



*I don't know half of you half as well as I should like, and I like  
less than half of you half as well as you deserve.*

Bilbo Baggins - *The Lord of the Rings*

## CHAPTER 2

# Mathematical Information Retrieval

### Contents

2.1	Background and Overview .....	18
2.2	Mathematical Formats and Their Conversions .....	19
2.2.1	Web Formats .....	20
2.2.1.1	MathML .....	21
2.2.1.2	OpenMath .....	23
2.2.1.3	OMDoc .....	25
2.2.2	Word Processor Formats .....	25
2.2.2.1	LaTeX .....	25
2.2.2.2	Semantic/Content LaTeX .....	28
2.2.2.3	sTeX .....	30
2.2.2.4	Template Editors .....	31
2.2.3	Computable Formats .....	32
2.2.3.1	Computer Algebra Systems .....	32
2.2.3.2	Theorem Prover .....	34
2.2.4	Images and Tree Representations .....	34
2.2.5	Math Embeddings .....	37
2.3	From Presentation to Content Languages .....	38
2.3.1	Background .....	39
2.3.1.1	Related Work .....	42
2.3.2	Benchmarking MathML .....	43
2.3.2.1	Collection .....	43
2.3.2.2	Gold Standard .....	44
2.3.2.3	Evaluation Metrics .....	48
2.3.3	Evaluation of Context-Agnostic Conversion Tools .....	48
2.3.3.1	Tool Selection .....	48
2.3.3.2	Testing framework .....	49
2.3.3.3	Results .....	49
2.3.4	Summary of MathML Converters .....	51
2.4	Mathematical Information Retrieval for LaTeX Translations .....	51

**Supplementary Information** The online version contains supplementary material available at [https://doi.org/10.1007/978-3-658-40473-4\\_2](https://doi.org/10.1007/978-3-658-40473-4_2).

© The Author(s) 2023

A. Greiner-Petter, *Making Presentation Math Computable*,  
[https://doi.org/10.1007/978-3-658-40473-4\\_2](https://doi.org/10.1007/978-3-658-40473-4_2)

Making presentational math computable implies a transformation from one mathematical representation to another. In order to frame this task, we need to introduce presentational and computable formats, and analyze available transformation tools between these formats. There is a large variety of different formats available to encode mathematical expressions, from visual formats, such as  $\LaTeX$  [220] or MathML [60], to semantic enhanced encodings, such as content MathML [270], semantic  $\LaTeX$  [260],  $\text{\texttt{S}\TeX}$  [200], or OpenMath [19], and entire programming languages, such as CAS syntaxes [36, 128, 173, 175, 176, 177, 178, 393], theorem provers [37, 266, 287, 340, 354, 384], or mathematical packages in C++ [168], Python [252] or Java [79]. This chapter introduces what we understand as presentational and computable formats, provides an overview of math formats, and discusses existing transformation tools between these formats.

In particular, Section 2.1 introduces presentational and computable formats. Section 2.2 provides an extensive overview of mathematical formats, their attributes, and conversion approaches between them. Since there are a large variety of conversion tools and approaches available for many different formats [39, 200, 18, 351, 406] a translation from a presentational to a computable format can be achieved in many different ways. In this thesis, we mainly focus on translations from  $\LaTeX$  to CAS syntaxes. The most well-studied translation path from  $\LaTeX$  to CAS syntaxes would use content MathML as an intermediate, semantically enriched format. Hence, Section 2.3 analyzes state-of-the-art  $\LaTeX$  to MathML converters. Section 2.4 underlines the research gap and paves the way for the rest of the thesis by briefly discussing MathIR approaches for conversions from presentational to computable formats. Section 2.3 has been published at the JCDL [18]. The introduction of math embeddings in Section 2.2 was published as a workshop paper at the SIGIR conference [9] and later reused in an extended article for the Scientometrics journal [15].

## 2.1 Background and Overview

Computable encodings are interpretable formal languages in which keywords or sequences of tokens are associated with specific implemented definitions, which allows performing certain mathematical actions on these elements, such as evaluating numeric values or symbolically manipulating the elements. Computable encodings, therefore, must be semantically unambiguous. Otherwise, an interpreter is unable to associate the sequence of tokens with a unique underlying definition. This ambiguity problem is mainly solved by interpreters in two ways: either the system automatically performs disambiguation steps following a decision tree with a fixed set of internal rules, such as  $x^y \cdot z$  in Mathematica, or the system refuses to parse the expression and returns an error, such as for  $x^y \cdot z$  in Maple.



**Computable formats** are formal languages that link key words or phrases with unique implemented definitions. Computable expressions are semantically unambiguous.

Presentational formats, on the other hand, focus on controlling the visualization of mathematical formulae. They generally allow users to change spaces between tokens (e.g.,  $\backslash$ , and  $\backslash;$  in  $\LaTeX$ ), support two-dimensional visualizations (e.g.,  $\int_a^b \frac{dx}{x}$ ), or render entire graphs and images. However, pure presentational formats (in contrast to enhanced semantic encodings) do not specify the meaning of an expression. Consequently, mathematical expressions in presentational for-

mats are generally semantically ambiguous, and it is the author's responsibility to disambiguate the meaning of the expression by providing additional information in the context. Digital presentational formats, such as  $\text{\LaTeX}$ , are also interpretable formal languages<sup>1</sup>. In contrast to computable formats, presentational languages link tokens with specific visualizations rather than executable subroutines. Hence, expressions in these formats must be *unambiguous* too. Otherwise, interpreters are unable to link an expression with a unique visualization (see  $x^y \cdot z$  in  $\text{\LaTeX}$ ). The difference to computable encodings is that expressions in presentational formats must be visually but not semantically unambiguous. For instance,  $\text{\LaTeX}$  refuses to parse  $x^y \cdot z$  because the rendering of  $\{x^y\} \cdot z$  (see  $x^{y \cdot z}$ ) and  $x \cdot \{y \cdot z\}$  (see  $x^{y^z}$ ) is different. In contrast, Maple rejects  $x^y \cdot z$  because there is a mathematical (and in consequence a computational) difference between  $(x^y)^z$  and  $x^{(y^z)}$ .



**Presentational formats** are formal languages with a focus on visualization. Presentational expressions can be semantically but not visually ambiguous.

In this thesis, we focus on  $\text{\LaTeX}$  for the presentational format and CAS syntaxes for computable formats. We choose  $\text{\LaTeX}$  because it is currently the de-facto standard for writing scientific papers in the STEM disciplines [129, 402]. Several other word processors, such as the article's editor in Wikipedia<sup>2</sup> or Microsoft's Word [248], entirely or partially support  $\text{\LaTeX}$  inputs. In addition,  $\text{\LaTeX}$  is the main presentational format that is entered by hand. In contrast, MathML, due to its XML datastructure, is not a user-friendly<sup>3</sup> encoding and mostly automatically generated from other formats [82, 159, 18, 374]. Image formats are the result of pictures, scans, or handwritten inputs, and, therefore, less machine-readable. As a consequence, image formats of mathematical formulae are mainly converted into  $\text{\LaTeX}$  or MathML in a pre-processing step [27, 39, 267, 378, 379, 406, 411]. We choose CAS syntaxes for our target computable format because CAS generally support a large variety of different use cases, from manipulations and visualizations to computations and simulations [81, 413]. Especially general-purpose CAS, such as Maple [36] and Mathematica [393], address a broad range of topics [128, 392]. In contrast, theorem provers, proof assistants, and similar software, as potential other computable formats, solely focus on automated reasoning [147, 266, 354, 384]. Hence, the computation of mathematical formulae plays a less significant role in such software.

## 2.2 Mathematical Formats and Their Conversions

Figure 2.1 provides an overview of different math encodings and existing conversion approaches between them. In addition to the figure, Table 2.1 provides quick access to references for specific translation directions. Figure 2.1 organizes formats by their level of semantics and the level of machine readability. This categorization is not meant to be as accurate as possible nor to be complete. Instead, the figure aims to provide a rough visualization of the most common encodings and their differences. For instance, there is no notable technical difference between

<sup>1</sup>Note that this interpretation of presentational formats does not include images. Since images are less machine-readable formats, they are generally first converted into interpretable formats, such as  $\text{\LaTeX}$ . This conversion process is very challenging on its own [406, 411]. Hence, including images for our task would not provide any benefits but makes it unnecessarily more complicated.

<sup>2</sup>[https://en.wikipedia.org/wiki/Help:Displaying\\_a\\_formula](https://en.wikipedia.org/wiki/Help:Displaying_a_formula) [accessed 2021-10-01]

<sup>3</sup>A little historically described as 'Making humans edit XML is sadistic' from the Django 1.7.11 documentation [118].

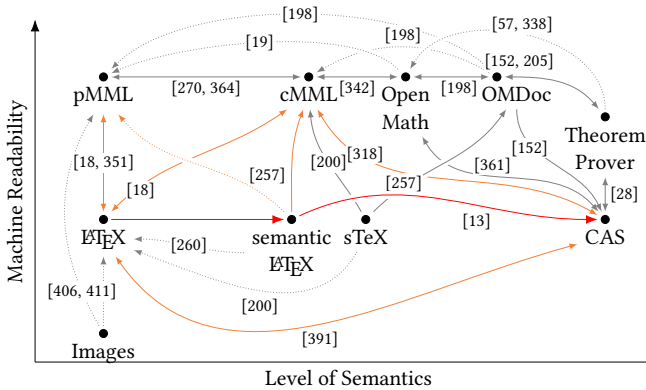


Figure 2.1: Reference map of mathematical formats and translations between them. The red path illustrates the main subject of this thesis. In Section 2.3, we focus specifically on existing translation approaches from LATEX to MathML (orange arrows) to evaluate an alternative to the red translation path.

the levels of semantics in content MathML and OpenMath (see the paragraph about OpenMath in Section 2.2.1). Nonetheless, OpenMath defines the content dictionaries that content MathML uses to semantically annotate symbols beyond school mathematics. Hence, we could argue that content MathML encodes less semantic information without the help of OpenMath and, therefore, should be positioned more to the left. Another disparity can be found in the level of machine readability between CAS syntaxes and theorem provers. Since both formats are programming languages, any CAS or theorem prover expression requires a very specific (often proprietary) parser. Thus, a programming language is arguably never more *machine readable* than any other programming language. Nonetheless, most CAS prefer a more intuitive input format (sometimes even 2D inputs) similar to LATEX over a machine-readable syntax [88, 128, 179] to improve their user experience. Because of these more user-friendly input formats, we positioned CAS syntaxes below theorem prover formats. Note also that math embeddings, i.e., vector representation of math tokens, are not in Figure 2.1 because the level of semantics these vectors capture is still unclear and an open research question (see Section 2.2.5). The red path in Figure 2.1 shows the new translation path that we focus on in this thesis. Dotted arrows represent translation paths that generally do not require context analysis and are, therefore, of less interest for the subject of this thesis. The orange and red arrows (and highlighted cells in Table 2.1) refer to our contributions for this thesis. The red arrows refer the main research contribution explained in the chapters 3 and 4.

### 2.2.1 Web Formats

Web formats are designed to display mathematical formulae and knowledge on the web. Consequently, those formats prioritize machine readability over user experience. Hence, a variety of different translation approaches to, from, or between web formats exists. Since mathematics in the web is generally embedded in HTML code, most web formats use the XML encoding

Table 2.1: Overview table of available mathematical format translations. The highlighted conversion fields refer to contributions made in this thesis. The columns and rows refer to: ‘pMML’ for presentation MathML, ‘cMML’ for content MathML, ‘sem.LaTeX’ for semantic  $\text{\LaTeX}$ , ‘Theo. Prov.’ for theorem prover or proof assistants, ‘Img’ for images, and ‘Speech’ for spoken (audio) mathematical content. The group ‘Comp.’ refers to computable formats. In some cases, no transformation is necessary, e.g., from OMDoc to OpenMath because OMDoc uses OpenMath internally. In this (and similar) cases, we simply refer to the overview publication of the format, here [198] for OMDoc.

