $E(\rho')$ </**separator**> $E(\rho)$ </**iterate**>

TABLE 3. Encoding of Notations

the German, Russian, and French notation of the binomial coefficient. Analogously, further context parameters such as the expertise level (novice, intermediate, expert) or area of application (mathematics, physics) can be added.

Figure 7 provides two alternative representations of notation tags to support the previously described options for the collection (see Section 4.1): The `ec` attribute (input option *EC*) as well as a `tag` element (in combination with input option *Doc* and *F*). Both, *ec* attribute and `tag` element, associate a list of `rendering` elements to every node in the input document, in particular, mathematical expression (both in OpenMath and MathML). The first URI in the list references the rendering element that was or should be used during the conversion, and the latter URIs reference alternative rendering elements for the expression. Please note that each `ec` attribute only references the set of renderings for a particular symbol of an expression. More complex expressions will thus include several `ec` attributes.

The `tag` element is specified by the following attributes: The `type` attribute provides the type of the tag; here, we only use the type `notation` for notation tags. The `xref` attribute points to the referenced notation, i. e., the `rendering` element of a `notation` specification. Its value is a URI reference to a rendering element.

$E(\sigma(b,c,n))$	<**om:OMS** cdbase="b" **cd**="c" **name**="n"/>
$E(\upsilon(n))$	<**om:OMV name**="n"/>
$E(@(\varphi_1,\ldots,\varphi_n))$	<**om:OMA**>$E(\varphi_1),\ldots,E(\varphi_n)$</**om:OMA**>
$E(\beta(\varphi,\Upsilon,\varphi'))$	<**om:OMBIND**> $E(\varphi)$ $E(\Upsilon)$ $E(\varphi')$ </**om:OMBIND**>
$E(\alpha(\varphi,s\mapsto\varphi'))$	<**om:OMATTR**> <**om:OMATP**>$E(s)$ $E(\varphi')$</**om:OMATP**> $E(\varphi)$ </**om:OMATTR**>
$E(\upsilon_1,\ldots,\upsilon_n)$	<**om:OMBVAR**> $E(\upsilon_1),\ldots,E(\upsilon_n)$ </**om:OMBVAR**>
$E(\underline{s})$	<**expr name**="s" type="symbol"/>
$E(\underline{v})$	<**expr name**="v" type="variable"/>
$E(\underline{o}[\varphi])$	<**expr name**="o" type="expression"> $E(\varphi)$ </**expr**>
$E(\underline{o})$	<**expr name**="o" type="expression"/>
$E(\underline{l}(\varphi))$	< **exprlist name**="l"> $E(\varphi)$ </ **exprlist** >

Table 4. OpenMath Encoding of Patterns

The remaining attributes specify the *conditions*, which are used to prioritize tags: The `weight` attribute represents the relevance of the tag. The `context` attribute associates context parameters with the notation tag. The `object` attribute references the mathematical objects (or subparts), to which the rendering has been and preferably should be applied to. If the `object` reference of a notation tag is set, the notation tag is only valid for the referenced objects. Vice versa, if the object reference is missing, the notation tag is valid for any object that matches its pattern. The notation tag have an optional attribute `owner`, which e. g. relate a tags to the respective author of a document. Its value is a URI reference to the user profile or unique identifier (such as OpenID [Ope08] or FOAF URI [FOA08]). For simplicity, we do not display the latter attribute in Figure 7.

Declarative Syntax. Our pattern matching syntax is actually far too complicated for most cases. While it is needed to cover special cases like the multiple integral shown in section 3.1, the majority of notations can be written much simpler. Two simplifications turn out to be useful.

Firstly, all symbols that can be rendered compositionally do not need to give a pattern. For example, when rendering the application of the symbol *plus* to arguments $a_1,\ldots,a_n$, it is sufficient to render all arguments a_i independently, resulting in say r_i, and then give the rendering of the whole expression as a function

```
<notation xmlns="http://omdoc.org/ns"
    xmlns:m="http://www.w3.org/..."
    xmlns:om="http://www.openmath.org/...">
  <prototype>
    <om:OMA>
      <om:OMS cd="combinat1" name="binomial" />
      <expr name="arg1"/>
      <expr name="arg2"/>
    </om:OMA>
  </prototype>
  <rendering context="lang:Russian,ru">
    <m:msubsup>
    <m:mi>C</m:mi>
    <render name="arg1"/>
    <render name="arg2"/>
    </m:msubsup>
  </rendering>
  ...
  <rendering context="lang:German,de">
    <m:mrow>
      <m:mo>(</m:mo>
      <m:mfrac linethickness="0">
        <render name="arg1"/>
        <render name="arg2"/>
      </m:mfrac>
      <m:mo>)</m:mo>
    </m:mrow>
  </rendering>
  <rendering context="lang:French,fr">
    <m:msubsup>
      <m:mi>C</m:mi>
      <render name="arg2"/>
      <render name="arg1"/>
    </m:msubsup>
  </rendering>
</notation>
```

FIGURE 6. XML Representation of a Notation Practice.

```
<notation ... xml:id="ntn123">
  <rendering xml:id="rend456"> ... </rendering>
  <rendering xml:id="rend789"> ... </rendering>
</notation>

<tag type="notation" xref="ntn123#rend456"
     context="hasLanguage:German" weight="5" object="obj455,obj456" />

<m:math> ...
 <m:mrow xml:id="obj456">...</m:mrow> ...
 <m:mrow xml:id="obj222"" ec="ntn123#rend789">...</m:mrow>
</math>
```

FIGURE 7. Two Example Notation Tags

$R(r_1, \ldots, r_n) = r_1 + \ldots + r_n$. In particular, R does not depend on the structure of the a_i. Such notations should be given by simply specifying R.

