VMEXT: A Visualization Tool for Mathematical Expression Trees

Moritz Schubotz[1], Norman Meuschke[1], Thomas Hepp[1], Howard S. Cohl[2], and Bela Gipp[1]

[1] Dept. of Computer and Information Science, University of Konstanz, Box 76, 78457 Konstanz, Germany
{first.last}@uni-konstanz.de, www.isg.uni-konstanz.de

[2] Applied and Computational Mathematics Division, National Institute of Standards and Technology Gaithersburg, Maryland 20899-8910, USA
howard.cohl@nist.gov, www.nist.gov/people/howard-cohl

Abstract. Mathematical expressions can be represented as a tree consisting of terminal symbols, such as identifiers or numbers (leaf nodes), and functions or operators (non-leaf nodes). Expression trees are an important mechanism for storing and processing mathematical expressions as well as the most frequently used visualization of the structure of mathematical expressions. Typically, researchers and practitioners manually visualize expression trees using general-purpose tools. This approach is laborious, redundant, and error-prone. Manual visualizations represents a user's notion of what the markup of an expression should be, but not necessarily what the actual markup is. This paper presents VMEXT – a free and open source tool to directly visualize expression trees from parallel MathML. VMEXT simultaneously visualizes the presentation elements and the semantic structure of mathematical expressions to enable users to quickly spot deficiencies in the Content MathML markup that does not affect the presentation of the expression. Identifying such discrepancies previously required reading the verbose and complex MathML markup. VMEXT also allows one to visualize similar and identical elements of two expressions. Visualizing expression similarity can support support developers in designing retrieval approaches and enable improved interaction concepts for users of mathematical information retrieval systems. We demonstrate VMEXT's visualizations in two web-based applications. The first application presents the visualizations alone. The second application shows a possible integration of the visualizations in systems for mathematical knowledge management and mathematical information retrieval. The application converts LaTeX input to parallel MathML, computes basic similarity measures for mathematical expressions, and visualizes the results using VMEXT.

Keywords: Mathematical Information Retrieval, Expression Tree, LaTeX, MathML, Visualization

1 Introduction

Mathematical notation strives to have a well-defined vocabulary, syntax, and semantics. Similar to sentences in natural language or constructs in a programming language, mathematical expressions consist of constituents that have a coherent meaning, such as terms or functions. We consider a mathematical expression to be any sequence of mathematical symbols that can be evaluated, e.g., typically formulae. The syntactic rules of mathematical notation, such as operator precedence and function scope, determine a hierarchical structure for mathematical expressions, which can be understood, represented, and processed as a tree. *Mathematical expression trees* consist of functions or operators and their arguments. Experiments by Jansen, Marriott, and Yelland suggest that mathematicians use some notion of mathematical expression trees as a mental representation to perform mathematical tasks [JMY00].

Describing and processing mathematical content using expression trees is established practice in mathematics and computer science as our review of related work in Section 2 shows. However, no standard for the content of nodes, or the structure and visual representation of such trees has yet emerged. Additionally, we did not find tools that support generating expression tree visualizations from mathematical markup. All of the visualizations that we were able to glean from the literature were manually created using general purpose tools.

With this paper, we seek to contribute to the establishment of an openly available, widely accepted, visualization of mathematical expression trees, encoded using the MathML standard. For this purpose, we propose a tree visualization that operates on parallel MathML markup and provides the visualization as a free and open source tool. We structure the presentation of our contributions as follows. Section 2.1 presents details of the MathML standard that serves as the data structure for our visualization approach. Section 2.2 reviews the strength and weaknesses of existing approaches for visualizing mathematical expression trees to derive our visualization concept. Section 3 present our visualization tool VMEXT. Section 3.3 describes a demo application that shows how the visualization can be integrated into other applications. Section 3.4 explains how end users and developers can apply and obtain VMEXT. Section 4 concludes the paper by discussing our plans for further extending and improving VMEXT.

2 Related Work

As briefly motivated in the previous section, we seek to reduce the effort for researchers and practitioners to generate expression tree visualizations for mathematical content. Additionally, we hope to contribute to establishing a standardized representation of mathematical expression trees. In Section 2.1, we present the MathML standard and explain why we see it as a promising data format to achieve this goal. In Section 2.2, we review existing approaches for visualizing mathematical expression trees to explain how we derived the major building blocks of our visualization approach.

2.1 MathML

Mathematical Markup Language (MathML) is a W3C[3] and ISO standard (ISO/IEC DIS 40314) for representing mathematical content using XML syntax. MathML is part of HTML5 and enables one to serve, receive, and process mathematical content on the World Wide Web. MathML allows users to describe the notation and/or the meaning of mathematical content using two vocabularies: Presentation MathML (PMML) and Content MathML (CMML). The vocabularies can be used independently of each other or in conjunction.

Presentation MathML focuses on describing the visual layout of mathematical content. The PMML vocabulary contains elements for basic mathematical symbols and structures. Each element specifies the role of the presentation element, e.g., the element `<mi>` represents identifiers and the element `<mo>` represents operators. The structure of PMML markup reflects the two-dimensional layout of the mathematical expression. Elements that form semantic units are encapsulated in `<mrow>` elements, which are comparable to `<div>` elements in HTML. Listing 1.1 exemplifies PMML markup for the expression $f(a+b)$.