From To	pMML	cMML	OpenMath	OMDoc	LaTeX	sem.LaTeX	$\text{\LaTeX}$	Theo. Prov.	CAS	Vector	Img	Speech	
pMML	/	[364]			[61]			[391]	[86]	[358]	[349]		
cMML	[300]	/	[342]	[198]				[391]	[318]	[242]	[257]		Web
OpenMath	[59]	[342]	/	[198]	[61]			[57]	[303]				
OMDoc	[198]	[198]	[198]	/	[198]		[198]	[152]	[152]				
LaTeX	[18]	[159]	[257]	[198]	/	[11]	[195]		[11]	[15]	[358]	[249]	
sem.LaTeX	[257]	[18]	[257]		[257]	/			[13]	[404]	[257]		TeX
$\text{\LaTeX}$	[257]	[257]	[257]	[195]	[198]		/				[257]		
Theo. Prov.			[205]	[205]	[167]			[62]	[338]				
CAS	[391]	[391]			[391]	[13]		[338]	[361]		[391]		Comp.
Vector					[400]					/			
Img	[406]				[406]					[406]	/		
Speech	[386]				[386]			[387]					
			Web			TeX		Comp.					

structure. Thus, web formats are often described as verbose and rarely edited or created by hand. On the other hand, the XML structure simplifies the inter-connectivity between web formats, e.g., via XSL Transformations (XSLT) [362]. There are three main formats used in the web: the current web standard MathML, the pure semantic encoding OpenMath, and the semantic document encoding OMDoc. Note that many websites still use image formats to display math. We will discuss image formats in Section 2.2.4.

### 2.2.1.1 MathML

For the web, the Mathematical Markup Language (MathML) [60] is the current official recommendation from the World Wide Web Consortium (W3C) and even an official standard since 2015 [169] for HTML5. MathML is defined via two different markups: the *presentation*<sup>4</sup> and

<sup>4</sup><https://www.w3.org/TR/MathML3/chapter3.html> [accessed 2021-10-01]

the *content*<sup>5</sup> markup. MathML containing only presentation markup elements is, therefore, also called presentation MathML or, in case it only contains content markup elements, content MathML, respectively. Both markups can be used together side by side for a single expression in so-called parallel markup [202, 259, 270]. If elements in the presentation markup are linked back and forth with elements from the content markup, the encoding is also called cross-referenced MathML.

Content MathML, in contrast to presentation MathML, aims to encode the meaning, i.e., the semantics, of mathematical expressions. Content MathML addresses the issues of ambiguous presentational encodings by providing a standard representation of the content of mathematics. The encoding comes with a large number of predefined functions, e.g., for sin and log, intending to cover most of K-14<sup>6</sup> mathematics. For formulae beyond school mathematics, content MathML use so-called Content Dictionaries (DCs) [204] (see the OpenMath paragraph for more details about CDs). Listing 2.1 shows presentation and content MathML encodings for the Legendre polynomial  $P_n(x)$ . Note that the presentation MathML encoding contains an operator (`<mo>` for *mathematical operator*) between  $P_n$  and  $(x)$  which contains the invisible character *function application* (unicode character U+2061). Nowadays, content MathML is often used in digital libraries to improve the performance of math search engines with accessible semantic information [345, 347, 348, 381].

Since MathML is the web standard, there are numerous tools available that convert other encodings from and to MathML. Most common conversions include translations between presentation and content MathML [139, 270, 364], from [159, 257, 267, 335, 374] and to<sup>7</sup>  $\text{\LaTeX}$ , OpenMath [59, 342, 343], CAS [318], PDF [27, 267], images [406], and audio encodings (mainly in the math to speech research field) [67, 349, 387]. The W3C officially lists 42 converters and other software tools that generate MathML on their wiki<sup>8</sup>. In addition, the official *interoperability report*<sup>9</sup> of MathML provides a comprehensive overview of software that supports MathML and show official statements from implementors. Due to its XML format, most conversion tools use XSLT [362] to transform MathML into either other XML encodings or string representations [59, 61]. This translation approach can be described as rule-based, because in XSLT, we define a set of transformation rules for XML subtrees.

Most of the converters to MathML do not support content MathML. Translations from presentational formats to content MathML face a wide range of ambiguity issues [159, 259, 374]. For example, the `<mo>` element in Listing 2.1 regularly contains the *invisible times* symbol (unicode character U+2062) rather than *function application* because most conversion tools interpret  $P_n$  not as a function. For content MathML, even more disambiguation steps are required to link  $P$  with the Legendre polynomial correctly. For such disambiguation, a combination of semantification and XSLT rules are used to perform translations to content MathML [139, 270, 364]. Nghiem et al. [270] proposes a machine translation approach to generate content MathML from presentation MathML but does not consider textual descriptions from the surrounding context of a formula. Likewise, Toloaca and Kohlhasse [364] uses patterns of notation definitions

<sup>5</sup><https://www.w3.org/TR/MathML3/chapter4.html> [accessed 2021-10-01]

<sup>6</sup>Kindergarten to early college.

<sup>7</sup>Two well-known projects for translations from MathML to  $\text{\LaTeX}$  use XSL transformations: `web-xslt` <https://github.com/davidcarlisle/web-xslt/tree/main/pmml2tex> and `mm12tex` <https://github.com/transpect/mml2tex> [accessed 2021-10-01].

<sup>8</sup>[https://www.w3.org/wiki/Math\\_Tools](https://www.w3.org/wiki/Math_Tools) [accessed 2021-10-01]

<sup>9</sup><https://www.w3.org/Math/iandi/mml3-impl-interop20090520.html> [accessed 2021-10-01]

to find a content MathML expression that matches the presentation MathML parse tree. Grigore et al. [139], on the other hand, generates a local context of five nouns prior to the expression first to conclude symbol declarations from OpenMath CDs. Besides Grigore et al. [139], other existing approaches for translations to content MathML only consider the semantics within the given formula itself or in formulae in the same document [159, 259, 374] but ignore the textual context surrounding a formula. For example, these tools follow the assumption that a  $P$  with subscript followed by an expression in parenthesis should be interpreted as the Legendre polynomial. However, many expressions cannot be disambiguated without considering the textual context, such as the  $\pi(x + y)$  example from the introduction.

Most CAS support MathML either directly or via external software packages [318, 343]. However, to the best of our knowledge, no CAS currently consider the CD in content MathML correctly. Hence, these import and export functions in CAS are generally limited to school mathematics. It should be noted that the CDs are considered by CAS but only in OpenMath, e.g., via the transport protocol *Symbolic Computation Software Composability Protocol* (SCSCP) [361]. Since this protocol was developed to enable inter-CAS communication, we explain this project more in detail in Section 2.2.3.

In summary, a reliable generation of content MathML requires a semantic enhanced source formula, e.g., in CAS syntaxes [318, 343], theorem prover formats [152], or OpenMath [59, 342]. Otherwise, translations tend to generate inaccurate MathML. In Section 2.3, we will examine existing  $\text{\LaTeX}$  to MathML converters more in detail to investigate the practicality of using MathML as an intermediate format for translations from  $\text{\LaTeX}$  to CAS encodings.




### 2.2.1.2 OpenMath

The OpenMath Society (originally OpenMath Consortium [19]) defines another standard encoding called OpenMath [53]. The OpenMath standard aims to focus exclusively on the semantics of mathematics and, therefore, going a step further compared to MathML [204], which aims to cover both the presentation and the content information in a single format. Originally, OpenMath was invented during a series of workshops starting in 1993, mainly from researchers in the computer algebra community, to easily exchange mathematical expressions between CAS and other systems [19, 89]. MathML, originally developed with the same goal, was first released in 1998<sup>10</sup>. Both formats are very similar to each other [204] and one may ask for the purpose of two different formats for more or less the same tasks [82, 114]. Discussions about the necessity of both formats raise from time to time even decades later [25, 204]. However, OpenMath and MathML have been and are still developed alongside each other rather than competing with one another due to a large overlap of people working on both formats [204]. To summarize the coexistence today: MathML provides rendered visualizations for OpenMath, while the Content Dictionaries (CDs) from OpenMath add semantics to MathML<sup>11</sup>.

The OpenMath Society maintains a set of standard CDs. A CD is a set of declarations (i.e., definitions, notations, constraints, etc.) for mathematical symbols, functions, operators, and other mathematical concepts. The idea behind the publicly maintained CDs by the OpenMath

<sup>10</sup><https://www.w3.org/TR/1998/REC-xml-19980210> [accessed 2021-10-01]

<sup>11</sup>A more detailed discussion about the history of both formats can be found at <https://openmath.org/projects/esprit/final/node6.htm>, <https://openmath.org/om-mml/> [both accessed 2021-10-01], and [198, pp. 5].

 <b>Presentational MathML</b>	 <b>Content MathML</b>
<pre> 1 &lt;mrow&gt; 2   &lt;msub&gt; 3     &lt;mi&gt;P&lt;/mi&gt; 4     &lt;mi&gt;n&lt;/mi&gt; 5   &lt;/msub&gt; 6 &lt;/mrow&gt; 7 &lt;!-- Invisible 8   Funct. Appl. 9   Unicode U+2061 --&gt; 10 &lt;/mo&gt; 11 &lt;mrow&gt; 12   &lt;mo&gt;(&lt;/mo&gt; 13   &lt;mi&gt;x&lt;/mi&gt; 14   &lt;mo&gt;)&lt;/mo&gt; 15 &lt;/mrow&gt; 16 &lt;/mrow&gt; </pre>	<pre> 1 &lt;apply&gt; 2   &lt;csymbol definitionURL="http://www. 3     openmath.org/cd/orthpoly1.ocd" 4     encoding="OpenMath"&gt;legendreP 5 &lt;/csymbol&gt; 6 &lt;ci&gt;n&lt;/ci&gt;&lt;ci&gt;x&lt;/ci&gt; 7 &lt;/apply&gt; </pre> <div>  <b>OpenMath</b> </div> <pre> 1 &lt;OMOBJ&gt;&lt;OMA&gt; 2   &lt;OMS name="legendreP" cd="orthpoly1"/&gt; 3   &lt;OMV name="n"/&gt; 4   &lt;OMV name="x"/&gt; 5 &lt;/OMA&gt;&lt;/OMOBJ&gt; </pre>

Listing 2.1: The Legendre polynomial in two MathML encodings and in OpenMath.

Society is to provide a ground truth for math declarations so that the used symbols become interchangeable among different parties. However, everybody can create new custom CDs which might be integrated into the existing standard set maintained by the OpenMath Society [90]. M. Schubotz [327], for example, proposed a concept for a CD that uses on the knowledge base Wikidata. More recently, B. Miller [258] created a content dictionary specifically for the functions in the DLMF.

Listing 2.1 compares both MathML markups with OpenMath. While the tree structures of content MathML and OpenMath cannot directly be compared with mathematical expression trees [331] (see also Section 2.2.4), the XML tree structure of both formats is unique. Both formats rely on the CD entry of the Legendre polynomial in `orthpoly1`<sup>12</sup>. Since the CD is from OpenMath, the OpenMath encoding does not require the entire url. The CD entry further specifies that the Legendre polynomial has two arguments. Hence, the following two siblings in the tree structure are considered to be the arguments. OpenMath specifically annotate them as OMV (for variable objects). Alternatively to the `orthpoly1` CD by OpenMath, one can also use Schubotz's [327] Wikidata CD to annotate  $P$  with the Wikidata item [Q215405](https://www.wikidata.org/wiki/Q215405) or Miller's [258] DLMF CD to link  $P$  to §18.3 of the DLMF [98, (18.3)].