Secondly, many notations are instances of very general classes of notations. For example, the above notation of *plus* can in fact be given much simpler by saying that *plus* is an associative infix symbol with operator +.

In the following we define two kinds of abbreviated notations and define their semantics by elaboration into our main syntax.

The first kind of notations is defined by abbreviated patterns and normal renderings. An abbreviated pattern is a tuple of

- a component $g?m?s$, which is a URI identifying the symbol $\sigma(g, m, s)$,
- a component $role$, which is the role in which the symbol is used, and whose value is among `constant`, `application`, `binding`, `attribution`,
- a fixity string fix of the form $i^*[ai^*]$ where i and a are terminal symbols and [] denotes optional parts, whose meaning is explained below.

The intuition behind $fix = i^m a i^n$ is that the first m and the last n arguments receive special treatment whereas the remaining arguments in the middle are rendered as some kind of list. Precisely, the abbreviated pattern $(g?m?s, role, i^m a i^n)$ is expanded as follows:

- if $role$ is `constant`: $\underline{operator}[\sigma(g,m,s)]$,
- if $role$ is `application`: $@(\underline{operator}[\sigma(g,m,s)], \varphi)$ where $\varphi := \underline{pre1}, \ldots, \underline{prem}, \underline{list}(\underline{mid}), \underline{post1}, \ldots, \underline{postn}$,
- if $role$ is `binding`: $\beta(\underline{operator}[\sigma(g,m,s)], \underline{list}(\underline{mid}), \underline{post1})$,
- if $role$ is `attribution`: $\alpha(\underline{pre1}, \underline{operator}[\sigma(g,m,s)] \mapsto \underline{pre2})$.

Note that the fixity string is only relevant in the first case.

The XML encoding of such notations is analogous to the one given above, except that the prototype child is dropped and attributes `for`, `role`, and `fixity` are added, whose values are the components of the abbreviated pattern. For example, the notation $T_1 \times \ldots \times T_n \to T$ of the $n+1$-ary function type constructor can be given in LaTeX as

```
<notation for="http://cds.omdoc.org/logics?simpletypes?funtype" role="application" fixity="ai">
 <rendering>
  <iterate name="list" step="1">
   <separator><text>\times</text></separator>
   <render name="mid"/>
  </iterate>
  <text>\to</text>
  <render name="post1"/>
 </rendering>
</notation>
```

The second kind of abbreviated notations additionally uses abbreviations for the rendering. Such a notation is a tuple of

- components $g?m?s$ and $role$ as above,
- a fixity string fix (fixity) of the form
$$fix ::= (o|i|a|A|s|S)^*$$
where only one occurrence of a or A is permitted,
- two components sep and Sep (separators) of the form ρ^*,
- a component $brack$ (bracketing) of the form $(\rho|@)^*$,
- a precedence p.

The pattern of the elaborated notation is determined as above where all occurrences of o, s, and S in the fixity string are ignored and occurrences of A are treated like occurrences of a.

The unbracketed rendering of the elaborated notation is determined by characterwise induction on the fixity string:

- o yields $\underline{operator}^{\infty}$.
- i yields $\underline{pren}^p$ or $\underline{postn}^p$ for the appropriate value of n according to the position of i in the fixity string.
- a yields $\texttt{for}(\underline{list}, sep)\{\underline{mid}^p\}$.
- A yields the same as a but with Sep instead of sep.

- s and S yield *sep* and *Sep*, respectively.

Finally, this is bracketed by wrapping it in *brack* at the position determined by @.

The XML encoding of such notations is given by `notation` elements with `for`, `role`, `fixity`, and `precedence` attributes, and `separator`, `Separator`, and `brackets` children.

For example, the Presentation-MathML notation $\forall x_1, \ldots, x_n.F$ of a universal binder can be given as in the following figure. Note that the appearance of the operator symbol itself (∀, the o in the fixity string) is defined by an abbreviated notation of the first kind.

```
<notation for="http://www.openmath.org/cd?quant1?forall" role="application" fixity="oaSi"
                                                                       precedence="p">
 <separator><element name="mo"><text>,</text></element></separator>
 <Separator><element name="mo"><text>.</text></element></Separator>
 <brackets><element name="mfenced"><at/></element></brackets>
</notation>

<notation for="http://www.openmath.org/cd?quant1?forall" role="constant" fixity="">
 <rendering><text>∀</text></rendering>
</notation>
```

Simple ASCII Syntax. As a different approach towards a syntax that is easier to author than the XML pattern matching syntax, we have also developed an ASCII syntax for pattern matching. While we envisage the XML syntaxes to be used in authoring environments with a dedicated support for an easy input of XML structures, such as the rich text editor of the semantic wiki SWiM (see section 7.5) or a form-based interface, the ASCII syntax is rather targeted at text editors or command-line interfaces, where it is more suitable due to its conciseness.