Content MathML focuses on explicitly encoding the semantic structure and the meaning of mathematical content using expression trees. In other words, the CMML vocabulary seeks to specify the frequently ambiguous mapping from the presentation of mathematical content to its meaning. For example, the presentation of the expression $f(a+b)$ represents two possible syntactic structures: e.g., f could represent either an identifier or a function. CMML uses `<apply>` elements to make explicit which elements represent functions. Subordinate elements represent the arguments of the functions. Listing 1.2 illustrates CMML markup for the expression $f(a+b)$.

```
<math xmlns=”http://www.w3.org/1998/Math/MathML”>
  <semantics>
    <mrow id=”r1”>
      <mi id=”i1”>f</mi>
      <mo id=”o1”>(</mo>
      <mrow id=”r2”>
        <mi id=”i2”>a</mi>
        <mo id=”o2”>+</mo>
        <mi id=”i3”>b</mi>
      </mrow>
      <mo id=”o3”>)</mi>
    </mrow>
```

Listing 1.1. Presentation MathML encoding of the expression $f(a+b)$ [Sch17]

Content MathML offers two subsets of elements to specify function types: Pragmatic Content MathML and Strict Content MathML. Pragmatic Content MathML uses a large set of predefined functions encoded as empty elements, e.g., `<plus/>`, as used in Line 17 in Listing 1.2, or `<log/>` for the logarithm. Strict Content

[3] www.w3.org/Math/

```
13    <annotation-xml encoding="MathML-Content">
14      <apply xref="r1">
15        <ci xref="b">f</ci>
16        <apply xref="r2">
17          <plus xref="o2"/><!-- <csymbol cd
              ="arith1">plus</csymbol> in strict encoding -->
18          <ci xref="i2">a</ci>
19          <ci xref="i3">b</ci>
20        </apply>
21      </apply>
22    </annotation-xml>
```

Listing 1.2. Content MathML encoding of the expression $f(a+b)$ [Sch17]

MathML uses a minimal set of elements, which are further specified by referencing extensible content dictionaries. For example, the plus operator (+) is defined in the content dictionary `arith1`. Using Strict CMML, the operator is encoded using the element for symbols <`csymbol`>, and declaring that the specification of the symbol is available under the term `plus` in the content dictionary `arith1`. Line 17 in Listing 1.2 shows this option of specifying the plus operator as a comment (green font color).

As described above, the PMML and CMML vocabularies can be used individually and independent of each other. For example, PMML is frequently used without content markup to display mathematical content on websites. CMML without presentation markup can, for instance, be used to exchange data between computer algebra systems. However, PMML and CMML markup can also be used in conjunction to simultaneously describe the presentation, structure, and semantics of mathematical expressions. The combined use of PMML and CMML is commonly referred to as parallel MathML.

In parallel MathML markup, presentation and content elements are mutually interlinked by including `xref` arguments that point to the corresponding element in the other vocabulary. The PMML and CMML markup in Listing 1.1 and Listing 1.2 respectively contain `xref`-links to create parallel MathML.

2.2 Expression Tree Visualizations

Researchers, especially in math information retrieval (MIR), have employed several use-case-specific tree visualizations for mathematical expressions. All visualizations appear to have been created manually to illustrate research in publications. The content and structure of the visualizations vary significantly. Figure 1 and Figure 2 give an overview of the visualizations, which we describe hereafter.

Youssef and Shatnawi use simple ASCII graphics to visualize expression trees. Their visualization resembles binary expression trees. Leaf nodes represent identifiers or numbers; inner nodes represent operators, functions, or brackets [YS06].

In later work, Shatnawi and Youssef replace the ASCII graphics with an equivalent chart. Altamimi and Youssef further improve their visualization by marking subexpression groups with dashed lines (see Figure 1, b) [AY08].

Miner and Munavalli use a different tree to illustrate their research on math search. They render the full expression in the root of the tree and create sub-nodes for each sub-expression (see Figure 1, c) [MM07]. Sojka and Líška use a similar visualization to illustrate the tokenization and indexing process of their math search system.

Hashimoto, Hijikata, and Nishida use a tree layout that represents the DOM structure of Presentation MathML markup to illustrate the author's research on MathML indexing [HHN08]. In this layout, inner nodes represent MathML elements depicted as circles and leaf nodes represent the content of elements depicted as squares (see Figure 1, d). We assume the authors manually created the visualization, since the focus of their paper is on math search and does not mention an automated visualization approach.