As previously mentioned, both formats (content MathML and OpenMath) are rather similar to each other [56, 343]. Hence, there are several ways to transform mathematical expressions between both formats [343], e.g., via XSLT [59, 342]. This transformation is possible without information retrieval techniques since both formats encode the same level of semantic information via CDs. Even though the primary goal for OpenMath was to provide a format that allows communication between mathematical software [19], most CAS do not support OpenMath directly. Instead, an independent project of research institutions funded by the European Union was launched to improve the *symbolic computation infrastructure in Europe*. The main

<sup>12</sup><https://openmath.org/cd/orthpoly1.html#legendreP> [accessed 2021-10-01]



result of this project was the SCSCP protocol for inter-CAS communication via OpenMath. We will discuss the SCSCP protocol and the project more in detail in Section 2.2.3. Several CAS, including Maple [243] and Mathematica [44], implemented endpoints for the SCSCP protocol. Hence, via this new protocol, CAS support OpenMath to some degree. Apart from the protocol solution, there are some research projects available that use OpenMath as an interface to and between CAS and theorem prover formats [57, 152, 303, 338, 343].

### 2.2.1.3 OMDoc

Sometimes, it might be worthwhile to annotate the context of mathematical expressions with additional information explicitly. For example, an equation might be part of a theorem that has not been proven yet. Hence, that particular equation and its context should not be confused with a definition. Since this meta-information about mathematical expressions is organized on a document level, Kohlhasse [198, 199] introduced another format, the Open Mathematical Document (OMDoc), to semantically describe entire mathematical documents. While formats like OpenMath or MathML encode the semantics of single expressions, which Kohlhasse describes as the *microscopic* level, OMDoc aims for the *macroscopic*, i.e., the document level. This format can be especially useful for interactive documents [80, 85, 131, 150, 162, 201] and theorem prover [38, 146, 163, 340] which generally rely more on the meta information from a document level. Single math expressions in OMDoc are still encoded as OpenMath for the semantics and MathML for the visualization. In turn, this thesis focuses more on the formats that directly encode mathematical expressions rather than a *macroscopic* level encoding. Nonetheless, it should be noticed that a translation to a CAS might be different depending on the scope of an equation, e.g., an equation symbol in a definition differs from an equation symbol in an example. Heras et al. [152], for example, used OMDoc to interface CAS and theorem prover. Hence, the OMDoc format might be worth supporting once the translation reaches a level of reliability and comprehensiveness that the semantics on the document level matter (see the future work section 6.3).

## 2.2.2 Word Processor Formats

The previously explained formats of mathematics are beneficial for web applications and exchanging mathematical knowledge between systems. However, the underlying verbose XML data structure makes manual maintenance of these formats too cumbersome. In turn, MathML and OpenMath, considering a specific size, are almost always computer-generated. The actual source of the data, something a human manually typed, uses a different format, such as  $\LaTeX$ , visual template editors, or image formats. In the following, we introduce formats and methods used to type mathematics in word processors manually.

### 2.2.2.1 $\LaTeX$

$\LaTeX$  is currently the de-facto standard for writing scientific papers in the STEM disciplines [129, 220, 402] and has even been described as *the lingua franca of the scientific world* [220]. Numerous other word processors entirely or partially support  $\LaTeX$  inputs.  $\LaTeX$  was developed by Leslie Lamport and extended the  $\TeX$  system with some valuable macros that make working with  $\TeX$  easier [220].  $\TeX$  was developed by Donald E. Knuth [189, p.559] in 1977. Knuth was dissatisfied with the typography of his book, *The Art of Computer Programming* [189, pp. 5, 6, and 24] and created  $\TeX$  to overcome the hurdles of consistently and reliably typesetting mathematical

formulae for printing. Today, there is no significant difference between  $\LaTeX$  and  $\TeX$  in terms of mathematical expressions. Hence, we continue using  $\LaTeX$  as the modern successor and refer to  $\TeX$  only to underline technical differences or to describe the underlying base for other  $\TeX$ -like encodings.  $\LaTeX$  provides an intuitive syntax for mathematics that is similar to the way a person would write the math by hand, e.g., by using the underscore to set a sequence of tokens in subscript.

$\LaTeX$  is an interpretable language that requires a parser. Theoretically, the flexibility of  $\LaTeX$  (and especially the underlying  $\TeX$  implementation) makes parsing  $\LaTeX$  really challenging [187]. For example,  $\TeX$  allows to redefine every literal at runtime, making  $\TeX$  (and therefore  $\LaTeX$  too) to a context-sensitive formal language. However, in practice, most  $\LaTeX$  literals are generally not redefined. Instead, it is common to extend  $\LaTeX$  with additional commands rather than redefining existing logic. Especially in mathematical expressions, several projects simply presume that  $\LaTeX$  is parsable with a context-free grammar, which makes parsing mathematical expressions in  $\LaTeX$  a lot simpler [71, 402].

Since  $\LaTeX$  is the standard to typeset mathematics, there are numerous of translation tools to the webstandard MathML available [133, 135, 159, 257, 267, 335, 374] (see also MathML explanation in Section 2.2.1). In the next Section 2.3, we will focus more closely on translations between  $\LaTeX$  and MathML.  $\LaTeX$  is also a standard target encoding for Optical Character Recognition (OCR) techniques [406, 411], which retrieve mathematical expressions from images or PDF files (see Section 2.2.4).  $\LaTeX$  focus solely on the representation of math (similar to presentation MathML). Additionally, recent studies try to explore the capabilities of trained vector representations of  $\LaTeX$  expressions [121, 15, 215, 360, 400, 404] to explore new similarity measure and search engines [404], classification approaches [404], and even automatically generating new  $\LaTeX$  expressions [400]. Nonetheless, the effectiveness of capturing the semantic information with these methods is controversial [9].

**$\LaTeX$  to CAS converters** Most relevant for our task are existing translation approaches directly from  $\LaTeX$  to CAS sytanxes. These translators can be categorized in two groups: (1) CAS internal import functions and (2) external programs for specific or multiple CAS. Mathematica [391] and SymPy [357] are two CAS with the ability to import  $\LaTeX$  expressions directly. SymPy's import function was ported from the external `latex2sympy`<sup>13</sup> project. Examples of external tools are SnuggleTeX [251] and our in-house translator  $\mathcal{L}\mathcal{C}\mathcal{A}\mathcal{T}$  [3, 13]. SnuggleTeX is a  $\LaTeX$  to MathML converter with the experimental feature to perform translations to the CAS Maxima [324].  $\mathcal{L}\mathcal{C}\mathcal{A}\mathcal{T}$  is the predecessor project of this thesis and focused on translating semantic  $\LaTeX$  from the DLMF to the CAS Maple.

All of these converters are rule-based translators, i.e., they perform translations on hard-coded pre-defined conversion rules. SnuggleTeX support translations to Maxima since version 1.1.0 [251]. The tool allows users to manually predefine translation rules, such as interpreting  $e$  as the mathematical constant,  $\Gamma$  as the Gamma function, or  $f$  as a general function. SnuggleTeX is no longer actively maintained and mostly fail to translate general expressions. The developers themselves declare the translation to Maxima as experimental and limited<sup>14</sup>. SymPy, in contrast,

<sup>13</sup>The project is therefore no longer actively developed but still available on GitHub: <https://github.com/augustt198/latex2sympy> [accessed 2021-10-01]

<sup>14</sup><https://www2.ph.ed.ac.uk/snuggletex/documentation/semantic-enrichment.html> [accessed 2021-10-01]

is actively maintained and provide a more sophisticated import function for  $\text{\LaTeX}$  expressions. SymPy's import function parses a given  $\text{\LaTeX}$  expression via ANTLR<sup>15</sup> and traverses through the parse tree to convert each token (and subtree) into the SymPy syntax. SymPy uses a set of heuristics that mostly cover standard notations, including  $\backslash \sin$ . Additionally, it uses pattern matching approaches to identify typical mathematical concepts, such as the derivative notation in  $\frac{d}{dx} \sin(x)$ . Similarly,  $\text{\LaTeX}$  first parses the input expression with the Part-of-Math (POM) tagger [402] and performs translations by traversing through the parse tree. The POM tagger tags tokens with additional information from external lexicon files.  $\text{\LaTeX}$  manipulates these lexicon files to tag tokens with their appropriate translation patterns.  $\text{\LaTeX}$  takes the translation patterns attached to a single token and fills them with the following and preceding nodes in the parse tree to perform a translation. Within this thesis, we will extend  $\text{\LaTeX}$  further with pattern matching techniques and human-inspired heuristics to perform more general formulae, including the derivative notation example, sums, products, and other operators. A more detailed discussion about the first version of  $\text{\LaTeX}$  is available in [13].

While SymPy and SnuggleTeX are open source and allows interested readers to analyze the internal implementation details, we can only speculate about the solutions in proprietary software, such as Mathematica. As we saw in Table 1.2 (and later in Chapter 4), Mathematica seems to follow a pattern recognition approach to link known notations, such as  $P_n^{(\alpha, \beta)}(x)$ , to their internal counterparts, such as `JacobiP[n, \[Alpha], \[Beta], x]`. Since Mathematica (nor does any other CAS or mentioned converter) analyze the textual context of a formula, importing ambiguous notations generally fail. Since the internal logic (and therefore the underlying patterns) is hidden, it is difficult to estimate the accuracy and power of Mathematica's  $\text{\LaTeX}$  import function. As an alternative to Mathematica itself, one can use WolframAlpha<sup>16</sup> [309]. WolframAlpha is described as a knowledge or answer engine. Technically, WolframAlpha is a web interface which uses Mathematica as backbone for computations. WolframAlpha performs numerous of pre-processing and interpretation steps to allow users to generate scientific information without inputting specific Mathematica syntax [64, 383].

Table 2.2 compares the converters on our introduction examples (see Table 1.2). The table contains also  $\text{\LaTeX}$  first version (published in 2017 [3]) for comparison. We observe that WolframAlpha clearly performs best on this simple general inputs. The reason is that WolframAlpha focus on a broad, less scientific audience which allows the system to make several assumptions. On more topic specific inputs, such as  $P_n^{(\alpha, \beta)}(\cos(a\theta))$ , it fails. This is further underlined by the fact that Mathematica itself has no trouble interpreting  $P_n^{(\alpha, \beta)}(\cos(a\theta))$ . This indicates that both systems are optimized for their expected user groups. On these simple cases, SymPy also performs better compared to Mathematica. However, SymPy's size and support of special functions is not comparable with Mathematica and therefore falls behind Mathematica on a more scientific dataset, such as the DLMF.

A more sophisticated evaluation on 100 randomly selected DLMF formulae revealed that Mathematica can be considered the current state-of-the-art for translating  $\text{\LaTeX}$  to CAS. Nonetheless, it only translated 11 cases correctly compared to 7 successful translations by SymPy and 22 by  $\text{\LaTeX}$ . The full benchmark is available in Table E.1 in Appendix E.1 available in the electronic supplementary material.