The ASCII syntax closely corresponds to the abstract pattern matching syntax introduced in table 2, except that non-ASCII symbols are transcribed using similar-looking characters. We did not exactly reproduce the syntax for patterns either, but used a style that looks less intrusive and more familiar to programmers and users of automated theorem provers or computer algebra systems due to its similarity with Lisp S-expressions. Similarly as for the XML syntax, we define the semantics of the ASCII syntax by an encoding function $E_A(-)$ that maps the grammar of section 2 to ASCII.

6. Exploiting Notations

Having focused on authoring notations in the previous section, we will now take the end-user's point of view and present notation-based services that can be offered on top of a knowledge base containing notation definitions.

6.1. Overview

In order to evaluate our framework, we have gathered a list of requirements for notation-based services through systematic discussions with mathematics lecturers

$E_A(\varphi_1,\ldots,\varphi_r \vdash (\lambda_1\colon \rho_1)^{p_1},\ldots,(\lambda_s\colon \rho_s)^{p_s})$	$E_A(\varphi_1)\texttt{,}\ldots\texttt{,}E_A(\varphi_r)\vert\texttt{-}$ $\{\lambda_1\}E_A(\rho_1)\texttt{\^{}}p_1\texttt{,}\ldots\texttt{,}\{\lambda_s\}E_A(\rho_s)\texttt{\^{}}p_s$
$E_A(\langle S\rangle\rho\langle/\rangle)$	<S>$E_A(\rho)$</>
$E_A(S=\text{"}\rho\text{"})$	S="$E_A(\rho)$"
$E_A(S)$	"S"
$E_A(\underline{p})$	$\$p$
$E_A(\underline{p}^q)$	$\$p\texttt{\^{}}q$
$E_A(\texttt{for}(\underline{l},I,\rho')\{\rho\})$	$\texttt{for(}\$l\texttt{;}\ E_A(\rho')\texttt{;}\ I\texttt{)}\{E_A(\rho)\}$
$E_A(\sigma(n,c,b))$	$b?c?n$
$E_A(\upsilon(n))$	N
$E_A(@(\varphi_1,\ldots,\varphi_n))$	$(E_A(\varphi_1)\ \ldots\ E_A(\varphi_n))$
$E_A(\beta(\varphi,\Upsilon,\varphi'))$	$(E_A(\varphi)\ E_A(\Upsilon)\ E_A(\varphi'))$
$E_A(\alpha(\varphi,s\mapsto\varphi'))$	$E_A(\varphi)\{E_A(s)\texttt{:}E_A(\varphi')\}$
$E_A(\upsilon_1,\ldots,\upsilon_n)$	$[E_A(\upsilon_1)\ldots E_A(\upsilon_n)]$
$E_A(\underline{o})$	$\$o$
$E_A(\underline{l}(\varphi))$	$\$l(E_A(\varphi))$

TABLE 5. Encoding Notations into ASCII

and researchers. Below we outline a "wish list" compiled from these interviews, applied to the binomial coefficient example for illustration.

1. Alternative ways of displaying symbols:
   - (a) Show all alternative notations: $\mathcal{C}(n,k)$, ${}_n\mathcal{C}_k$, $\mathcal{C}^n_k$, $\mathcal{C}^k_n$, $\binom{n}{k}$
   - (b) Point out particular notational differences: "We write $\binom{n}{k}$, but you used to know it as $\mathcal{C}^n_k$."
   - (c) Allow to change notations on the fly while reading a document
   - (d) Provide an example of the notation with concrete values: $\binom{5}{3}$
   - (e) Read notations out loud: "*n choose k*" (an "aural notation", in fact) – a service that we consider particularly helpful for foreigners to become acquainted with the technical vocabulary of a new language
   - (f) Provide a natural language term for the concept: *binomial coefficient*
2. Explain the structure of a formula:
   - (a) Flexible display and hiding ("elision") of brackets: If the reader is not yet familiar with the precedences of new operators, allow for making the structure of a term more explicit by showing redundant brackets.
   - (b) Folding of complex subterms: allow for collapsing terms whose full rendering takes up too much space; replace subterms that are hard to understand by instructive labels – e. g. by their scientific meaning: "potential energy" or $W_{\text{pot}}(R)$ instead of $\frac{-e^2}{4\pi\epsilon_0 R/2}$.
3. Use additional knowledge from the definition of a symbol, if the reader does not understand the notation itself, or wants to do further explorations for other reasons:

(a) Interlink symbols and their definition (of different level of formality)

(i) Expand a symbol using its formal definition: $\binom{n}{k} \rightsquigarrow \frac{n!}{k!(n-k)!} \rightsquigarrow \frac{n(n-1)...(n-k+1)}{k(k-1)...1}$ – either showing the definition as a tooltip, or inserting it into the formula in place of the symbol

(ii) Provide an informal explanation: *The number of k-element subsets of an n-element set.*

(iii) Provide an even more informal explanation: *The number of ways that k things can be chosen from a set of n things.*

(b) Generate a guided tour to explain a given (complex) formula.