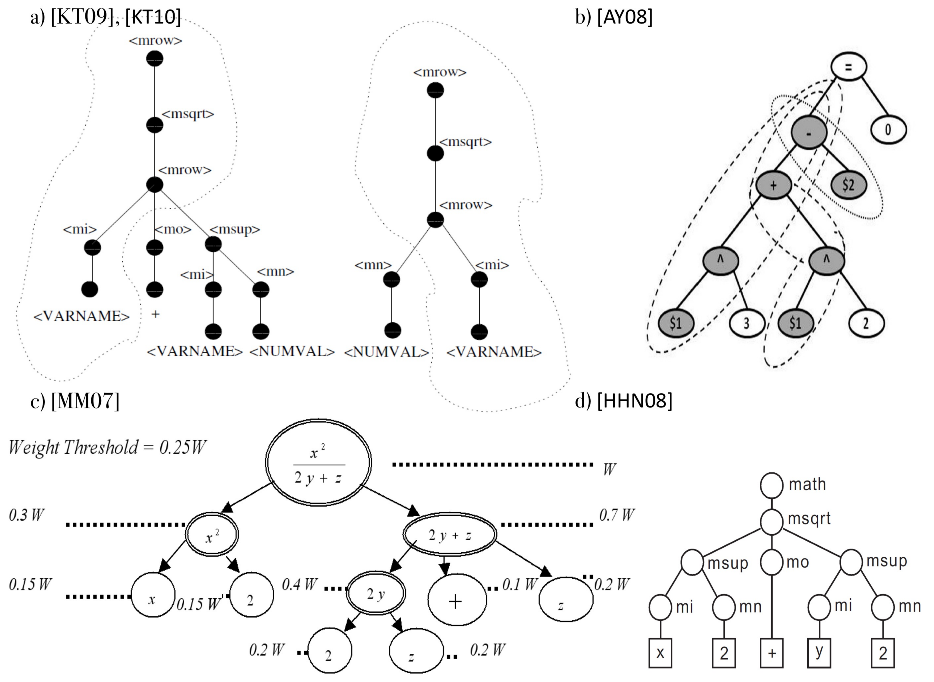
Kamali and Tompa [KT09] and Kamali and Tompa [KT10] use a similar tree representation of the Presentation MathML structure in their works on math similarity and retrieval. Their visualization does not distinguish between inner nodes and leaf nodes, but depicts all nodes as circles (see Figure 1, a). Two things are notable about this visualization. First, the layout corresponds to the data structure of the mathematical expressions. Second, Kamali and Tompa introduce the notion of defining and visualizing the similarity of mathematical expressions in terms of the structural similarity of sub-trees. The authors visually indicate similar sub-trees by enclosing the respective sub-tree in a dashed line (see Figure 1, a). In subsequent work, Kamali and Tompa [KT13] use a horizontal layout to visualize the same tree. The tree uses boxes instead of circles and directed instead of undirected edges. Kamali and Tompa exclusively consider PMML and do not present an automated approach to create their visualization of the structure and similarity of PMML expressions.

Yokoi and Aizawa consider Content MathML markup for their research on math similarity search and devise a visualization of the CMML tree [YA09]. Their work introduces apply-free content markup, i.e., omitting the first `<apply>` element in the CMML markup, since it provides little information on the applied function. Instead, their markup uses the first child of an `<apply>` element. Their manually created visualization also omits `<apply>` elements (see Figure 2, a). We consider this approach valuable, since it reduces the number of nodes to visualize and facilitates the recognition of function types.

Hagino and Saito also consider apply-free Content MathML markup for their research on partial match retrieval in math search [HS13]. To illustrate their research, they use a tree that depicts the CMML element names in the case of inner nodes and the CMML element names in combination with the elements' content in the case of leaf nodes (see Figure 2, b).

In their review of approaches for math recognition and retrieval, Zanibbi and Blostein point out that building a symbol layout tree is important for math recognition tasks [ZB12]. Symbol layout trees represent horizontally adjacent symbols

Fig. 1. Overview of expression tree visualizations part 1



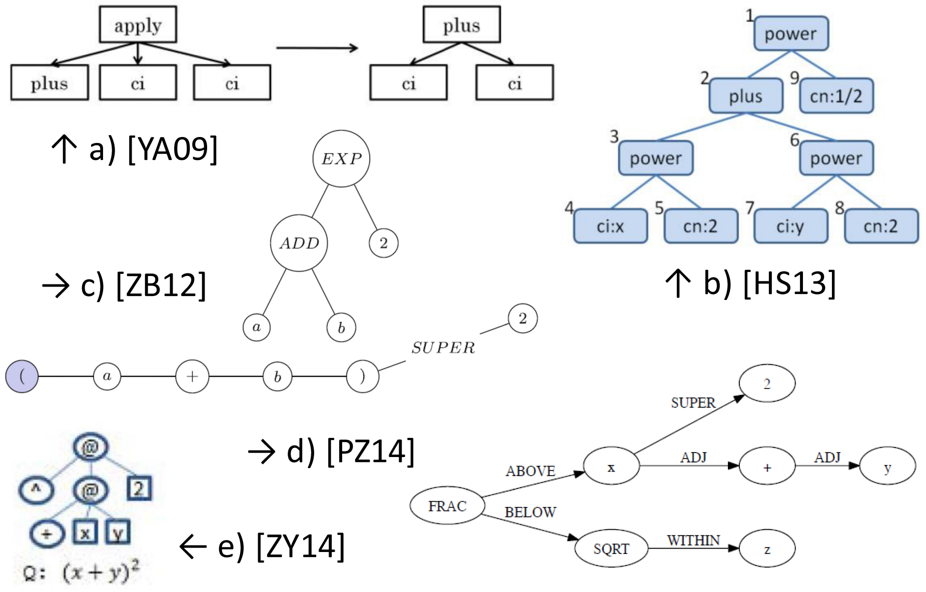
that share a writing line and indicate subscript, superscript, above, below, and containment relationships. The authors present a horizontal illustration of the symbol layout tree and a simplified expression tree using a vertical layout (see Figure 2, d). Pattaniyil and Zanibbi uses a similar horizontal illustration of the symbol layout tree (see Figure 2, e) [PZ14].

Zhang and Youssef use Strict Content MathML for their research [ZY14]. In their visualizations of the CMML tree, they omit the element names for <`ci`> and <`cn`> elements, but include <`apply`> elements. They replace the names of CMML elements with shorter symbols. For instance, they replace <`apply`> with `@` and <`power`> with ^.