<sup>15</sup>ANother Tool for Language Recognition (ANTLR): <https://www.antlr.org/index.html> [accessed 2021-10-01]

<sup>16</sup>Often stylized with Wolfram|Alpha

Table 2.2:  $\LaTeX$  to CAS translation comparison between Mathematica’s (MM) and SymPy’s (SP) import functions, SnuggleTeX (ST) translation to Maxima, WolframAlpha (WA) interpretation of  $\LaTeX$  inputs, and the first version of  $\LaTeX$  (LCT<sub>1</sub>)

$\LaTeX$	Rendering	MM	SP	ST	WA	LCT <sub>1</sub>
$\int_a^b x \, dx$	$\int_a^b x dx$	✗	✓	✗	✓	✗
$\int_a^b x \, \mathrm{d}x$	$\int_a^b x dx$	✗	✗	✗	✓	✗
$\int_a^b x \, , \, dx$	$\int_a^b x dx$	✓	✓	✗	✓	✗
$\int_a^b x \, ; \, dx$	$\int_a^b x dx$	✗	✓	✗	✓	✗
$\int_a^b x \, , \, \mathrm{d}x$	$\int_a^b x dx$	✗	✗	✗	✓	✗
$\int_a^b \frac{dx}{x}$	$\int_a^b \frac{dx}{x}$	✗	✓	✗	✓	✗
$\sum_{n=0}^N n^2$	$\sum_{n=0}^N n^2$	✓	✓	✓	✓	✓
$\sum_{n=0}^N n^2 + n$	$\sum_{n=0}^N n^2 + n$	?	?	✗	?	?
$\{n \text{ choose } m\}$	$\binom{n}{m}$	✗	✗	✗	✓	✗
$\binom{n}{m}$	$\binom{n}{m}$	✓	✓	✓	✓	✓
$P_n^{(\alpha, \beta)}(\cos(a\Theta))$	$P_n^{(\alpha, \beta)}(\cos(a\Theta))$	✓	✗	✗	✗	✓
$\cos(a\Theta)$	$\cos(a\Theta)$	✓	✓	✓	✓	✓
$\frac{d}{dx} \sin(x)$	$\frac{d}{dx} \sin(x)$	✗	✓	✗	✓	✗

Since  $\LaTeX$  can be easily extended with new content via macros, some projects try to semantically enhance  $\LaTeX$  with unambiguous commands. The two most comprehensive projects are semantic  $\LaTeX$  and  $\S\LaTeX$ .

### 2.2.2.2 Semantic/Content $\LaTeX$



#### The Jacobi polynomial in $\LaTeX$ and semantic $\LaTeX$

```

1 P_n^{(\alpha, \beta)}(x) % Generic LaTeX
2 \JacobipolyP{n}{\alpha}{\beta}@{x} % Semantic LaTeX

```

Listing 2.2: The Jacobi polynomial in  $\LaTeX$  (line 1) and semantic  $\LaTeX$  (line 2).

Semantic  $\LaTeX$  (also known as content  $\LaTeX$ ) was developed by Bruce Miller [260] at the National Institute of Standards and Technology (NIST) to semantically enhance the equations in the DLMF [403]. Essentially, semantic  $\LaTeX$  is a set of custom  $\LaTeX$  macros which are linked to unique definitions in the DLMF. Consider for example the Jacobi polynomial in Listing 2.2. The general  $\LaTeX$  expression does not contain any information linked to the Jacobi polynomial. However, semantic  $\LaTeX$  replaces the general expression with a new macro `\JacobipolyP` which is linked to the DLMF [98, (18.3#T1.1.r2)]<sup>17</sup>. In addition, all variable arguments (param-

<sup>17</sup>Hereafter, we refer to specific equations in the DLMF by their labels. The label can be added to the base URL of the DLMF. For example, the sine function is defined at 4.14.E1, which can be reached via <https://dlmf.nist.gov/4.14.E1> [accessed 2021-10-01].

ters and variables) are separated and ordered following the function command. This separation is essential to disambiguate notations. For example, the sine function is sometimes written without parenthesis, such as  $\sin x$ , resulting in ambiguous semantic notations, such as in  $\sin x + y$ . The semantic  $\text{\LaTeX}$  macros allow to visualize this expression but encode it unambiguously via `\sin@@{x+y}` (which is rendered as  $\sin x + y$ ). Originally, the semantic  $\text{\LaTeX}$  helped to develop a reliable search engine for the DLMF [260]. Nowadays, the macros are also in use in other projects and have been even extended for the Digital Repository of Mathematical Formulae (DRMF) [77, 78], an outgrowth of the DLMF.

Semantic  $\text{\LaTeX}$  will play a crucial role in the rest of this thesis because it allows us to stick with the easily maintainable syntax of  $\text{\LaTeX}$  but semantically elevates the information of math expressions to a level that can be exploited for translations towards CAS [3, 8, 13]. The main reason is that the semantic  $\text{\LaTeX}$  macros mostly cover OPSF from the DLMF. OPSF are a set of functions and polynomials which are generally considered as important, such as the trigonometric functions (also categorized as elementary functions), the Beta function, or orthogonal polynomials. Most OPSF have more or less well-established names and standard notations. The DLMF (i.e., especially the original book [276]) is considered a standard reference for OPSF [381]. General-purpose CAS, such as Mathematica and Maple, focus also on the comprehensive support of OPSF [381]. Hence, semantic  $\text{\LaTeX}$  macros play a crucial role for translations from  $\text{\LaTeX}$  to CAS syntaxes. Since CAS syntaxes are programming languages, CAS can be extended with new code. However, translating new math formulae to CAS can become arbitrarily complex. Consider the prime counting function would be not supported by Mathematica. In this case,  $\pi(x + y)$  cannot be translated to a simple mathematical formula in the syntax of Mathematica but would require entire new subroutines. Therefore, a comprehensive, viable, and reliable translator from  $\text{\LaTeX}$  to the syntax of CAS should maximize its support for OPSF in order to be useful.

Definition 2.1 provides a brief definition for the elements of a semantic macro. While the semantic source of the DLMF is publicly available [403], the actual definitions, i.e., the  $\text{\LaTeX}$  style files, of the macros, are still private<sup>18</sup>. B. Miller provided access to the definitions of the macros for this thesis. Later in this thesis, we will rely on additional meta-information given for each semantic macro. This includes default parameters and variables, a short textual description, and links to the DLMF CD [258]. Further information is not explicitly given in the macro definition files. For example, function constraints, domains, branch cut positions, singularities, and other properties are only given in the DLMF.

As previously mentioned, we<sup>19</sup> developed  $\text{\LaTeX}$  to translating semantic  $\text{\LaTeX}$  DLMF formulae to CAS [3, 13]. The first version did not contain any disambiguation steps or pattern matching approaches to deduce the intended meaning of an expression. Instead, it fully relied on the semantic  $\text{\LaTeX}$  macros to perform translations to Maple. For example, sums or products were not supported directly but required the semantically enhanced macros from the DRMF [77, 78]. The source of  $\text{\LaTeX}$  is not yet publicly available<sup>20</sup> due to the dependency to the POM tagger [402] and the semantic  $\text{\LaTeX}$  macros [260, 403] but accessible via open API endpoints<sup>21</sup>.

<sup>18</sup>As of 2021-10-1.

<sup>19</sup>The first version of  $\text{\LaTeX}$  was the subject of my Master's thesis and laid the foundation for a reliable translation from semantic  $\text{\LaTeX}$  to multiple CAS.

<sup>20</sup>As of 2021-10-01.

<sup>21</sup>The API contains a Swagger UI and is reachable at <https://vmext-demo.formulasearchengine.com> [accessed 2021-10-01].  $\text{\LaTeX}$  is available under `math/translation` path (in the math controller). The experimental

**Definition 2.1: The elements of a semantic macro**

A semantic  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  macro is a  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  macro with a unique name followed by a number of arguments. Certain elements of the following arguments are optional but the order remains the same. While a caret and primes are interchangeable, each order would have a different meaning, as it can be seen in the example below.

**A semantic macro and its arguments:**

<code>\macro</code>	The unique semantic macro name with a backslash
<code>[optPar]</code>	An optional parameter in square brackets
<code>{par}</code>	Parameters in curly brackets
<code>'</code> or <code>^</code>	Optional prime symbols or a caret for power notations
<code>@</code>	A number of <code>@</code> symbols to control the visualization of the macro
<code>{var}</code>	Variables in curly brackets

**Examples:**

```

\sin@{x} → sin(x)
\sin@@{x} → sin x
\BesselJ{\nu}''^2@{z} → J_n''^2(z)
\BesselJ{\nu}^2''@{z} → (J_n^2)''(z)
\genhyperF{2}{1}@{a,b}{c}{z} → {}_2F_1(a, b; c; z)
\genhyperF{2}{1}@@{a,b}{c}{z} → {}_2F_1(a, b; c; z)
\genhyperF{2}{1}@@@{a,b}{c}{z} → {}_2F_1(a, b; c; z)

```

Apart from  $\mathbb{E}\mathbb{C}\mathbb{A}\mathbb{S}\mathbb{T}$ ,  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}\mathbb{M}\mathbb{L}$  [257] is another tool that supports semantic  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  and provides conversions to  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$ , MathML, and a variety of image formats.  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}\mathbb{M}\mathbb{L}$  was also developed by B. Miller with the original goal to support the development of DLMF [133].  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}\mathbb{M}\mathbb{L}$  is a general  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  to XML converter. However, in order to support the development of the DLMF,  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}\mathbb{M}\mathbb{L}$  is able to fully load semantic  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  definition files to convert semantic  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  into semantically appropriate content MathML. With this ability,  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}\mathbb{M}\mathbb{L}$  is generally capable of converting other  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  encodings too, such as the following  $\mathbb{S}\mathbb{T}\mathbb{E}\mathbb{X}$ .

**2.2.2.3 sTeX**

$\mathbb{S}\mathbb{T}\mathbb{E}\mathbb{X}$  refers to semantic  $\mathbb{T}\mathbb{E}\mathbb{X}$  and should not be confused with B. Miller's semantic  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$ .  $\mathbb{S}\mathbb{T}\mathbb{E}\mathbb{X}$  was developed around 2008 [194, 195, 200] with the goal to semantically annotate  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  documents with semantic macros. Specifically,  $\mathbb{S}\mathbb{T}\mathbb{E}\mathbb{X}$  should serve as a source format to generate the semantic document format OMDoc. While the underlying motivation and technical solution of  $\mathbb{S}\mathbb{T}\mathbb{E}\mathbb{X}$  and semantic  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  are very similar, there are some core differences between both formats. Semantic  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  was developed specifically for the DLMF and, therefore, provide semantic macros for OPSF. In particular, a semantic macro in the DLMF represents a specific unique function. In turn,  $\mathbb{S}\mathbb{T}\mathbb{E}\mathbb{X}$  aim to cover general mathematical notations and provide a logic to semantically annotate general functions and symbols. Consider the aforementioned example  $\pi(x + y)$ . If  $\pi$  is referring to the prime counting function, we can resolve the ambiguity with semantic  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  via `\nprimes@{x+y}` since the semantic macro `\nprimes` is referring to that function.

flag performs pattern matching approaches described later in this thesis. The label allows to specify a DLMF equation label to perform specific assumptions (e.g., that  $i$  is an index and not the imaginary unit).

In  $\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$ , an author can use modules and IDs to define the function and set the notation via `\symdef{\pi}[1]{\prefix{\pi}{#1}}`. While this makes the interpretation of  $\pi(x + y)$  unambiguous, an underlying definition is still missing. Hence,  $\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$  provides the option to link symbols with their definitions in the document. This definition linking underlines the original motivation and connection to the semantic document format OMDoc.

Since  $\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$  is not limited to specific domains, we could define any notation we want in our semantic document. On the other hand, this generalizability of  $\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$  makes the format more verbose and somehow similar to a programming language. In  $\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$ , we need to define and declare symbols explicitly. In addition, a defined new symbol still needs to be manually linked to an underlying definition. In semantic  $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$ , the macro itself is linked to the appropriate definition in the DLMF.  $\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$  provide access to predefined sets of macros that aim to cover K-14 mathematics [195].

In conclusion,  $\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$  is flexible but verbose. The format is useful when it comes to annotating a general mathematical document semantically. However, the strength of  $\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$ , for example, the ability to define any symbol with specific semantics, is generally not very important for translations to CAS. CAS have a fixed set of supported functions and often try to mimic common notation styles, e.g., one does not need to define — as a unary postfix operator in  $-2$ . In turn, a translation from  $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$  to CAS faces the issue of identifying the name of the functions involved, its arguments, and the appropriate mappings to counterparts in CAS syntax. Semantic  $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$ , on the other hand, provides a syntax that makes it easy to solve these issues. The name of the function is directly encoded in the name of the macro, the arguments are explicitly declared and distinguishable (by curly brackets), and a mapping to an appropriate counterpart in the CAS can be more easily found due to the large overlap of functions in the DLMF and supported functions in CAS.

