Notations for Living Mathematical Documents

Michael Kohlhase and Christine Müller and Florian Rabe

Computer Science, Jacobs University Bremen
{m.kohlhase,c.mueller,f.rabe}@jacobs-university.de

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 over the document and their own prior publications. In print media where notations are fixed at publication time, this is an over-constrained problem. In living documents notations can be adapted at reading time, taking reader preferences into account.
We present a representational infrastructure for notations in living 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 give an abstract specification of notation definitions and the flexible rendering algorithms and show their coverage on paradigmatic examples. We show how to use this framework to render OpenMath and Content-MathML to Presentation-MathML, but the approach extends to arbitrary content and presentation formats. We discuss prototypical implementations of all aspects of the rendering pipeline.

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 paper, we conceptualize the presentation of mathematical formulae as consisting of two components: the two-dimensional **composition** of visual sub-presentations to larger ones and the **elision** of formula parts that can be deduced from context.

Most current presentation processes concentrate on the relatively well-understood composition aspect and implement only rather simple bracket elision algorithms. But the visual renderings of formulae in mathematical practice are not simple direct compositions of the concepts involved: mathematicians gloss over parts of the formulae, e.g., leaving out arguments, iff they are non-essential, conventionalized or can be deduced from the context. Indeed this is part of what makes mathematics so hard to read for beginners, but also what makes mathematical language so efficient for the initiates. A common example is the use of $\log(x)$ or even $\log x$ for $\log_{10}(x)$ or similarly $[\![t]\!]$ for $[\![t]\!]^{\varphi}_{\mathcal{M}}$, if there is only one model $\mathcal{M}$ in the context and φ is the most salient variable assignment.

Another example are the bracket elision rules in arithmetical expressions: $ax + y$ is actually $(ax) + y$, since multiplication "binds stronger" than addition. Note that we would not consider the "invisible times" operation as another elision, but as an alternative presentation.

In this situation we propose to encode the presentational characteristics of symbols (for composition *and* elision) 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 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.

Before we present our system, let us review the state of the art. Naylor, Smirnova, and Watt [NW01a,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 and, based on the manual selection of users, generates user-specific XSLT style sheets [Kay06] for the adaptation of documents. Naylor and Watt [NW01a] introduce a one-dimensional context annotation of content expressions to intensionally select an appropriate nota-

tion 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 collection of notations can be inherited, extended, reused, and shared among users. For the system presented in this paper, we have reengineered and extended the latter two proposals.

In Sect. 2, we introduce abstract syntax for notation definitions, which is used for the internal representation of our notation objects. (We use a straightforward XML encoding as concrete syntax.) In Sect. 3, we describe how a given notation definition is used to translate an OpenMath object into its presentation. After this local view of notation definitions, the remainder of the paper takes a more global perspective by introducing markup that permits users to control which notation definitions are used to present which document fragment. There are two conflicting ways how to define this set of available notation definitions: extensionally by pointing to a notation container; or intensionally by attaching properties to notation definitions and using them to select between them. These ways are handled in Sect. 4 and 5, respectively.

2 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 re-engineer content MathML representations based on this model. Furthermore, we observe that the target of the presentation process is also a tree expression: a layout tree made of layout primitives and glyphs, e.g., a presentation MathML or LaTeX expression.

To specify notation definitions, we use the one given by the abstract grammar from Fig. 1. Here $|$, $[-]$, $-^*$, and $-^+$ denote alternative, bracketing, and non-empty and possibly empty repetition, respectively. The non-terminal symbol ω is used for patterns φ that do not contain jokers. Throughout this paper, we will use the non-terminal symbols of the grammar as meta-variables for objects of the respective syntactic class.

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

Fig. 1. 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 (see Sect. 5). Each context annotation is a key-value list des-

ignating the intended rendering context. The integer values p_i give the output precedences of the renderings.

The patterns φ_i are formed from a formal grammar for a subset of OpenMath objects extended with named jokers. The jokers $\underline{o}$ and $\underline{l}(\varphi)$ correspond to \(.\) and \(\varphi\)$^+$ 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. 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 are used, together with the output precedences, to determine the placement of brackets.

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

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

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

We can do that with the following notations:

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

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

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

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

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 ∞ make 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 suppress them. Rather, we annotate them with an elision level, which is computed as the difference of the two precedences. Dynamic output formats that can change their appearance, such as XHTML with JavaScript, can use the elision level to determine the visibility of symbols based on user-provided elision thresholds: the higher its elision level, the less important a bracket.

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 $\underline{o}$ 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 $\mu(\varphi)$.
– 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 Semantics of Notation Definitions

The rendering algorithm takes as input a notation context Π (a list of notation definitions, computed as described in Sect. 4), a rendering context Λ (a list of context annotations, computed as described in Sect. 5), an OpenMath object ω, and an input precedence p. If the algorithm is invoked from top level (as opposed to a recursive call), p should be set to ∞ to suppress top level brackets.