2.3 Summary of Related Work and Research Gap

From our review of the literature, we draw the following conclusions. First, representing mathematical expressions as trees is essential for performing many tasks in mathematical knowledge management (MKM) and mathematical information retrieval (MIR). Expression trees, in which leaf nodes represent terminal symbols and inner nodes represent operators, functions, or brackets are widely used as a data representation. The MathML standard is a well-established data format for

Fig. 2. Overview of expression tree visualizations part 2



representing the presentation, structure, and semantics of mathematical content using the expression tree concept. Many researcher rely on MathML encoded content for MIR and MKM tasks.

Second, researchers frequently employ expression tree visualizations to illustrate their math-related research. While some visualizations reflect the information extracted from mathematical markup, such as MathML, other visualizations illustrate abstract mathematical expressions. The elements included in the visualizations, their spatial arrangement, and visual appearance varies greatly. Depending on the use case, visualizations may include presentation elements, content elements, or combinations thereof. Especially in the MIR domain, researchers frequently need to visualize similarity of operator (sub-)trees.

Third, although the expression tree concept is at the heart of MathML and visualizations of MathML markup are widely used for analysis and presentation purposes, we found no tool that generates such visualizations from MathML markup. Researchers typically create expression tree visualizations manually using general-purpose tools. This approach results in much manual and redundant effort, diverse visual representations of identical markup, and the danger of creating a visualization that does not reflect the underlying data. To reduce the effort for creating expression tree visualizations and to contribute towards establishing a more canonical design of expression trees, we present the VMEXT system, which we describe in the following section.

3 VMEXT System

VMEXT is an acronym for Visualizing Mathematical Expression Trees. This tool seeks to visually support researchers and practitioners in two well-defined use cases:

1. curating semantically enriched mathematical content, e.g., for use in digital repositories or systems for mathematical knowledge management;
2. examining similarities of two mathematical expressions, e.g., for developing mathematical information retrieval approaches or for examining and interacting with the results of MIR systems.

VMEXT addresses the use cases with two visualizations available as widgets that can easily be integrated into any web application. We present the widgets in Section 3.1 and Section 3.2. Both widgets are available as a demo system at: `http://vmext.formulasearchengine.com/`. Section 3.3 presents a demo application that exemplifies the possible use of the widgets as part of MKM and MIR systems. Section 3.4 describes how interested parties may use VMEXT's visualizations; integrate the visualizations as widgets or via an API into their own applications; and how to adapt and extend the code.

3.1 Curating Semantically-enriched Mathematical Content

Making mathematical knowledge accessible through recognition, retrieval, and management systems is a task that has attracted many contributions by researchers and practitioners. (Guidi and Sacerdoti Coen [GS16] and Zanibbi and Blostein [ZB12] present comprehensive reviews on the topic). The MathML standard (see Section 2.1) has been widely adopted to expose both the presentation and semantics of mathematical content for such systems.

However, the MathML syntax is verbose, complex and therefore not easy to grasp for humans. Furthermore, creating parallel MathML markup is complicated and error-prone. This is true, especially for the creation of parallel MathML by converting other formats, such as LaTeX, and often results in ambiguous or erroneous markup. Typically, Presentation MathML elements are less frequently affected by errors than their respective Content MathML elements. This leads to a situation, in which the visual representation of an expression is correct, yet its semantics are wrong.

VMEXT supports users in quickly checking and improving parallel MathML by providing an interactive expression tree visualization that simultaneously illustrates the semantic structure (as well as the presentation elements) encoded in the markup.

VMEXT visualizes the structure of the tree as encoded in the Content MathML markup. However, the labels for each node render the Presentation MathML elements linked to the respective content elements. VMEXT uses the apply-free CMML notation introduced in [YA09]. In other words, our parser renders the first child of each `<apply>` element, not the `<apply>` itself, as an operator or function. All following children are considered as arguments of the function. For a clear layout, VMEXT renders the complete PMML element for the first child, even if the first child is itself an `<apply>` element. To reduce the size of the individual edges, we replace some CMML elements with shorthand symbols, e.g., we replace `<power>` with $\wedge$ as can be seen in Figure 3 (cf. [ZY14], see also Section 2).

To facilitate human inspection, VMEXT follows the information seeking mantra proposed by Shneiderman [Shn96]: *overview first*, *zoom and filter*, then *details-on-demand*. The nodes in VMEXT can be interactively *filtered* by expanding or collapsing nodes either one at a time or all at once using the expand button. The view-port is adjustable using *pan* and *zoom* interactions to enable focusing on specific parts of the tree. The resize button resets the zoom level. User *navigation* is supported through an overview infix expression rendered at the top of the screen. Hovering over parts of the infix expression or nodes in the tree, highlights the corresponding parts in the tree and the infix expression. subsection 3.2 shows how hovering over the divide operator highlights the respective sub-tree in light blue. The user can *export* the chosen (sub-)tree rendering, including all manipulations performed through filtering and zooming, as a high-resolution `png` image, e.g., for use in publications.