As previously mentioned,  $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}\mathcal{M}\mathcal{L}$  [257] is able to load  $\mathcal{T}\mathcal{E}\mathcal{X}$  definition files and support conversion to XML encodings. Hence,  $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}\mathcal{M}\mathcal{L}$  can transform  $\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$  expressions to content MathML[200]. The ability to link  $\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$  symbols with their definitions in a document or external source further makes it to a source for generating entire semantic enhanced OMDoc documents [195].  $\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$  could be also used as an alternative to semantic  $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$  for translations to CAS. However, due to the natural overlap of functions in the DLMF and CAS, at some point in the development of a translation process on  $\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$ , we would create semantic enhanced macros for OPSF similar to the existing semantic  $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$  macros. Hence, using  $\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$  in comparison to semantic  $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$  has no direct advantages to perform translations towards CAS. The higher flexibility of  $\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$  makes it a good candidate for translations beyond OPSF.

#### 2.2.2.4 Template Editors

Since  $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$  is an interpretable language with over ten thousand mathematical symbols alone [280], learning  $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$  syntax is often simply too time-consuming and complex for many users. To provide an easier access to rendered mathematics, especially in so-called *what you see is what you get* (WYSIWYG) editors, such as Microsoft's Office programs<sup>22</sup> or Wikipedia's visual article editor<sup>23</sup>, template editors become the norm. Template editors provide visual templates

<sup>22</sup><https://support.microsoft.com/en-us/office/equation-editor-6eac7d71-3c74-437b-80d3-c7dea24fd3f3> [accessed 2021-10-01]

<sup>23</sup>The wikipedia's article about formula editors ([https://en.wikipedia.org/wiki/Formula\\_editor](https://en.wikipedia.org/wiki/Formula_editor)) [accessed 2021-10-01]

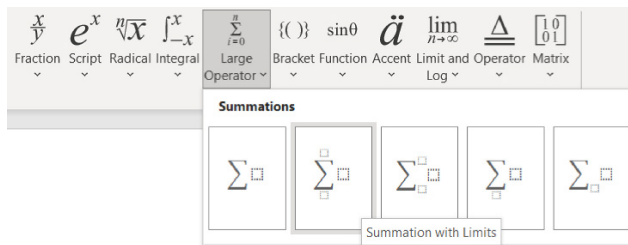


Figure 2.2: The math template editor of Microsoft's Word [395].

of standard mathematical notations so that the user only needs to fill in the remaining spaces. Figure 2.2 shows the template editor of Microsoft's Word [395] for a snippet of the templates for sums. Modern graphic interfaces of CAS also often contain such template editors to improve the user experience further. In comparison to  $\text{\LaTeX}$ , template editors are generally easier to use but limited to the offered templates. Hence, for more complex expressions, template editors are often described as confining [273]. Template editors do not introduce a new math format. The editors only provide a different input method but encode the mathematical formulae in system-specific formats, such as MathML in Microsoft's Word or Maple syntax in Maple.

### 2.2.3 Computable Formats

So far, we have covered the major formats that focus on the presentation of mathematical expressions and on formats that capture the semantics. Even though formats like content MathML, OpenMath, and the semantic  $\text{\LaTeX}$  extensions can resolve the ambiguity of math formulae, they are not computable formats, i.e., we cannot perform actual calculations and computations on them. The syntax of a computable format is a formal language in which every word is linked to specific subroutines. Much like programming languages, computable formats are semantically unambiguous and interpretable. In turn, computable formats are generally part of a larger software package that ships an interpreter to parse inputs and an engine that performs the computations. In the following, we briefly discuss CAS and theorem prover formats as examples of computable formats. We will not specifically focus on math packages for specific programming languages, such as C++ [168], Python [252] or Java [79]. Most CAS and theorem provers, however, internally rely on those *lower-level* packages to some degree.

#### 2.2.3.1 Computer Algebra Systems

A CAS is a mathematical software that can perform a variety of mathematical operations on math inputs, such as symbolic manipulations, numeric calculations, plotting and visualization, simplification, and many more [76, 81, 128, 413]. With the increasing power of computers, CAS became a crucial part of the modern scientific world [32, 262, 352, 356] and are widely used for mathematical problem solving [49, 51, 127, 216, 414], simulations [46, 142, 166, 265, 294], symbolic manipulations [115, 325], and even for teaching students from schools to universities [158, 237, 244, 350, 363, 365, 389, 390]. Due to their complexity, CAS are often large and expensive proprietary software packages [36, 164, 393]. However, there are several well-known open



source options available [42], such as SymPy [252], Axiom<sup>24</sup> [176], and Reduce<sup>25</sup> [151]. Many CAS focus on specific domains or mathematical tasks, such as Cadabra [289, 290, 291] (tensor field theory), FORM [372] (particle physics), GAP [177] (group theory and combinatorics), PARI/GP [283] (number theory), or MATLAB [164] (primarily for numeric computation). In contrast, general-purpose CAS, including Mathematica [393], Maple [36], Axiom [176], SymPy [178, 252], Maxima [264, 324], or Reduce [151], aim to provide a large set of tools and algorithms that are beneficial for many mathematical applications. Therefore, general-purpose CAS support a large number of OPSF, since these functions and polynomials are used in a large variety of different scientific fields, from pure and applied mathematics to physics and engineering. Therefore, we primarily focus on translations to general-purpose CAS in this thesis rather than to domain-specific CAS.

The input formats of general-purpose CAS are often multi-paradigm programming languages [88], i.e., they combine multiple standard programming features, such as functional, mathematical, and procedural approaches. Major CAS generally use their own input language, such as the *Wolfram Language* in Mathematica [392]. Like any programming language, the input format must be unambiguous to the underlying parser of the CAS so that every keyword is uniquely linked to subroutines in the CAS engine. This link to a subroutine makes the expression computable. In contrast, the semantic  $\text{\LaTeX}$  macros are linked to theoretical mathematical concepts defined in the DLMF but not with specific implementations. Hence, a translation to a CAS syntax requires to link mathematical notations, e.g.,  $\Gamma(z)$ , that refer to specific mathematical concepts, e.g., the Gamma function, to the correct sequence of keywords in the CAS, e.g., `GAMMA(z)` in Maple.

Since computable languages naturally encode the highest level of semantic information in their expressions, a translation towards other systems that encode less semantic information is possible with a comprehensive list of simple mapping rules. Many CAS therefore provide a variety of different output formats, from  $\text{\LaTeX}$  to MathML (including content MathML) and images. Translations between CAS or other mathematical software, such as theorem prover, require more sophisticated mappings due to system-specific implementations [110]. From 2006 to 2011, a joint research project funded by the European Union with over 3 Million Euro launched intending to improve the symbolic computation infrastructure for Europe<sup>26</sup>. The result of the *SCIENCE project* was the *Symbolic Computation Software Composability Protocol* (SCSCP) [119, 361], which uses the OpenMath encoding to transfer mathematical expressions. Using the SCSCP, interfaces for GAP [206], KANT [120], Maple [243], MuPAD [155], Mathematica [44], and Macaulay2 [311] were implemented.

Note that there are solutions available that do not require any translation between  $\text{\LaTeX}$  and CAS. For example, the CAS syntax of Cadabra [291] is a subset of  $\text{\TeX}$  itself. Similarly, SageTeX<sup>27</sup> is a  $\text{\LaTeX}$  package that allows authors to enter SageMath [317] expressions into  $\text{\LaTeX}$  documents, turning the document into an interactive document [201] to some degree. SageMath is a general-purpose CAS that relies on existing solutions for domain-specific tasks, such as GAP [177] for group theory or PARI/GP [283] for number theory problems. These solutions do not require

<sup>24</sup>Open source since 2001 (first released in 1965).

<sup>25</sup>Open source since 2008 (first released in 1963).

<sup>26</sup>EU FP6 project 026133: <https://cordis.europa.eu/project/id/26133/> [accessed 2021-10-01]

<sup>27</sup><https://doc.sagemath.org/html/en/tutorial/sagetex.html> [accessed 2021-10-01]

translations since the input must be provided in the syntax of the CAS. Hence, a translation must be performed manually or via external tools.

In the introduction, we mentioned potential issues of CAS with multi-valued functions. Multi-valued functions map values from a domain to multiple values in a codomain and frequently appear in the complex analysis of elementary and special functions [8]. Prominent examples are the inverse trigonometric functions, the complex logarithm, or the square root. All modern CAS<sup>28</sup> compute multi-valued functions on their principle branches which makes these functions effectively single-valued (e.g., a calculator always returns 2 for  $\sqrt{4}$  rather than  $\pm 2$  or just  $-2$ ). The correct properties of multi-valued functions on the complex plane may no longer be valid by their counterpart functions on CAS, e.g.,  $(1/z)^w = 1/(z^w)$  for  $z, w \in \mathbb{C}$  and  $z \neq 0$  is no longer valid within CAS. The positioning and handling of branch cuts in CAS is often discussed in scientific articles and generally prominently noticed in CAS handbooks [83, 84, 91, 108, 171, 172]. However, especially in more complex scenarios, it is easy to lose track of branch cut positioning and evaluate expressions on incorrect values. We provide a more complex example and a more detailed explanation of branch cuts in Appendix A available in the electronic supplementary material. To the best of our knowledge, no available translation tool from, to, or between CAS (including the SCSCP solutions) consider branch cut positions.

### 2.2.3.2 Theorem Prover

The idea of automated reasoning and deduction systems is as old as computers [147]. With the power of computers and a strict axiomatic approach as in *Principia Mathematica* [385], computers can perform automatic reasoning steps to discover and proof new mathematical theorems. Up until today, automated theorem proving and verifying is an extensive research area with an ever-growing interest [266, 354, 384]. There are numerous theorem provers and proof assistants systems available, such as HOL Light [146], HOLF [340], or Isabelle [287]. However, focusing on the deduction, the encoding of theorem provers generally goes beyond mathematical expressions. The syntax provides specific options for assumptions, links between multiple concepts, and logical steps. An example of a proof by Isabelle, which clearly visualizes the different notation of theorem provers and CAS, is given in Appendix C available in the electronic supplementary material.

Nonetheless, theorem prover formats are computable formats with specific mathematical applications. Hence, there is a genuine interest in transferring findings and solutions from one system to the other. There are some translation approaches between theorem prover and CAS available, from direct translations [28, 148] to translations over OpenMath [57, 338] and OM-Doc [152]. Theorem provers are generally unable to *compute* a single mathematical formula in the sense of numeric computations or symbolic manipulations. Hence, we do not choose theorem provers as the target computable format for our desired translation process.

### 2.2.4 Images and Tree Representations

In the following, we briefly discuss formats with the specific visualization focus: images and tree representations. Especially older literature is often only available in digital scans, and many copies of publications do not provide access to the original L<sup>A</sup>T<sub>E</sub>X source. Images can be con-

---

<sup>28</sup>The authors are not aware of any example of a CAS which treats multi-valued functions without adopting principal branches.

sidered as the purest presentational format of mathematical expressions. Tree representations of math expressions, on the other hand, are more theoretical concepts to visualize the logical or presentational structure of math. Tree representations are primarily used for explanation purposes to underline or visualize an idea or concept. Parse trees, as a generated specific tree format of mathematical string inputs, on the other hand, play a crucial role in almost every mathematical software tool. Often, digital mathematical formats try to mimic the logical tree structure of math expressions. This is also one of the reasons why the web formats (MathML and OpenMath) use XML to encode mathematical content.