Our framework allows for alternative ways of displaying symbols (item 1) using the information on the object level: Thanks to parallel markup, an application knows the formal symbol from which a certain presentational symbol was rendered and can look up alternative notations in the knowledge base. Looking up alternatives can optionally be guided by high level knowledge: Instead of showing *all* alternative notations for a symbol, a system can display a selection of *relevant* alternatives, e.g., notations common in the user's community or notations preferred by users with similar profiles. Aural output (item 1e) is not directly possible using MathML, but enabling it is merely a matter of encoding spoken language and transforming it into actual sound; thus, it does not require any conceptual changes to our framework.

The structure-oriented services outlined in item 2 can also be provided on the object level. Bracket elision relies on information about operator precedences (see section 6.2). Subterm folding only requires a proper encoding of the rendered formula. That given, the rendering engine as a core part of our framework enables the folding service (see section 6.3).

Accessing additional knowledge, beyond the symbol and its notation(s) themselves, as in item 3, requires semantic markup on higher levels of mathematical knowledge (cf. section 5.1). Formal definitions on the statement level enable expansion of definitions (item 3(a)i); informal explanations of symbols (items 3(a)ii, 3(a)iii) can be provided as metadata of those formal definitions. Guided tours (item 3b) go beyond direct symbol→definition links but may extend into additional, more foundational theories.

The amount of knowledge required by a service has implications on its architecture. In a client/server system, where documents are rendered on the server and browsed on the client, one needs to balance the number of server requests and the size of a response from the server. All information that is required for the structural manipulations in item 2 can be provided inside the formula markup at almost no additional cost. Alternative notations (item 1) are likely to be requested occasionally, and just for particular symbols; thus, it makes more sense to serve them on demand. (Still, the client can maintain a cache to speed up switching back to a notation that has been selected previously.) The same holds for the expansion of symbol definitions (item 3). Guided tours (item 3b) are an extreme

case, where all required information could only be embedded into a single request at an exponential cost.

Unless the server wants to maintain the complete state of every client, both notation selection and definition expansion require a linear overhead for additional markup. For selecting alternative notations, the client must at least know what notation n has been used to obtain the current rendering, so that it can ask the server to provide a list of alternatives to n. This is achieved by having the renderer attach notation tags (cf. section 4.1) to the presentation markup. For *rendering* some symbol with an alternative notation, parallel markup is used to send a reference to the needed content-markup expression to the server. For expanding definitions, parallel markup is used as well, as it tells what formal symbol identifier corresponds to a rendered symbol; using that information, the server can look up the definition.

So far, our framework supports the structural services (2a, 2b), which we will introduce in the following sections. However, as outlined above, further services can be added incrementally without affecting our current design. In particular, our rendering algorithm and the web-based user interface presented below can stay the same.

6.2. Flexible Elision

Automated presentation of mathematical formulae can be regarded as a two-step process of *composition* of visual sub-presentations to larger ones (see section 3.2) and *elision* of formula parts that can be deduced from context. 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.

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. **Brackets** that are redundant due to operator precedences can be omitted. $ax + y$ is actually $(ax) + y$, since multiplication binds stronger than addition[3]. **Arguments** with default values are frequently omitted: $\log_{10} x$ is often written as $\log x$, as the default base 10 is established by common practice. Arguments whose values can be inferred from other arguments can also be 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. Finally, there are arguments that are strictly necessary but omitted nevertheless, as the reader is trusted to fill them in from the context: We write $[\![t]\!]$ for $[\![t]\!]^{\varphi}_{\mathcal{M}}$, if there is only one model $\mathcal{M}$ in the context and φ is the most salient variable assignment. A concluding extreme example of bracket elision is **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, $\forall x, y.\varphi \wedge \psi$ stands for $\forall x, y(\varphi \wedge \psi)$.

[3]Note that we would not consider the "invisible times" rendering as another elision, but as an alternative notation.

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 and to make reading more efficient, 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 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 component of a rendering (e.g. a separator or some text) an integer *elision level* and group them into *elision groups*. The elision level can then be used to determine the visibility of symbols based on user-provided thresholds: the higher its elision level, the less important a symbol. In our XML encoding of notation definitions (cf. section 5.3), we achieve that by allowing additional attributes `elevel` and `egroup` on all elements that are allowed as children of renderings and can generate output or contain child elements that can generate output.

Brackets form a special elidability group of their own, named "fence". Elision levels in this group are computed by the rendering algorithm. Assume we render a content-markup object O to $\overline{O}^p$, where p is the initial input precedence, and the rendering chosen for O has the output precedence q and returns the the token string B. Then every bracket token in B is given the elision level $q - p$. Thus brackets where the difference between input and output precedence is higher are more elidable because they are easier to reconstruct. Brackets with a non-positive elision level 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 infinity, which means that top-level brackets have the highest elidability. In static (print) media, 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 elision group g as a user parameter and then elide all tokens in elision group g with level $l > T_g$.