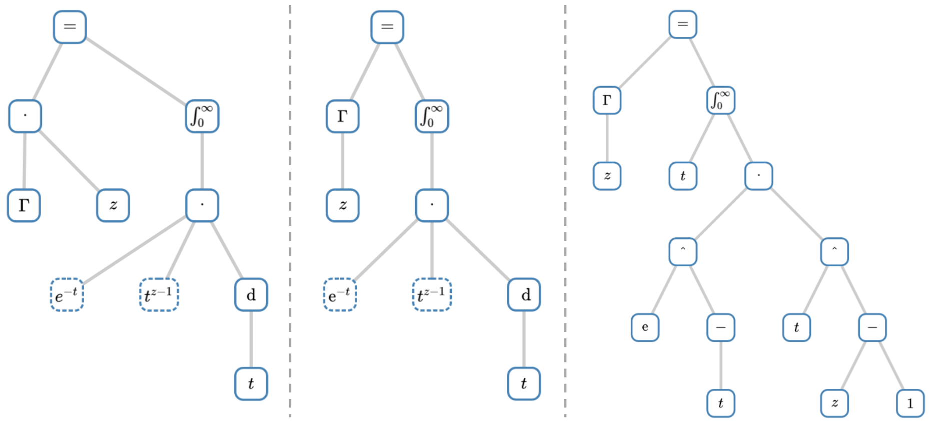
To demonstrate how VMEXT's expression tree visualization can aid in curating semantically enriched MathML markup, we use the integral representation of the Euler gamma function [Olv+, (5.2.1)] as an example

$$\mathit{\Gamma}(z) = \int_0^\infty e^{-t} t^{z-1} \mathrm{d}t. \tag{1}$$

Figures 3 a-c show VMEXT's rendering for three markup variants of the Euler gamma function. All variants have identical PMML markup, i.e., produce identical visual output as shown in Equation 1. However, the CMML differs, because we generated the MathML using LaTeXML [Mil15] using different LaTeX input (shown in the captions of the figures). Note, that these different LaTeX versions encode more or less semantics.

The trees in Figures 3 a and b show that VMEXT allows an arbitrary number of child nodes, as opposed to the binary expression tree concept we briefly described in Section 1. The conversion of generic LaTeX input (a), misinterpreted some invisible operators, such as the invisible operator between $\mathit{\Gamma}$ and (z) that was interpreted as times rather than a function application. Additionally, LaTeXML marked some CMML elements as ambiguous, i.e., could not establish a one-to-one relation to a PMML element. For ambiguous nodes, VMEXT renders all PMML elements enclosed by the ambiguous CMML element in a node with dashed borders to emphasize the defective markup for the user. For example, the node for e^{-t} in Figure 3 was marked as ambiguous.

The LaTeX representation using DLMF macros (b) resolves the problem of invisible operators by using the @ symbol to make such operators explicit. However, this representation still results in ambiguous nodes. Representing the Euler gamma function using DLMF and DRMF macros [Coh+14; Coh+15] results in correct CMML markup. In (c), we specify the integral using the semantic macro



a) generic LaTeX:
```
\Gamma\!\left(z\right)=
\int_{0}^{\infty}
e^{-t}t^{z-1}\mathrm{d}t
```

b) DLMF LaTeX macros:
```
EulerGamma@{z}=
\int_0^\infty
\expe^{-t}t^{z-1}\diff{t}
```

c) DLMF, DRMF LaTeX macros:
```
EulerGamma@{z}=
\Int{0}{\infty}@{t}
{\expe^{-t} t^{z-1}}
```

Fig. 3. Expression trees rendered for MathML input obtained from converting different LaTeX input. The Presentation MathML is identical for all three cases, yet the Content MathML differs.

`\Int` rather than the generic `\int` command. We have required that all occurrences of the ∧-operator must denote the power operator. Note that, in order to make this workable, one must create beneficial custom semantic macros for all other uses of the ∧-operator. These include matrix operations ($A^\dagger$), labeling (x^*), function spaces (C^k), norms (L^p), sums ($\sum_{n=0}^\infty$), products ($\prod_{n=0}^\infty$), derivatives ($f^{(2)}(x)$), etc.

By rendering the expression tree as encoded by the CMML markup, VMEXT enables users to quickly spot markup deficiencies and illuminates the effects of using different conversions or manually changing markup.

3.2 Examining Similarities of Mathematical Expressions

Our review of MIR literature (see Section 2.2) shows that researchers often seek to visualize the similarity of two mathematical expressions, e.g., the similarity between a query expression and a retrieval candidate. To facilitate this task, VMEXT includes a specialized visualization shown in subsection 3.2. The presented example compares two notations of the measure Mean Reciprocal Rank.

The widget accepts CMML input for the expressions to compare. Similar elements can be specified by stating the IDs of the similar CMML elements in both trees using JSON. Currently, VMEXT allows one to specify that elements are either similar or identical. The two types of similarity are rendered differently. Since VMEXT is designed to be a visualization tool, it includes no functionality to compute similarities. We demonstrate the integration of the widgets with a basic application that computes similarities in Section 3.3.

The center view renders the trees (including the infix overview) for both expressions and visually distinguishes the trees using different background colors. The visualizations offer the same interaction features as the expression tree widget (see Section 3.1). In the lower part of the center view, VMEXT renders a combined expression tree. The combined tree includes all nodes from both trees color-coded with the background color of the tree from which they originate. Unique, i.e., dissimilar, sub-trees of both trees are collapsed to direct the user's attention to the similar parts of the trees. For elements marked as similar, VMEXT renders the nodes from both trees and highlights them as exemplified by the nodes MRR and MMR. Nodes that are marked as identical are rendered only once and are highlighted as exemplified by the node $\sum_{i=1}^{|Q|} \frac{1}{r}$.