**Symbolic Layout, Operator, Parse, and Expression Trees** Mathematical expressions are often represented in tree structures. For example, MathML itself is an XML tree data structure. Moreover, mathematicians often have a logical but theoretical tree representation of a formula in mind in which numbers and identifiers are terminal symbols (leaves) and children of math operators, functions, and relations [192, 331]. These so-called *expression trees* are more or less a theoretical structure and are mainly used to visualize logical correlations and connections in mathematical expressions. Schubotz et al. [331] attempted to automate the visualization process of expression trees based on cross-referenced MathML data which resulted in VMEXT, a visualization tool for MathML. Figure 2.3 shows a possible expression tree visualization for the Jacobi polynomial definition in terms of the hypergeometric function.

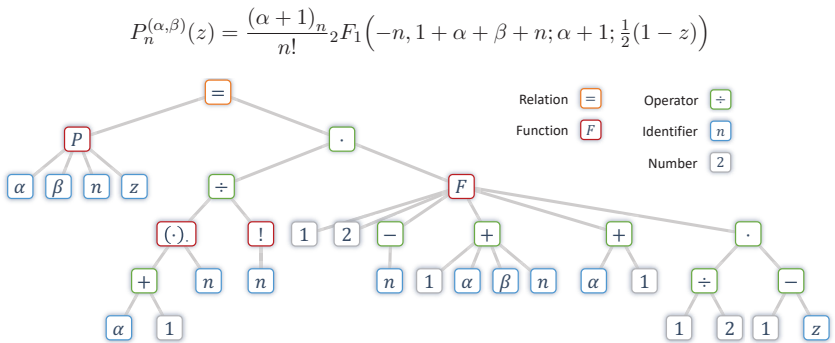


Figure 2.3: An expression tree representation of the explicit Jacobi polynomial definition in terms of the hypergeometric function.

For visualization and education purposes, these tree representations can be beneficial. However, generating these trees requires a deep understanding of the logical structure of the expression. In addition, there is no exact definition available for expression trees. Hence, the exact visualization is often up for discussions, e.g., whether parameters are children similar to variables or part of the function node itself [9]. A missing standard definition makes expression trees unreliable and, therefore, less practical for a mathematical encoding.

**Parse Trees** Parse trees are generated tree representations of source expressions (strings). These trees are generated by a parser that follows a strict set of rules, e.g., a context-free grammar [101, 188, 298]. Mathematical  $\text{\LaTeX}$  (as a subset of  $\text{\TeX}$ ) considering a couple simplifications

(e.g., no re-defined standard literals and macros) can also be described in a context-free grammar [402] even though  $\text{\TeX}$  itself is Turing complete [133, 135, 187]. The POM tagger [402], for example, parses mathematical  $\text{\TeX}$  following a context-free grammar. Similarly, Chien and Cheng [71] build a custom context-free grammar parser for their semantic tokenization of mathematical  $\text{\TeX}$  expressions.  $\text{\TeXML}$  follows the more sophisticated  $\text{\TeX}$ -like digestion methods [187] to parse entire  $\text{\TeX}$  files [133, 135]. CAS inputs are parsed internally for further processing [138, 392]. Maple’s internal parser also generates a parse tree in which equivalent nodes are merged together for more efficient memory usage (mathematically speaking, this data structure is no longer a valid tree but instead a directed, acyclic graph, or simply DAG) [3, 13].

In contrast to theoretical tree representations, such as the mentioned expression trees, parse trees are crucial for many applications because a tree data format is more easy to process due to their structural logic [93, 242, 286, 406]. While string sequences of commands may contain ambiguities, tree data structures are unique and provide easy access to single logical nodes, groups of nodes, and their dependencies. Hence, parsing a mathematical input (such as in CAS inputs or  $\text{\TeX}$  expressions) is typically the first step in any processing pipeline. Later in this thesis, we will also take advantage of tree representations by defining a translation between math formats as graph transformations on their tree representations. To generate a tree representation of mathematical  $\text{\TeX}$  formats, we can either build a custom parser [71] or rely on existing parsers, such as  $\text{\TeXML}$  [257] or the POM tagger [402]. Parse trees (and other custom generated tree formats that are generated by analyzing a given input) can also be categorized into symbol layout trees (for presentational formats) and operator trees (for content/semantic formats) [406]. For example, parsing  $\text{\TeX}$  may result in a symbol layout tree that describes the visual structure of formulae while parsing semantic  $\text{\TeX}$  (or CAS inputs) may result in operator trees which describe the logical mathematical structure of the input.

**Images** From pixel graphics (e.g., JPEG or PNG) to vector graphics (e.g., Scalable Vector Graphics (SVG)) and document formats (e.g., PDF), mathematical expression can appear in a variety of different image formats. The two-dimensional structure of mathematics makes drawing mathematical formulae on a sheet of paper or touch screens the most intuitive input method for mathematics. In addition, with rising digitization, scans of old scientific articles are no longer the only source of math images. Handwriting systems are more and more adopted in offices and educational institutions [411]. In 2016, Wikipedia switched from non-scalable PNG images to vector graphics for visualizing mathematics [17] (see Appendix B available in the electronic supplementary material, for a more sophisticated overview of the history of math formulae in Wikipedia).

However, image formats are not directly interpretable and are, therefore, less machine-readable. Hence, the first step of analyzing mathematics in images is always converting into a more machine-readable, digital format. The majority of conversion approaches, including handwriting recognition and Optical Character Recognition (OCR), focus on translations to MathML or  $\text{\TeX}$  [373, 406, 411]. Hence, for our task (translating presentational formats to computable formats), starting with image formats is not practically useful.

Nonetheless, one particular issue in math OCR is also of interest for our translation task: detection of inline mathematics. In image formats, detecting inline mathematics is difficult because formulae may blend into texts [74, 125, 126, 230, 398]. Even a detection of italic fonts

can be a challenging task [66, 112, 113, 233]. A variable can easily be confused with words, such as the Latin letter ‘a.’ A similar issue raises in other formats, including L<sup>A</sup>T<sub>E</sub>X documents and Wikipedia articles when an author does not correctly annotate mathematical formulae. In Wikipedia, for example, single identifiers in a text are often put in italic font rather than in mathematical environments. The capability of using UTF-8 encodings incites Wikipedia editors to put inline mathematics into the text directly, even when special characters are involved. For example, the mathematical expression  $0 \leq \phi \leq 4\pi$  in the English Wikipedia article about Jacobi polynomials<sup>29</sup> is a sequence of UTF-8 characters and thus challenging to identify as mathematics for MathIR parser. Nevertheless, identifying all mathematical expressions in a document might be necessary for more reliable translations towards computable formats. For example, the mentioned relation of  $\phi$  defines the domain of the Wigner d-matrix and is of interest for automatic evaluations (see Chapter 5).

## 2.2.5 Math Embeddings

Word embedding techniques has received significant attention over the last years in the Natural Language Processing (NLP) community, especially after the publication of word2vec [256]. Therefore, more and more projects try to adapt this knowledge for solving tasks in the MathIR arena [121, 15, 141, 215, 353, 360, 400, 404]. These projects try to embed math expressions into natural languages to create a vector representation of the formula. A vector representation is the data format with the highest machine readability among all other representations of mathematical formula. The math embeddings successfully enabled a new approach to measure the similarity between math expressions, which is especially useful for math search, classification, and similar tasks [121, 215, 400, 404].

Considering the equation embedding techniques in [215], we devise three main types of mathematical embedding: *Mathematical Expressions as Single Tokens*, *Stream of Tokens*, and *Semantic Groups of Tokens*. In the following we briefly explain each type on an example expression containing the inequality for Van der Waerden numbers

$$W(2, k) > 2^k / k^\varepsilon. \quad (2.1)$$

This expression is the first entry in the the MathML benchmark [18] we are going to explain in detail in Section 2.3.

**Mathematical Expressions as Single Tokens** So called equation embeddings (EqEmb) were introduced by Krstovski and Blei [215] and use an entire mathematical expression as one token. In a one-token representation, the inner structure of the mathematical expression is not considered. For example,  $W(r, k)$  is represented as one single token  $t_1$ . Any other expression, such as  $W(2, k)$  in the context, is an entirely independent token  $t_2$ . Therefore, this approach does not learn any connections between  $W(2, k)$  and  $W(r, k)$ . However, [215] has shown promising results for comparing mathematical expressions with this approach.

**Stream of Tokens** As an alternative to embedding mathematical expressions as a single token, one can also represent an expression through a sequence of its inner elements. For example, considering only the identifiers in Equation (2.1), it would generate  $W$ ,  $k$ , and  $\varepsilon$  as a sequence/stream of tokens. This approach has the advantage of learning all mathematical tokens.

<sup>29</sup>[https://en.wikipedia.org/wiki/Jacobi\\_polynomials#Applications](https://en.wikipedia.org/wiki/Jacobi_polynomials#Applications) [accessed 2021-10-01]

However, this method also has some drawbacks. Complex mathematical expressions may lead to long chains of elements, which can be especially problematic when the window size of the training model is too small. Naturally, there are approaches to reduce the length of chains. Gao et al. [121] use a continuous bag of words (CBOW) approach and embed all mathematical symbols, including identifiers and operands, such as  $+$ ,  $-$  or variations of equalities  $=$ . Krstovski and Blei [215] also evaluated the stream of tokens approach but do not cut out symbols. They trained their model on the entire sequence of tokens that the  $\text{\LaTeX}$  tokenizer generates. Considering Equation (2.1), it would result in a stream of 13 tokens. They use a long short-term memory (LSTM) architecture to overcome the limiting window size and further limit chain lengths to 20 – 150 tokens. Usually, in word embedding, such behaviour is not preferred since it increases the noise in the data.

We [15] also use this stream of tokens approach to train our model on the DLMF without any filters. Thus, Equation (2.1) generates all 13 tokens. Later in Section 3.1, we show another model trained on the arXiv collection, which uses a stream of mathematical identifiers and cut out all other expressions, i.e., in case of (2.1), we embed  $W$ ,  $k$ , and  $\varepsilon$ . We presume this approach is more appropriate to learn connections between identifiers and their definiens. We will see later that both of our models trained on math embedding are able to detect similarities between mathematical objects, but does not perform well on detecting connections to word descriptors.

**Semantic Groups of Tokens** The third approach of embedding mathematics is only theoretical. Current MathIR and Machine Learning (ML) approaches would benefit from a basic structural knowledge of mathematical expressions, such that variations of function calls (e.g.,  $W(r, k)$  and  $W(2, k)$ ) can be recognized as the same function. Instead of defining a unified standard, current techniques use their ad-hoc interpretations of structural connections. We assume that an embedding technique would benefit from a system that can detect the parts of interest in mathematical expressions before any training process. However, such a system still does not exist. Later in Section 3.2, we will introduce a new concept to interpret logical groups of mathematical objects that may enable a semantic embedding in the future.

It is important to mention that it remains unclear to what degree math semantic information can be embedded in a vector representation [9]. Since there is no answer to this question, we have not included math embeddings (i.e., vector representations of formulae) to Figure 2.1. Nonetheless, a vector representation can be decoded into a CAS syntax representation again to perform a ML based translation [296]. We will elaborate on such an approach more in Chapter 4.

## 2.3 From Presentation to Content Languages

We introduced several different formats for encoding mathematical formulae digitally and provided an overview of several existing conversion tools between these formats. Considering Figure 2.1, the goal of this thesis, i.e., making presentational math computable, requires to convert mathematical formats from the most left of the figure to the most right. We have chosen  $\text{\LaTeX}$  as the source format and general-purpose CAS syntaxes for the target formats. Considering the merit of communicating knowledge in sciences, it comes to no surprise that there are numerous of translation tools and theoretical approaches available to convert math formulae between multiple formats, including our goal translation from  $\text{\LaTeX}$  to CAS syntaxes.