For active documents, the information about elision groups and levels can be exported to the target format, so that a viewer application can then dynamically change the visibility thresholds by user interaction. Figure 8 shows a document with two elision groups (brackets and type annotations). In the main view, all symbols are made visible, higher elision levels being indicated by a lighter shade of gray[4]. We have pasted the appearance with all elidable symbols hidden into

[4]Actually, we had to resort to plain XHTML instead of MathML for the formulae in this demo, as MathML does not yet support dynamic control of styles via DOM access to CSS.

Flexible Elision Demo

- Powered by JOMDoc and JOBAD
- Tested with Firefox ≥ 2.0 and Opera ≥ 9.0

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

$\wedge^*_{(\iota\to o)\to((\iota\to o)\to(\iota\to o))}=\lambda F_{\iota\to o}G_{\iota\to o}X_\iota.F_{\iota\to o}(X)_\iota\wedge_{o\to(o\to o)}G_{\iota\to o}(X)_\iota$

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

$\wedge^*=\lambda FGX_\iota.F(X)\wedge G(X)$

Visibility Thresholds:

- brackets: ○ 0 ○ 100 ○ 200 ○ 300 ○ 400 ○ 500 ◉ infinite
- types: ○ 0 ○ 100 ○ 200 ◉ 300

Operator	Mixfix declaration
x^y	199|₁ ∞|₂ :200
!	300|₁ !:300
·	400|₁ · 400|₂ :400
+	500|₁ + 500|₂ :500

Operator	Mixfix declaration
¬	¬ 600|₁ :600
≤	700|₁ ≤ 700|₂ :700
∧	1000|₁ ∧ 1000|₂ :1000
∨	1200|₁ ∨ 1200|₂ :1200

FIGURE 8. Flexible Elision of Brackets and Types

the inverted-color area on the right. As an alternative access method, elision could also be triggered locally for subterms via their context menus.

Our notation framework 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 article: *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 article to constitute a first step into the right direction, but also that we need more insights to solve the abbreviation generation problem.

6.3. Folding Subexpressions

We have considered two cases of folding subterms: just making them collapsible in case the reader feels distracted by them, or allowing to switch between the original term and a meaningful abbreviation (see item 2b on page 28).

Support for folding with abbreviations prepared by the document author is easiest to realize: we provide the special attribution key symbol *?folding?abbrev* that can be used as an attribution key. That said, our example from physics would look like $\alpha(\frac{-e^2}{4\pi\epsilon_0 R/2}, \mathit{?folding?abbrev} \mapsto W_{\mathrm{pot}}(R))$. The built-in notation definition for this symbol turns such an attributed formula into a MathML *action* element with two children – the abbreviated and the expanded form:

$$\begin{array}{l} \alpha(\underline{\texttt{formula}}, \mathit{?folding?abbrev} \mapsto \underline{\texttt{abbr}}) \\ \vdash \langle \mathit{maction\ actiontype} = \text{"}\mathit{folding}\text{" } \mathit{selection} = \text{"1"} \rangle \\ \quad \underline{\mathit{abbr}} \\ \quad \underline{\mathit{formula}} \langle / \rangle \end{array}$$

Here, the first child (i. e. the abbreviated form) would be displayed, but any document viewer that recognizes the enclosing `maction` can offer the user to switch to the expanded form, e. g. via a command in the context menu. Alternatively, the other form either be shown as a tooltip. Another alternative is a third state of display – showing both forms, where the abbreviation is a label for the expanded form: $\underbrace{\frac{-e^2}{4\pi\epsilon_0 R/2}}_{W_{\mathrm{pot}}(R)}$.

Folding arbitrary subterms – then with default placeholders instead of hand-crafted abbreviations – requires no work on part of the document author, but some attention when writing notation definitions. The rendering for every symbol must be properly grouped. Some Presentation MathML operators group implicitly, e. g. fractions (`mfrac`) or radicals (`msqrt` and `mroot`); otherwise `mrow` must be used. For example, the expression $(x + 1) \cdot (x - 1)$ becomes foldable by subterms if encoded as follows:

```
<mrow>
  <mrow><mi>x</mi><mo>+</mo><mn>1</mn></mrow>
  <mo>·</mo>
  <mrow><mi>x</mi><mo>−</mo><mn>1</mn></mrow>
</mrow>
```

Here, the document viewer can create `maction`s on demand, in order to remember the expanded state and to allow the user to switch back efficiently. As `mrow`s do not consume any space, we can actually afford to hard-code a rule in the renderer that wraps a rendering into an `mrow` whenever it is not yet grouped.

7. Implementations

We do not work towards a monolithic implementation of our framework. Rather do we employ our notations as a standardized interface for various tools that support document-related processes. In this section we give a brief overview how our notations are used in our systems.

7.1. JOMDoc

The open-source Java library for OMDoc provides an API, a command-line frontend, and a graphical user interface, via which OMDoc documents can be parsed, validated, manipulated, and serialized. Since our notations are part of the OMDoc format, JOMDoc is their reference implementation. JOMDoc includes a full implementation of the rendering algorithm developed in the preceding sections. In particular, it provides

- the context-sensitive conversion of mathematical formulae in OMDoc documents containing by collection and application of notation definitions, context parameters, and notation tags,
- the rendering of complete documents using XSLT transformations,
- the parsing of notation definitions given in pattern-matching, ASCII, or declarative representation,
- the tracking of all renderings that are applied during the conversion and the preservation of this information in the output document as described in Section 4.1 and 5.3.