The integrated visualization of the two expression trees and the combined tree, allows users to quickly inspect the full structure of both expressions and similar sub-trees. The highlight on hover feature helps users to look up the corresponding subtrees for nodes marked as similar in the combined tree.

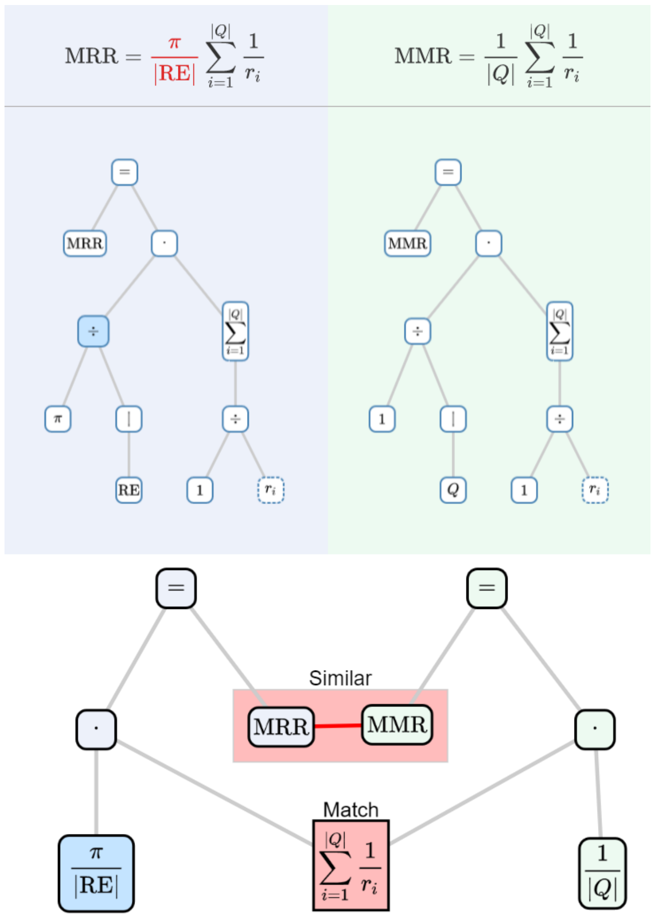
A specific application that benefits from visualizing the similarity of mathematical expressions is our prototype of a hybrid plagiarism detection system CitePlag[4] [MGB12; Gip+13]. Forms of academic plagiarism vary greatly in their degree of obfuscation ranging from blatant copying to strongly disguised idea plagiarism [MG13]. Our research indicates that not a single, but combined PD approaches are most promising to reliably detect the wide range of plagiarism forms [GMB14; Gip+14; Gip14]. Combined approaches accumulate evidence on potentially suspicious similarity using heterogeneous features, such as literally matching text, similarities in the citations used, and similarity of mathematical content [MG14]. CitePlag is the first system to implement such an integrated analysis and uses the VMEXT framework to visualize the similarity of mathematical content.

3.3 Demo Application

To showcase a possible integration of VMEXT's widgets into MIR and MKM applications, we developed a Java application for input conversion and similarity computation. The demo provides a basic web frontend available at: `http://vmext-demo.formulasearchengine.com` and offers two main features.

First, it converts LaTeX input to parallel MathML. The backend of the demo application offers two alternative converters. The first converter employs LaTeXML, whose configuration can be customized via input fields included in the web frontend. The second converter passes the LaTeX input to the Mathoid system[5] [SW14], which employs the speech rule engine[6] [CKS15] to generate Presentation MathML with **CDATA** annotations. These annotations give hints on the possible semantic meaning of expressions. Using a simple XSLT stylesheet, the demo application converts this non-standard-conforming markup to standard parallel MathML markup. The application enables users to quickly run different LaTeX to MathML

[4] `http://www.citeplag.org`
[5] `https://www.mediawiki.org/wiki/Mathoid`
[6] `https://github.com/zorkow/speech-rule-engine`



$\mathrm{MRR} = \frac{\pi}{|\mathrm{RE}|}\sum_{i=1}^{|Q|}\frac{1}{r_i}$
$\mathrm{MMR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{r_i}$
Similar
MRR
MMR
Match

Fig. 4. VMEXT Expression Tree Similarity Widget

conversions and immediately examines the effects on the conversion quality using the VMEXT visualizations described in Section 3.1 and Section 3.2.

Second, the demo application computes basic similarity measures for two expressions. The most basic measure identifies identical nodes. A second measure uses the idea of taxonomic distance of expressions proposed in [ZY14]. Our implementation uses content dictionaries to model the taxonomic distance and builds upon the content dictionary abstraction as introduced in [Sch+14]. The system converts the CMML markup of the expression to Strict CMML to guarantee that the XML encodings of all symbols explicitly state from which content dictionary the symbols originate. All symbols originating from the same content dictionary, like plus and minus, or sine and cosine, are considered similar. Symbols from different content dictionaries, e.g., plus and cosine, are considered dissimilar. The objective of the similarity computation is to provide users with test data to explore the visualization approaches, and not to be meaningful from an analytical perspective.

3.4 Obtaining VMEXT

VMEXT is a free and open source JavaScript application. We host a ready-to-use instance of the tool at: `http://vmext.formulasearchengine.com`. We also provide a REST API that exposes the image export functionality and the internal representation of our visualization.