It returns as output either text or an XML element. There are two output types for the rendering algorithm: text and sequences 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. Otherwise, $O + O'$ is a sequence of XML elements treating text as an XML text node. This operation is associative if we agree that consecutive text nodes are always merged. The algorithm inserts brackets if necessary. And to give the user full control over the appearance of brackets, we obtain the brackets by the rendering of two symbols for left and right bracket from a special fixed content dictionary. The algorithm consists of the following three steps.

1. ω is matched against the patterns in the notation definitions in Π (in the listed order) until a matching pattern φ is found. The notation definition in which φ occurs induces a list $(\lambda_1 : \rho_1)^{p_1}, \ldots, (\lambda_n : \rho_n)^{p_n}$ of context-annotations, renderings, and output precedences.
2. The rendering context Λ is matched against the context annotations λ_i in order. The pair (ρ_j, p_j) with the best matching context-annotation λ_j is selected (see Section 5.2 for details).
3. 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 sub-object ω' of ω, $M(\varphi, \omega)$ contains $n \mapsto S$ where S is the name of ω'.
- 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.

- $\langle S \rangle \rho_1 \ \ldots \ \rho_r \langle / \rangle$ is rendered as an XML element with name S. The attributes are those ${\rho_i}^M$ that are rendered as attributes. The children are the concatenation of the remaining ${\rho_i}^M$ preserving their order.
- $S =$ "$\rho_1 \ \ldots \ \rho_r$" is rendered as an attribute with label S and value ${\rho_1}^M + \ldots + {\rho_n}^M$ (which has type text due to the well-formedness).
- S is rendered as the text S.
- $\underline{s}$ and $\underline{v}$ are rendered as the text $M(s)$ or $M(v)$, respectively.
- $\underline{o}^p$ is rendered by applying the rendering algorithm recursively to $M(o)$ and p.
- $\texttt{for}(\underline{l}, I, \rho_1 \ \ldots \ \rho_r)\{\rho'_1 \ \ldots \ \rho'_s\}$ is rendered by the following algorithm:
  1. Let $sep := {\rho_1}^M + \ldots + {\rho_r}^M$ and t be the length of $M(l)$.
  2. For $i = 1, \ldots, t$, let $R_i := {\rho'_1}^{M^l_i} + \ldots + {\rho'_s}^{M^l_i}$.

3. If $I = 0$, return nothing and stop. If I is negative, reverse the list R, and invert the sign of I.
4. Return $R_I + sep + R_{2*I} \ldots + sep + R_T$ where T is the greatest multiple of I smaller than or equal to t.

Here the match context M_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 Sect. 2. There we have

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

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

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

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

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

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

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

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 Choosing Notation Definitions Extensionally

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

4.1 Collecting Notation Definitions

The algorithm for the collection of notation definitions 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 [NW01b], we use the source names EC, F, Doc, CD, and SD 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 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** of the 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.

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 preserving the order of the $(\lambda^{\cdot}\rho)^p$ (see a concrete example below).

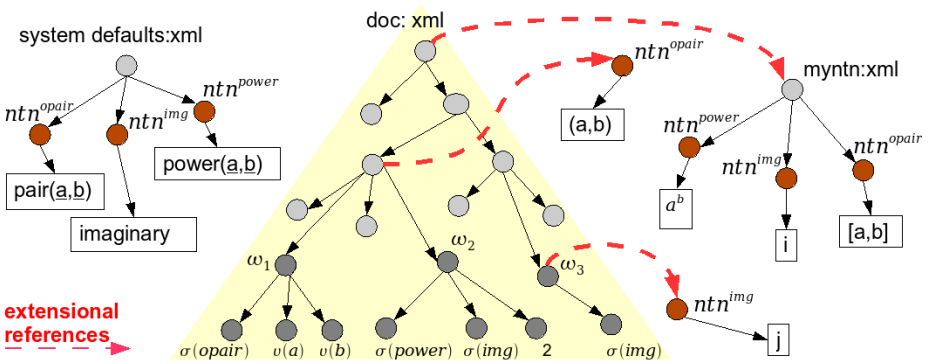

Fig. 2. Collection Example

We base our further illustration on the input document in Fig. 2 figure above, which includes three mathematical objects. For simplicity, we omit the `cdbase` and `cd` attributes of symbols.

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

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 definition 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 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 inner-most.

4.2 Discussion of Collection Strategies

In [KLM+08], we provide the specific algorithms for collecting notation definitions from EC, F, Doc, CD and SD and illustrate the advantages and drawbacks of basing the rendering on either one of the sources. We conclude with the following findings:

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. 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.

5 Choosing Renderings Intensionally

The extensional notation context declarations do not support authors to select between alternative renderings inside one notation definition. Consequently, if relying only on this mechanism, authors have to take extreme care about which notation definition they reference or embed. Moreover, other users cannot change the granular extensional declarations in EC without modifying the input document. They can only overwrite the author's granular specifications with their individual styles F, which may reduce the understandability of the document.