The JOMDoc library has been integrated into diverse systems, such as the semantic wiki SWiM (see Section 7.5) and the web-reader panta rhei [pan08] (see Section 7.4), allowing them to display OMDoc documents as active mathematical documents.

7.2. JOBAD

The Javascript API for OMDoc-based Active Documents (JOBAD) [JOB08] can be used in HTML documents displayed in browsers supporting JavaScript and MathML. It provides interactive views on rendered OMDoc documents. While independent from JOMDoc, it works best as a frontend to documents generated with JOMDoc as that guarantees that all necessary markup (such as parallel markup or notation tags) are created correctly.

JOBAD's user interface features mouse-over tooltips that are ready to display information retrieved from the server (such as the definition of a symbol) and a context menu for mathematical formulae. Both actions on symbols and expressions are supported. Services are represented as generic action objects, which are ready to be attached to future user interface elements like tool buttons or keystroke handlers. Concretely, JOBAD fully supports the client-side services of elision and folding; client/server services, such as notation selection and definition expansion, exist as proofs of concept.

7.3. sTeX

sTeX (semantic TeX) is a collection of macro packages that turns TeX into an input format for OMDoc [Koh08]. sTeX allows for annotating semantic structures of mathematical knowledge in TeX documents and can both be compiled to PDF – using the common TeX infrastructure – and converted to OMDoc. This architecture allows for continuing the usage of well-tried TeX editors but on the other hand paves the way for mathematical knowledge management on TeX documents.

As the PDF output targets *presentation*, on the computer screen as well as paper, mathematical notation had a high priority in the development of sTEX. The input syntax for formulae is a TEX-style equivalent of OpenMath; the usual TEX notation, in which formulae are to be rendered, has to be defined per symbol. sTEX's notation definition syntax has a similar expressivity as our declarative XML syntax, into which it can be translated.

7.4. panta rhei

panta rhei is an interactive and collaborative reader for active documents [MK07, pan08]. While users are reading, rating, and discussing their documents, implicit and explicit user modeling techniques are applied to personalize the adaptation of content (see [MK08] for details). Figure 9 displays the two constituents of the system: *panta* (the user interface) and *janta* (the backend service). *panta* implements the discussion, annotation, and tagging facilities and gathers information on the user's notation preferences. By integrating JOBAD (Section 7.2), *panta* can provide interactive services. *janta* takes over all content and user data handling.

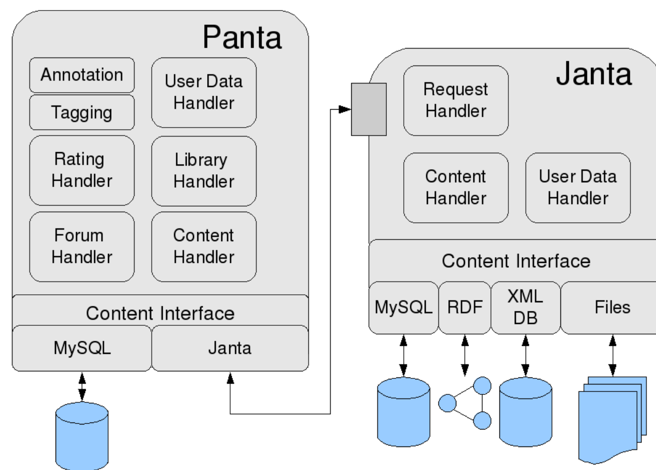

Figure 9. System Architecture

Authors can draw on the sTEX→OMDoc→XHTML workflow (see Section 7.3) to write documents in their preferred LaTeX editor and publish their results in *panta rhei*. During the conversion to XHTML semantic identifiers and metadata are preserved, which improves the web-accessibility of the imported documents. Markup of narrative structure allows us to adapt the size and navigation of documents (see [KMM07]), markup of concepts allows semantic search and easy cross-linking, enhancement with action triggers facilitates interactivity (see Section 6), and distinction of content and form supports different visualizations. For example, the introductory computer science lecture at Jacobs University Bremen has been

```
<om:OMA>
  <om:OMS cd="arith1" mcd:cr="fun" name="sum"/>
  <om:OMA>
    <om:OMS cd="interval1" name="integer_interval"/>
    <expr name="low"/>
    <expr name="high"/>
  </om:OMA>
  <om:OMBIND>
    <om:OMS cd="fns1" name="lambda"/>
    <om:OMBVAR>
      <expr name="var"/>
    </om:OMBVAR>
    <expr name="scope"/>
  </om:OMBIND>
</om:OMA>
```

FIGURE 10. An XML notation definition for the sum operator in SWIM. Note the visual preview of the rendering and the navigation links on the right.

written in sTEX and imported to *panta rhei* via JOMDOC. During the import, the lecturer can specify notation preferences, and these are used to generate an initial presentation of the course material, which can then be adapted by the students.