The demo application for converting and rendering LaTeX markup (see Section 3.3) is available at: `http://vmext-demo.formulasearchengine.com`.

For development purposes,VMEXT is available as a Node.js package from: `https://www.npmjs.com/package/vmext`. We actively maintain and enhance the tool; the latest code is available from `https://github.com/ag-gipp/vmext`. Pull requests and bug reports are highly welcome.

4 Conclusion and Future Work

In this paper, we present two tree-based visualization approaches for mathematical expressions. The first approach simultaneously illustrates the presentation, structure, and semantics of individual expressions. The second approach visualizes the structural and semantic similarity of two expressions. Both approaches operate on parallel MathML markup and incorporate key elements of expression tree visualizations proposed in the MIR literature.

We implemented the two approaches as part of VMEXT, a system we provide free and open source for end users and developers (see Section 3.4). Additionally, we provide two web-based demo applications. The first application[7] presents the visualization widgets alone. The second application[8] demonstrates a possible integration of the widgets in systems for mathematical knowledge management and mathematical information retrieval.

In our future work, we plan to extend VMEXT's functionality beyond exclusively visualizing MathML markup towards visually assisting markup creation and editing by humans. MathML shows great promise for enabling unprecedented access to mathematical knowledge. However, converting existing mathematical knowledge to semantic markup formats will require some human interaction. The complexity and verbosity of MathML makes direct interaction with MathML markup laborious and time-consuming. We see visual editors as a possible solution to this problem. Enabling users to create and manipulate mathematical notation and MathML markup via visual support tools would be valuable for increasing the digital accessibility of mathematical knowledge [CS17; Sch+16]. Another possible extension is the consideration of proof structures and the visualization of the directed acyclic graphs, which might occur, if the MathML `<share />` element is used.

[7] `http://vmext.formulasearchengine.com`
[8] `http://vmext-demo.formulasearchengine.com`

Acknowledgments We thank Ludwig Goohsen and Stefan Kaufhold for their support in developing VMEXT. Furthermore, we thank the Wikimedia Foundation for providing a server to run the VMEXT demo.

References

[AY08] M. E. Altamimi and A. S. Youssef. A Math Query Language with an Expanded Set of Wildcards. In: Math.Comput.Sci. (2008), pp. 305–331. DOI: 10.1007/s11786-008-0056-4.

[CKS15] D. Cervone, P. Krautzberger, and V. Sorge. Towards Meaningful Visual Abstraction of Mathematical Notation. In: Proc. CICM. 2015.

[Coh+14] H. S. Cohl et al. Digital Repository of Mathematical Formulae. In: Proc. CICM. Springer, 2014, pp. 419–422. DOI: 10.1007/978-3-319-08434-3_30.

[Coh+15] H. S. Cohl et al. Growing the Digital Repository of Mathematical Formulae with Generic LaTeX Sources. In: Proc. CICM. Springer, 2015, pp. 280–287. DOI: 10.1007/978-3-319-20615-8_18.

[CS17] J. Corneli and M. Schubotz. math.wikipedia.org: A vision for a collaborative semi-formal, language independent math(s) encyclopedia. In: Proc. CAITP. 2017.

[Gip+13] B. Gipp et al. Demonstration of the First Citation-based Plagiarism Detection Prototype. In: Proc. SIGIR. 2013, pp. 1119–1120. DOI: 10.1145/2484028.2484214.

[Gip+14] B. Gipp et al. Web-based Demonstration of Semantic Similarity Detection using Citation Pattern Visualization for a Cross Language Plagiarism Case. In: Proc. Int. Conf. on Enterprise Information Systems. 2014, pp. 677–683. DOI: 10.5220/0004985406770683.

[Gip14] B. Gipp. *Citation-based Plagiarism Detection - Detecting Disguised and Cross-language Plagiarism using Citation Pattern Analysis.* Springer Vieweg Research, 2014.

[GMB14] B. Gipp, N. Meuschke, and C. Breitinger. Citation-based Plagiarism Detection: Practicability on a Large-scale Scientific Corpus. In: JASIST (2014), pp. 1527–1540. DOI: 10.1002/asi.23228.

[GS16] F. Guidi and C. Sacerdoti Coen. A Survey on Retrieval of Mathematical Knowledge. In: Mathematics in Computer Science (2016), pp. 409–427. DOI: 10.1007/s11786-016-0274-0.

[HHN08] H. Hashimoto, Y. Hijikata, and S. Nishida. Incorporating breadth first search for indexing MathML objects. In: Proc. SMC. IEEE, 2008, pp. 3519–3523. DOI: 10.1109/ICSMC.2008.4811843.

[HS13] H. Hagino and H. Saito. Partial-match Retrieval with Structure-reflected Indices at the NTCIR-10 Math Task. In: Proc. NTCIR-10. National Institute of Informatics, 2013.

[JMY00] A. R. Jansen, K. Marriott, and G. W. Yelland. Constituent Structure in Mathematical Expressions. In: CogSci vol. 22. 2000.

[KT09] S. Kamali and F. W. Tompa. Improving mathematics retrieval. In: Proc. DML (2009), pp. 37–48.

[KT10] S. Kamali and F. W. Tompa. A new mathematics retrieval system. In: Proc. CIKM. ACM, 2010, pp. 1413–1416. DOI: 10.1145/1871437.1871635.