Consequently, we need a more *intelligent, context-sensitive* selection of renderings, which lets users guide the selection of alternative renderings. 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.

5.1 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 $(\lambda : \rho)^p$, 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 analogous 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 (see [KLM+08] for more details).

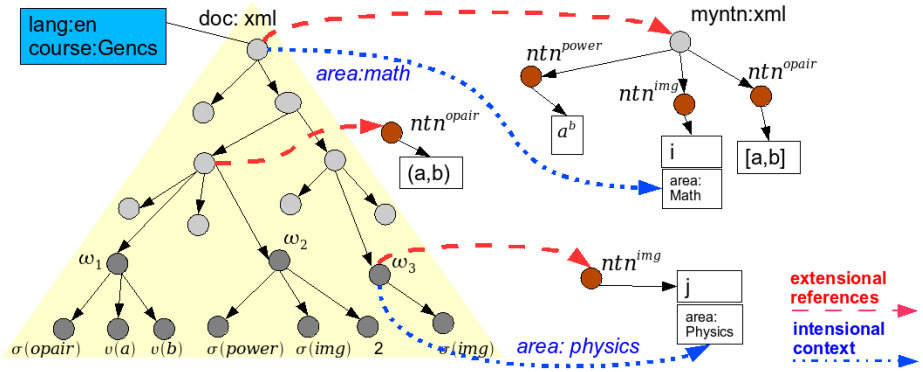

Fig. 3. Rendering Example

In Fig. 3, 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 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.

5.2 Discussion of Context-Sensitive Selection Strategies

In [KLM+08], we illustrate and evaluate the collection of contextual information from GC, CCF, IC and MD in detail. We conclude with the following findings:

1. The declaration of a *global context* provides a more intelligent intensional selection between alternative $(\lambda : \rho)^p$ 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 any node in the input document and cannot be overwritten without modifying the input document. Moreover, the available context dimensions and values are limited by the respective metadata format.
3. The *intensional context* supports a granular selection of renderings by associating an intensional context to any node of the input document. However, the intensional references cannot be overwritten on granular document levels.
4. *Cascading Context Files* permit a granular overwriting of contexts.

6 Conclusion and Future Work

We introduced a representational infrastructure for notations in living mathematical documents by providing a flexible declarative specification language for notation definitions together with a rendering algorithm. We described how authors and users can extensionally extend the set of available notation definitions on granular document levels, and how they can guide the notation selection via intensional context declarations. Moreover, we discussed different approaches for collecting notation definitions and contextual information.

To substantiate our approach, we have developed prototypical implementations of all aspects of the rendering pipeline:

– The Java toolkit `mmlkit` [MMK07] implements the conversion of OpenMath and Content-MathML expressions to Presentation-MathML. It supports the collection of *notation definitions* from various sources, constructs rendering contexts based on contextual annotations of the rendered object, identifies proper renderings for the conversion.

– The semantic wiki SWiM [Lan08] supports the collaborative browsing and editing of notation definitions in OpenMath content dictionaries.
– The *panta rhei* [Mül07] reader integrates mmlkit to present mathematical documents, provides facilities to categorize and describe notations, and uses these context annotations to adapt documents.

We will invest further work into our implementations as well as the evaluation of our approach. In particular, we want to address the following challenges:

– Write Protection: In some cases, users should be prevented to overwrite the author's declaration. On the contrary, static notations reduce the flexibility and adaptability of a document (see [KLM+08] for more details).
– Consistency: The flexible adaptation of notations can destroy the meaning of documents, in particular, if we use the same notation to denote different mathematical concepts.
– Elision: In [KLM+08], we have already adapted the elision of arbitrary parts of formulae from [KLR07].
– Notation Management: Users want to reuse, adapt, extend, and categorize notation definitions (see [KLM+08] for more details).
– Advanced notational forms: Ellipses and Andrews' dot are examples of advanced notations that we cannot express yet.

Acknowledgments Our special thanks go to Normen Müller for the initial implementation of the presentation pipeline. We would also like to thank Alberto Gonzáles Palomo and Paul Libbrecht for the discussions on their work. This 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. Available at http://www.w3.org/TR/MathML2.

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

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

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

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

KLM+08. Michael Kohlhase, Christoph Lange, Christine Müller, Normen Müller, and Florian Rabe. Adaptation of notations in living mathematical documents. KWARC report, Jacobs University Bremen, 2008.

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.

Lan08. 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*, Costa Adeje, Tenerife, Spain, June 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.

MMK07. Christine Müller, Normen Müller, and Michael Kohlhase. mmlkit - a toolkit for handling mathematical documents and MathML3 notation definitions. mmlkit v0.1 at `http://kwarc.info/projects/mmlkit`, seen November 2007.

Mül07. Christine Müller. Panta Rhei. Project Home Page at `http://kwarc.info/projects/panta-rhei/`, seen August 2007.

NW01a. 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.

NW01b. 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* [NW01a].

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

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, 2006.

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/.