7.5. SWiM

SWIM [Lan08b, Lan08c] is a semantic wiki for collaboratively editing, browsing, and discussing mathematical knowledge. Editing, change tracking, and discussing is performed per wiki page, which can be any fragment of an OMDOC document or OPENMATH content dictionary, in particular a notation definition. Formulae are rendered by the JOMDOC renderer, which is fed with all notation definitions available on the wiki pages.

To keep track of existing notation definitions, mathematical symbols, and their occurrences in formulae, an RDF [LS99] outline of the mathematical documents and their cross-links is extracted whenever a page is edited or imported. These can be browsed using a navigation bar. All symbols inside rendered formulae are linked to the wiki pages on which they are defined, which is implemented by post-processing the parallel markup generated by JOMDOC. Furthermore, symbols are linked to notations to support the maintenance workflow [Lan08a, LGP08].

For editing, SWIM uses the JavaScript-based visual editor TinyMCE [Tin08], extended by OMDOC-specific plugins for an annotation toolbar and a formula editor [LGP08]. Notations can be edited in XML syntax or declarative syntax via the annotation toolbar and in ASCII syntax by direct input. SWIM is currently being evaluated as an editor for maintaining the official OPENMATH 3 content dictionaries at http://wiki.openmath.org.

8. Conclusion and Future Work

We have presented a comprehensive framework for notations in active mathematical documents that covers all phases of the document life cycle, from authoring notations to viewing interactive documents. To define notations, we gave a simple grammar and a rigorous semantics. To select among possible notations, we provided a combination of extensional and intensional ways to obtain a context-dependent association of notations with mathematical objects. To author notations, we provided an XML, an ASCII, and an abbreviating declarative syntax. To take advantage of our notation framework, we gave several examples of added-value services such as folding and elision.

Finally, we gave an overview of our suite of notation-aware tools to demonstrate the practical viability and usefulness of our framework. We provide a number of libraries and full systems that enable users to create, view, and adapt active documents. All these tools use the OMDoc, OpenMath, and MathML formats, into which our notations are integrated via their XML syntax, as their standardized interface languages.

We have evaluated our framework based on a series of interviews with mathematicians. Our findings are that our framework provides a solid and scalable foundation for services that have the potential to intrigue mathematicians and become useful in real world use cases. Specific case studies of our main systems SWiM and *panta rhei* are currently in progress.

Future work will focus on building further added-value services into the components of our tool suite. Our experiences with the implementation of first added-value services have confirmed that such services can be implemented rapidly and conveniently within our design.

Beyond notation-based services, we have done first steps towards an adaptive visualization of rhetorical structures [Gic08] and proof explanation on various levels of formality, and we have developed other systems operating on a formal representation of mathematical knowledge, including a foundation-independent semantics of modularity in theories [RK08] and a semantic formula search engine [KAJ+08]. We are planning to enhance all these systems by integrating notation-awareness.

We would like to thank the KWARC research group for their valuable feedback and discussions. Special thanks go to Dimitar Mišev for the implementation of major parts of the JOMDoc library as well as to Maja Grintal for the JOMDoc plugin for facilitating input in an ASCII syntax. We would also like to thank Jana Giceva for her contribution to the JOBAD framework and Alberto González Palomo for the implementation of the semantic extension of the TinyMCE plugin for OpenMath. Further thanks go to Andrei Aiordachioaie, Stefania Dumbrava, Josip Dzolonga, Darko Makreshanski, Alen Stojanov, and Jakob Ücker for their contribution to the panta rhei project. Moreover, we would like to thank Gordan Ristovski for his contribution to the SWiM project, and Heinrich Stamerjohanns for use cases from physics. Our work was supported by JEM-Thematic-Network ECP-038208.

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.

[ABMP08] Ben Adida, Mark Birbeck, Shane McCarron, and Steven Pemberton. RDFa in XHTML: Syntax and processing. W3C Recommendation, World Wide Web Consortium, October 2008.

[AM06] Dominique Archambault and Victor Moco. Canonical MathML to Simplify Conversion of MathML to Braille Mathematical Notations. In *Lecture Notes in Computer Science*, volume 4061, pages 1191–1198. Springer Berlin/ Heidelberg, 2006.

[ASFM07] Dominique Archambault, Bernhard Stöger, Donal Fitzpatrick, and Klaus Miesenberger. Access to scientific content by visually impaired people. *Upgrade*, VIII(2):14 pages, April 2007. Digital journal of CEPIS. A monograph in spanish was published in Novática.

[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.

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

[Cas99] Cascading Style Sheets. http://www.w3.org/Style/CSS/, 1999.

[Cha87] George J. Chaitin. *Algorithmic Information Theory*. Cambridge University Press, 1987.

[FOA08] Friend of a Friend (FOAF) project. http://www.foaf-project.org/, seen June 2008.

[Gic08] Jana Giceva. Capturing Rhetorical Aspects in Mathematical Documents using OMDoc and SALT. Technical report, Jacobs University, DERI Galway, 2008.

[JOB08] JOBAD framework – JavaScript API for OMDoc-based active documents. https://jomdoc.omdoc.org/wiki/JOBAD, 2008.