Since MathML is the web standard which is supported by several CAS at least partially [57, 110, 303, 338] (or OpenMath respectively), a translation from  $\text{\LaTeX}$  to CAS could be performed over MathML (preferably content MathML). In this section, we analyze state-of-the-art  $\text{\LaTeX}$  to MathML converters to study the applicability of using MathML as an intermediate format for translations from  $\text{\LaTeX}$  to CAS syntaxes. This section was previously published [18].

### 2.3.1 Background

In the following, we use the Riemann hypothesis (2.2) as an example to explain typical challenges of converting different representation formats of mathematical formulae:

$$\zeta(s) = 0 \Rightarrow \Re s = \frac{1}{2} \vee \Im s = 0. \quad (2.2)$$

We will focus on the representation of the formula in  $\text{\LaTeX}$  and in the format of the CAS Mathematica.  $\text{\LaTeX}$  is a common language for encoding the presentation of mathematical formulae. In contrast to  $\text{\LaTeX}$ , Mathematica's representation focuses on making formulae computable. Hence the content must be encoded, i.e., both the structure and the semantics of mathematical formulae must be taken into consideration.

In  $\text{\LaTeX}$ , the Riemann hypothesis can be expressed using the following string:



#### Riemann hypothesis in $\text{\LaTeX}$

```
1 \zeta(s) = 0 \Rightarrow \Re s = \frac{1}{2} \vee \Im s = 0
```

In Mathematica, the Riemann hypothesis can be represented as:



#### Riemann hypothesis in Mathematica

```
1 Implies[Equal[Zeta[s], 0], Or[Equal[Re[s], Rational[1, 2]],
    Equal[Im[s], 0]]]
```

The conversion between these two formats is challenging due to a range of conceptual and technical differences.

First, the grammars underlying the two representation formats greatly differ.  $\text{\LaTeX}$  uses the unrestricted grammar of the  $\text{\TeX}$  typesetting system. The entire set of commands can be re-defined and extended at runtime, which means that  $\text{\TeX}$  effectively allows its users to change every character used for the markup, including the  $\backslash$  character typically used to start commands. The large degree of freedom of the  $\text{\TeX}$  grammar significantly complicates recognizing even the most basic tokens contained in mathematical formulae. In difference to  $\text{\LaTeX}$ , CAS use a significantly more restrictive grammar consisting of a predefined set of keywords and set rules that govern the structure of expressions. For example in Mathematica, function arguments must always be enclosed in square brackets and separated by commas.

Second, the extensive differences in the grammars of the two languages are reflected in the resulting expression trees. Similar to parse trees in natural language, the syntactic rules of mathematical notation, such as operator precedence and function scope, determine a hierarchical

structure for mathematical expressions that can be understood, represented, and processed as a tree. The mathematical expression trees of formulae consist of functions or operators and their arguments. We used nested square brackets to denote levels of the tree and Arabic numbers in a gray font to indicate individual tokens in the markup. For the  $\mathbb{L}\mathbb{T}\mathbb{E}\mathbb{X}$  representation of the Riemann hypothesis, the expression tree is:



### Representation tree of Riemann hypothesis in $\mathbb{L}\mathbb{T}\mathbb{E}\mathbb{X}$

$$\left[ \zeta_1^1 \left( \frac{2}{1} s_1^3 \right)_1^4 = \frac{5}{1} 0_1^6 \Rightarrow \frac{7}{1} \Re_1^8 s_1^9 = \frac{10}{1} \left[ \frac{11}{1} \frac{12}{1} \frac{13}{1} \right] \vee \frac{14}{1} \Im_1^{15} s_1^{16} = \frac{17}{1} 0_1^{18} \right].$$

The tree consists of 18 nodes, i.e., tokens, with a maximum depth of two (for the fraction command `\frac{12}`). The expression tree of the Mathematica expression consists of 16 tokens with a maximum depth of five:



### Representation tree of Riemann hypothesis in Mathematica

$$\left[ \Rightarrow \left[ \left[ \frac{19}{\zeta} \left[ \frac{20}{1} \frac{21}{s_1} \right] \right] 0_{n_1}^{23} \right] \left[ \frac{24}{\vee} \left[ \frac{25}{=} \left[ \frac{26}{\Re} \frac{27}{s_1} \right] \left[ \frac{28}{1} \frac{29}{n} \frac{30}{n} \right] \right] \left[ \frac{31}{=} \left[ \frac{32}{\Im} \frac{33}{s_1} \right] 0_n^{34} \right] \right] \right].$$

The higher complexity of the Mathematica expression reflects that a CAS represents the content structure of the formula, which is deeply nested. In contrast,  $\mathbb{L}\mathbb{T}\mathbb{E}\mathbb{X}$  exclusively represents the presentational layout of the Riemann hypothesis, which is almost linear.

For the given example of the Riemann hypothesis, finding alignments between the tokens in both representations and converting one representation into the other is possible. In fact, Mathematica and other CAS offer a direct import of  $\mathbb{T}\mathbb{E}\mathbb{X}$  expressions, which we evaluate in Section 2.3.3.

However, aside from technical obstacles, such as reliably determining tokens in  $\mathbb{T}\mathbb{E}\mathbb{X}$  expressions, conceptual differences also prevent a successful conversion between presentation languages, such as  $\mathbb{T}\mathbb{E}\mathbb{X}$ , and content languages. Even if there was only one generally accepted presentation language, e.g., a standardized  $\mathbb{T}\mathbb{E}\mathbb{X}$  dialect, and only one generally accepted content language, e.g., a standardized input language for CAS, an accurate conversion between the representation formats could not be guaranteed.

The reason is that neither the presentation language, nor the content language always provides all required information to convert an expression to the respective language. This can be illustrated by the simple expression:  $F(a + b) = Fa + Fb$ . The inherent content ambiguity of  $F$  prevents a deterministic conversion from the presentation language to a content language.  $F$  might, for example, represent a number, a matrix, a linear function or even a symbol. Without additional information, a correct conversion to a content language is not guaranteed. On the other hand, the transformation from content language to presentation language often depends on the preferences of the author and the context. For example, authors sometimes change the presentation of a formula to focus on specific parts of the formula or improve its readability.

Another obstacle to conversions between typical presentation languages and typical content languages, such as the formats of CAS, are the restricted set of functions and the simpler



grammars that CAS offer. While  $\text{T}_{\text{E}}\text{X}$  allows users to express the presentation of virtually all mathematical symbols, thus denoting any mathematical concept, CAS do not support all available mathematical functions or structures. A significant problem related to the discrepancy of the space of concepts expressible using presentation markup and the implementation of such concepts in CAS are branch cuts. Branch cuts are restrictions of the set of output values that CAS impose for functions that yield ambiguous, i.e., multiple mathematically permissible outputs. One example is the complex logarithm [98, (4.2.1)], which has an infinite set of permissible outputs resulting from the periodicity of its inverse function. To account for this circumstance, CAS typically restrict the set of permissible outputs by cutting the complex plane of permissible outputs. However, since the method of restricting the set of permissible outputs varies between systems, identical inputs can lead to drastically different results [3]. For example, multiple scientific publications address the problem of accounting for branch cuts when entering expressions in CAS, such as [109] for Maple.

Our review of obstacles to the conversion of representation formats for mathematical formulae highlights the need to store *both* presentation and content information to allow for reversible transformations. Mathematical representation formats that include presentation and content information can enable the reliable exchange of information between typesetting systems and CAS.

MathML offers standardized markup functionality for both presentation and content information. Moreover, the declarative MathML XML format is relatively easy to parse and allows for cross references between Presentation Language (PL) and Content Language (CL) elements. Listing 2.3 represents excerpts of the MathML markup for our example of the Riemann hypothesis (2.2). In this excerpt, the PL token 7 corresponds to the CL token 19, PL token 5 corresponds to CL token 20, and so forth.



### Riemann hypothesis in MathML

```

1 <math><semantics><mrow>...
2   <mo id="5" xref="20">=</mo>
3   <mn id="5" xref="21">0</mn>
4   <mo id="7" xref="19">=></ci>...</mrow>
5 <annotation-xml encoding="MathML-Content">
6   <apply><implies id="19" xref="7"/>
7   <apply><eq id="20" xref="5"/>...
8   <apply><csymbol id="21" xref="1" cd="wikidata">Q187235</csymbol>...
9 </annotation-xml></semantics></math>

```

Listing 2.3: MathML representation of the Riemann hypothesis (2.2) (excerpt).

Combined presentation and content formats, such as MathML, significantly improve the access to mathematical knowledge for users of digital libraries. For example, including content information of formulae can advance search and recommendation systems for mathematical content. The quality of these *mathematical information retrieval systems* crucially depends on the accuracy of the computed document-query and document-document similarities. Considering the content information of mathematical formulae can improve these computations by:

1. enabling the consideration of mathematical equivalence as a similarity feature. Instead of exclusively analyzing presentation information as indexed, e.g., by considering the overlap in presentational tokens, content information allows modifying the query and the indexed information. For example, it would become possible to recognize that the expressions  $a(\frac{b}{c} + \frac{d}{c})$  and  $\frac{a(b+d)}{c}$  have a distance of zero.
2. allowing the association of mathematical tokens with mathematical concepts. For example, linking identifiers, such as  $E$ ,  $m$ , and  $c$ , to energy, mass, and speed of light, could enable searching for all formulae that combine all or a subset of the concepts.
3. enabling the analysis of structural similarity. The availability of content information would enable the application of measures, such as derivatives of the tree edit distance, to discover structural similarity, e.g., using  $\lambda$ -calculus. This functionality could increase the capabilities of *math-based plagiarism detection systems* when it comes to identifying obfuscated instances of reused mathematical formulae [253].

Content information could furthermore enable interactive support functions for consumers and producers of mathematical content. For example, readers of mathematical documents could be offered interactive computations and visualizations of formulae to accelerate the understanding of STEM documents. Authors of mathematical documents could benefit from automated editing suggestions, such as auto completion, reference suggestion, and sanity checks, e.g., type and definiteness checking, similar to the functionality of word processors for natural language texts.

### 2.3.1.1 Related Work

A variety of tools exist to convert format representations of mathematical formulae. However, to our knowledge, Stamerjohanns et al. [351] presented the only study that evaluated the conversion quality of tools. Unfortunately, many of the tools evaluated by Stamerjohanns et al. are no longer available or out of date. Watt presents a strategy to preserve formula semantics in  $\text{\LaTeX}$  to MathML conversions. His approach relies on encoding the semantics in custom  $\text{\LaTeX}$  macros rather than to expand the macros [380]. Padovani discusses the roles of MathML and  $\text{\LaTeX}$  elements for managing large repositories of mathematical knowledge [278]. Nghiem et al. used statistical machine translation to convert presentation to content language [271]. However, they do not consider the textual context of formulae. We will present detailed descriptions and evaluation results for specific conversion approaches in Section 2.3.3.

Youssef addressed the semantic enrichment of mathematical formulae in presentation language. They developed an automated tagger that parses  $\text{\LaTeX}$  formulae and annotates recognized tokens very similarly to Part-of-Speech (POS) taggers for natural language [402]. Their tagger currently uses a predefined, context-independent dictionary to identify and annotate formula components. Schubotz et al. proposed an approach to semantically enrich formulae by analyzing their textual context for the definitions of identifiers [329, 330].

With their ‘math in the middle approach’, Dehay et al. envision an entirely different approach to exchanging machine readable mathematical expressions. In their vision, independent and enclosed virtual research environments use a standardized format for mathematics to avoid computations and transfers between different systems. [94].

For an extensive review of format conversion and retrieval approaches for mathematical formulae, refer to [326, Chapter 2].

### 2.3.2 Benchmarking MathML

This section presents MathMLben - a benchmark dataset for measuring the quality of MathML markup of mathematical formulae appearing in a textual context. MathMLben is an improvement of the gold standard provided by Schubotz et al. [329]. The dataset considers recent discussions of the International Mathematical Knowledge of Trust<sup>30</sup> working group, in particular the idea of a ‘Semantic Capture Language’ [165], which makes the gold standard more robust and easily accessible. MathMLben:

- allows comparisons to prior works;
- covers a wide range of research areas in STEM literature;
- provides references to manually annotated and corrected MathML items that are compliant with the MathML standard;
- is easy to modify and extend, i.e., by external collaborators;
- includes default distance measures; and
- facilitates the development of converters and tools.