[KT13] S. Kamali and F. W. Tompa. Structural Similarity Search for Mathematics Retrieval. In: Proc. CICM. Springer, 2013, pp. 246–262. DOI: 10.1007/978-3-642-39320-4_16.

[MG13] N. Meuschke and B. Gipp. State of the Art in Detecting Academic Plagiarism. In: Int. J. for Educational Integrity (2013), pp. 50–71.

[MG14] N. Meuschke and B. Gipp. Reducing Computational Effort for Plagiarism Detection by using Citation Characteristics to Limit Retrieval Space. In: Proc. JCDL. 2014, pp. 197–200. DOI: `10.1109/JCDL.2014.6970168`.

[MGB12] N. Meuschke, B. Gipp, and C. Breitinger. CitePlag: A Citation-based Plagiarism Detection System Prototype. In: Proc. Int. Plagiarism Conf. 2012.

[Mil15] B. R. Miller. Strategies for Parallel Markup. In: Proc. CICM. Cham: Springer International Publishing, 2015, pp. 203–210. DOI: `10.1007/978-3-319-20615-8_13`.

[MM07] R. Miner and R. Munavalli. An Approach to Mathematical Search Through Query Formulation and Data Normalization. In: Proc. MKM. Springer, 2007, pp. 342–355. DOI: `10.1007/978-3-540-73086-6_27`.

[Olv+] *NIST Digital Library of Mathematical Functions.* `http://dlmf.nist.gov/`, Release 1.0.14 of 2017-12-21. F.W.J. Olver, A.B. Olde Daalhuis, D.W. Lozier, B.I. Schneider, R.F. Boisvert, C.W. Clark, B.R. Miller and B.V. Saunders, eds.

[PZ14] N. Pattaniyil and R. Zanibbi. Combining TF-IDF Text Retrieval with an Inverted Index over Symbol Pairs in Math Expressions: The Tangent Math Search Engine. In: Proc. NTCIR-11. National Institute of Informatics, 2014.

[Sch+14] M. Schubotz et al. Evaluation of Similarity-Measure Factors for Formulae Based on the NTCIR-11 Math Task. In: Proc. NTCIR-11. National Institute of Informatics, 2014.

[Sch+16] M. Schubotz et al. Semantification of Identifiers in Mathematics for Better Math Information Retrieval. In: Proc. SIGIR. ACM, 2016, pp. 135–144. DOI: `10.1145/2911451.2911503`.

[Sch17] M. Schubotz. *Augmenting Mathematical Formulae for More Effective Querying & Efficient Presentation.* to appear in Epubli Verlag, Berlin, 2017.

[Shn96] B. Shneiderman. The eyes have it: a task by data type taxonomy for information visualizations. In: Proc. Visual Languages. 1996, pp. 336–343. DOI: `10.1109/VL.1996.545307`.

[SL11] P. Sojka and M. Líška. The Art of Mathematics Retrieval. In: Proc. DocEng. ACM, 2011, pp. 57–60. DOI: `10.1145/2034691.2034703`.

[SW14] M. Schubotz and G. Wicke. Mathoid: Robust, Scalable, Fast and Accessible Math Rendering for Wikipedia. In: Proc. CICM. Springer, 2014, pp. 224–235. DOI: `10.1007/978-3-319-08434-3_17`.

[SY07] M. Shatnawi and A. Youssef. Equivalence detection using parse-tree normalization for math search. In: Proc. ICDIM. IEEE, 2007, pp. 643–648. DOI: `10.1109/ICDIM.2007.4444297`.

[YA09] K. Yokoi and A. Aizawa. An Approach to Similarity Search for Mathematical Expressions using MathML. In: Proc. DML. Brno: Masaryk University Press, 2009, pp. 27–35.

[YS06] A. Youssef and M. Shatnawi. Math search with equivalence detection using parse-tree normalization. In: Proc. CoSIT. 2006.

[ZB12] R. Zanibbi and D. Blostein. Recognition and retrieval of mathematical expressions. In: Proc. IJDAR (2012), pp. 331–357. DOI: `10.1007/s10032-011-0174-4`.

[ZY14] Q. Zhang and A. Youssef. An Approach to Math-Similarity Search. English. In: Proc. CICM. Springer, 2014, pp. 404–418. DOI: `10.1007/978-3-319-08434-3_29`.

```
@inproceedings{SchubotzMHCG17,
author    = {Moritz Schubotz and
  Norman Meuschke and
  Thomas Hepp and
  Howard S. Cohl and
  Bela Gipp},
editor    = {Herman Geuvers and
  Matthew England and
  Osman Hasan and
  Florian Rabe and
  Olaf Teschke},
title     = {{VMEXT:} {A} Visualization Tool for
  Mathematical Expression Trees},
booktitle = {Intelligent Computer Mathematics -
  10th International Conference,
  {CICM} 2017, Edinburgh, UK, July 17-21, 2017, Proceedings},
series    = {Lecture Notes in Computer Science},
volume    = {10383},
pages     = {340--355},
publisher = {Springer},
year      = {2017},
url       = {https://doi.org/10.1007/978-3-319-62075-6_24},
doi       = {10.1007/978-3-319-62075-6_24},
timestamp = {Wed, 28 Jun 2017 13:10:13 +0200},
biburl    =
  {http://dblp.uni-trier.de/rec/bib/conf/mkm/SchubotzMHCG17},
bibsource = {dblp computer science bibliography}
}
```

Listing 1.3. Use the following BibTeX code to cite this article