[KAJ+08] Michael Kohlhase, Ştefan Anca, Constantin Jucovschi, Alberto González Palomo, and Ioan A. Şucan. MathWebSearch 0.4, a semantic search engine for mathematics. manuscript, 2008.

[Kay06] Michael Kay. XSL Transformations (XSLT) Version 2.0. W3C Candidate Recommendation, World Wide Web Consortium (W3C), June 2006.

[KLR07] Michael Kohlhase, Christoph Lange, and Florian Rabe. Presenting mathematical content with flexible elisions. In Olga Caprotti, Michael Kohlhase, and Paul Libbrecht, editors, OpenMath/ *JEM Workshop 2007*, 2007.

[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.

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

[Koh08] Michael Kohlhase. Using LaTeX as a semantic markup format. *Mathematics in Computer Science*, 2008. to appear.

[Lan08a] Christoph Lange. Mathematical Semantic Markup in a Wiki: The Roles of Symbols and Notations. In Christoph Lange, Sebastian Schaffert, Hala Skaf-Molli, and Max Völkel, editors, *Proceedings of the 3rd Workshop on Semantic Wikis, European Semantic Web Conference 2008*, volume 360 of *CEUR Workshop Proceedings*, Costa Adeje, Tenerife, Spain, June 2008.

[Lan08b] Christoph Lange. SWiM – a semantic wiki for mathematical knowledge management. In Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis, editors, *ESWC*, volume 5021 of *Lecture Notes in Computer Science*, pages 832–837. Springer, 2008.

[Lan08c] Christoph Lange. SWiM: A semantic wiki for mathematical knowledge management. web page at http://kwarc.info/projects/swim/, seen October 2008.

[LGP08] Christoph Lange and Alberto González Palomo. Easily editing and browsing complex OpenMath markup with SWiM. In Paul Libbrecht, editor, *Mathematical User Interfaces Workshop 2008*, 2008.

[LS99] Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. W3C recommendation, World Wide Web Consortium (W3C), 1999.

[Łuk67] Jan Łukasiewicz. *Philosophische Bemerkungen zu mehrwertigen Systemen des Aussagenkalküls, Comptes rendus des séances de la Société des Sciences et des Lettres de Varsovie 23:51-77 (1930). Translated by H. Weber as Philosophical Remarks on Many-Valued Systems of Propositional Logics*. Clarendon Press: Oxford, 1967. Originally published in 1930.

[LV97] Ming Li and Paul Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, 1997.

[MK07] Christine Müller and Michael Kohlhase. panta rhei. In Alexander Hinneburg, editor, *Wissens- und Erfahrungsmanagement LWA (Lernen, Wissensentdeckung und Adaptivität) conference proceedings*, pages 318–323, 2007.

[MK08] Christine Müller and Michael Kohlhase. Context Aware Adaptation: A Case Study on Mathematical Notations. Research reports, Centre for Discrete Mathematics and Theoretical Computer Science, University of Auckland, November 2008.

[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-Stylesheets for the Conversion of Mathematical Documents into Multiple Forms. In *Proceedings of the International Workshop on Mathematical Knowledge Management*, 2001.

[OMC08] OpenMath content dictionaries. web page at http://www.openmath.org/cd/, seen June2008.

[Ope07] OpenMath Home. http://www.openmath.org/, seen March 2007.

[Ope08] OpenID: Shared Identity Service, seen June 2008.

[pan08] The panta rhei Project. http://trac.kwarc.info/panta-rhei, 2008.

[POS88] IEEE POSIX, 1988. ISO/IEC 9945.

[PZ06] Luca Padovani and Stefano Zacchiroli. From notation to semantics: There and back again. In Jon Borwein and William M. Farmer, editors, *Mathematical Knowledge Management, MKM'06*, number 4108 in LNAI, pages 194–207. Springer Verlag, 2006.

[RK08] Florian Rabe and Michael Kohlhase. An exchange format for modular knowledge. In Piotr Rudnicki and Geoff Sutcliffe, editors, *Knowledge Exchange: Automated Provers and Proof Assistants (KEAPPA)*, November 2008.

[SW06a] Elena Smirnova and Stephen M. Watt. Generating TeX from Mathematical Content with Respect to Notational Settings. In *Proceedings International Conference on Digital Typography and Electronic Publishing: Localization and Internationalization (TUG 2006)*, pages 96–105, Marrakech, Morocco, 2006.

[SW06b] Elena Smirnova and Stephen M. Watt. Notation Selection in Mathematical Computing Environments. In *Proceddings Transgressive Computing 2006: A conference in honor of Jean Della Dora (TC 2006)*, pages 339–355, Granada, Spain, 2006.

[Tin08] TinyMCE – a platform independent web based JavaScript HTML WYSIWYG editor. http://tinymce.moxiecode.com/, seen June 2008.

[W3C03] W3C. Mathematical Markup Language (MathML) Version 2.0 (Second Edition). http://www.w3.org/TR/MathML2/, 2003. Seen July 2007.

[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/.

Michael Kohlhase, Christoph Lange, Christine Müller, Normen Müller, and Florian Rabe
Computer Science Department, Jacobs University Bremen
e-mail: {m.kohlhase,ch.lange,c.mueller,n.mueller,f.rabe}@jacobs-university.de