In Section 2.3.2.1, we present the test collection included in MathMLben. In Section 2.3.2.2, we present the encoding guidelines for the human assessors and describe the tools we developed to support assessors in creating the gold standard dataset. In Section 2.3.2.3, we describe the similarity measures used to assess the markup quality.

#### 2.3.2.1 Collection

Our test collection contains 305 formulae (more precisely, mathematical expressions ranging from individual symbols to complex multi-line formulae) and the documents in which they appear.

**Expressions 1 to 100** correspond to the search targets used for the ‘National Institute of Informatics Testbeds and Community for Information access Research Project’ (NTCIR) 11 Math Wikipedia Task [329]. This list of formulae has been used for formula search and content enrichment tasks by at least 7 different research institutions. The formulae were randomly sampled from Wikipedia and include expressions with incorrect presentation markup.

**Expressions 101 to 200** are random samples taken from the NIST DLMF [98]. The DLMF website contains 9,897 labeled formulae created from semantic  $\text{\LaTeX}$  source files [77, 78]. In contrast to the examples from Wikipedia, all these formulae are from the mathematics research field and exhibit high quality presentation markup. The formulae were curated by renowned mathematicians and the editorial board keeps improving the quality of the formulae’s markup<sup>31</sup>. Sometimes, a labeled formula contains multiple equations. In such cases, we randomly chose one of the equations.

**Expressions 201 to 305** were chosen from the queries of the NTCIR arXiv and NTCIR-12 Wikipedia datasets. 70% of these queries originate from the arXiv [22] and 30% from a Wikipedia dump.

---

<sup>30</sup><http://imkt.org/> [accessed 2021-08-03]

<sup>31</sup><http://dlmf.nist.gov/about/staff> [accessed 2021-08-03]

All data is openly available for research purposes and can be obtained from: <https://mathmlben.wmflabs.org><sup>32</sup>.

2.3.2.2 Gold Standard

We provide explicit markup with universal, context-independent symbols in content MathML. Since the symbols from the default content dictionary of MathML<sup>33</sup> alone were insufficient to cover the range of semantics in our collection, we added the Wikidata content dictionary [328]. As a result, we could refer to all Wikidata items as symbols in a content tree. This approach has several advantages. Descriptions and labels are available in many languages. Some symbols even have external identifiers, e.g., from the Wolfram Functions Site, or from stack-exchange topics. All symbols are linked to Wikipedia articles, which offer extensive human-readable descriptions. Finally, symbols have relations to other Wikidata items, which opens a range of new research opportunities, e.g., for improving the taxonomic distance measure [336].

Our Wikidata-enhanced, yet standard-compliant MathML markup, facilitates the manual creation of content markup. To further support human assessors in creating content annotations, we extended the VMEXT visualization tool [331] to develop a visual support tool for creating and editing the *MathMLben* gold standard.

Table 2.3: Special content symbols added to  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}\mathbb{M}\mathbb{L}$  for the creation of the gold standard.

No.	Rendering	Meaning	Example IDs
1	$[x, y]$	commutator	91
2	$x_z^y$	tensor	43, 208, 226
3	$x^\dagger$	adjoint	224, 277
4	$x'$	transformation	20
5	$x^\circ$	degree	20
6	$x^{(dim)}$	contraction	225

For each formula, we saved the source document written in different dialects of  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  and converted it into content MathML with parallel markup using  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}\mathbb{M}\mathbb{L}$  [135, 257].  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}\mathbb{M}\mathbb{L}$  is a Perl program that converts  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  documents to XML and HTML. We chose  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}\mathbb{M}\mathbb{L}$ , because it is the only tool that supports our semantic macro set. We manually annotated our dataset, generated the MathML representation, manually corrected errors in the MathML, and linked the identifiers to Wikidata concept entries whenever possible. Alternatively, one could initially generate MathML using a CAS and then manually improve the markup.

Since there is no generally accepted definition of expression trees, we made several design decision to create semantic representations of the formulae in our dataset using MathML trees. In some cases, we created new macros to be able to create a MathML tree for our purposes using  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}\mathbb{M}\mathbb{L}$ <sup>34</sup>. Table 2.3 lists the newly created macros. Hereafter, we explain our decisions and give examples of formulae in our dataset that were affected by the decisions.

<sup>32</sup>Visit <https://mathmlben.wmflabs.org/about> for a user guide [accessed 2021-08-03].

<sup>33</sup><http://www.openmath.org/cd> [accessed 2021-08-03]

<sup>34</sup><http://dlmf.nist.gov/latexml/manual/customization/customization.latexml.html#SS1.SSS0.Px1> [accessed 2021-08-03]

- not assign Wikidata items to basic mathematical identifiers and functions like `factorial`, `\log`, `\exp`, `\times`, `\pi`. Instead, we left these annotations to the DLMF  $\LaTeX$  macros, because they represent the mathematical concept by linking to the definition in the DLMF and  $\LaTeX$ ML creates valid and accurate content MathML for these macros [GoldID 3, 11, 19, ...];
- split up indices and labels of elements as child nodes of the element. For example, we represent `i` as a child node of `p` in `p_i` [GoldID 29, 36, 43, ...];
- create a special macro to represent tensors, such as for  $T_{\alpha\beta}$  [GoldID 43], to represent upper and lower indices as child nodes (see table 2.3);
- create a macro for dimensions of tensor contractions [GoldID 225], e.g., to distinguish the three dimensional contraction of the metric tensor in  $g^{(3)}$  from a power function (see table 2.3);
- chose one subexpression randomly if the original expression contained lists of expressions [GoldID 278];
- remove equation labels, as they are not part of the formula itself. For example, in

$$E = mc^2, \quad (*)$$

the  $(*)$  is the ignored label;

- remove operations applied to entire equations, e.g., applying the modulus. In such cases, we interpreted the modulus as a constraint of the equation [GoldID 177];
- use additional macros (see table 2.3) to interpret complex conjugations, transformation signs, and degree-symbols as functional operations (identifier is a child node of the operation symbol), e.g., `*` or `\dagger` for complex conjugations [GoldID 224, 277], `S'` for transformations [GoldID 20], `30^\circ` for thirty degrees [Gold ID 30];
- for formulae with multiple cases, render each case as a separate branch [GoldID 49];
- render variables that are part of separate branches in bracket notation. We implemented the Dirac Bracket commutator  $[\ ]$  (omitting the index `_text{DB}`) and an anticommutator by defining new macros (see table 2.3). Thus, there is a distinction between a (ring) commutator  $[a,b] = ab - ba$  and an anticommutator  $\{a,b\} = ab + ba$ , without further annotation of Dirac or Poisson brackets [GoldID 91];
- use the command `\operatorname{}` for multi-character identifiers or operators [GoldID 22]. This markup is necessary, because most  $\LaTeX$  parsers, including  $\LaTeX$ ML, interpret multi-character expressions as multiplications of the characters. In general, this interpretation is correct, since it is inconvenient to use multi-character identifiers [54].

Some of these design decisions are debatable. For example, introducing a new macro, such as `\identifiername{}`, to distinguish between multi-character identifiers and operators might be advantageous to our approach. However, introducing many highly specialized macros is likely not a viable approach and exaggerated. A borderline example in regard to this problem is  $\Delta x$  [GoldID 280]. Formulae of this form could be annotated as `\operatorname{}`, `\identifiername{}` or more generally as `\expressionname{}`. We interpret  $\Delta$  as a difference applied to a variable, and render the expression as a function call.

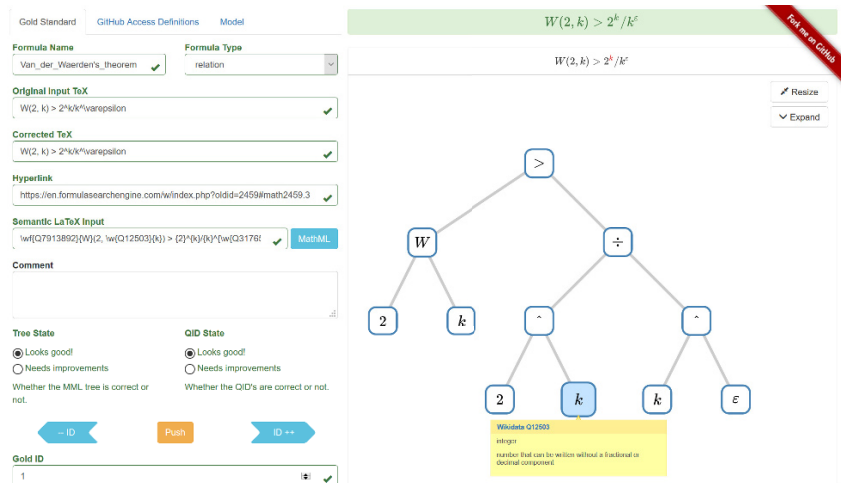


Figure 2.4: Graphical User Interface (GUI) to support the creation of our gold standard. The interface provides several  $\text{\LaTeX}$  input fields (left) and a mathematical expression tree rendered by the VMEXT visualization tool (right).

Similar cases of overfeeding the dataset with highly specialized macros are bracket notations. For example, the bracket (Dirac) notation, e.g., [GoldID 209], is mainly used in quantum physics. The angle brackets for the Dirac notation,  $\langle$  and  $\rangle$ , and a vertical bar  $|$  is already interpreted correctly as "latexml - quantum-operator-product". However, a more precise distinction between a twofold scalar product, e.g.,  $\langle a|b$ , and a threefold expectation value, e.g.,  $\langle a|A|a\rangle$ , might become necessary in some scenarios to distinguish between matrix elements and a scalar product.

We developed a Web application to create and cultivate the gold standard entries, which is available at: <https://mathmlben.wmflabs.org/>. The GUI provides the following information for each Gold ID entry.

- **Formula Name:** the name of the formula (optional)
- **Formula Type:** either *definition*, *equation*, *relation* or *General Formula* (if none of the previous names fit)
- **Original Input  $\text{\LaTeX}$ :** the  $\text{\LaTeX}$  expression extracted from the source
- **Corrected  $\text{\LaTeX}$ :** the manually corrected  $\text{\LaTeX}$  expression
- **Hyperlink:** the hyperlink to the position of the formula in the source
- **Semantic  $\text{\LaTeX}$  Input:** the manually created semantic version of the corrected  $\text{\LaTeX}$  field. This entry is used to generate our MathML with Wikidata annotations.

- **Preview of Corrected  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$ :** a preview of the corrected  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  input field rendered as an SVG image in real time using Mathoid [335], a service to generate SVGs and MathML from  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  input. It is shown in the top right corner of the GUI.
- **VMEXT Preview:** the VMEXT field renders the expression tree based on the content MathML. The symbol in each node is associated with the symbol in the cross referenced presentation MathML.

Figure 2.4 shows the GUI that allows to manually modify the different formats of a formula. While the other fields are intended to provide additional information, the pipeline to create and cultivate a gold standard entry starts with the semantic  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  input field.  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}\mathbb{M}\mathbb{L}$  will generate content MathML based on this input and VMEXT will render the generated content MathML afterwards. We control the output by using the DLMF  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  macros [260] and our developed extensions. The following list contains some example of the DLMF  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  macros.

- $\backslash\text{EulerGamma}@{z}$ :  $\Gamma(z)$ : gamma function,
- $\backslash\text{BesselJ}{\nu}@{z}$ :  $J_{\nu}(z)$ : Bessel function of the first kind,
- $\backslash\text{LegendreQ}[\mu]{\nu}@{z}$ :  $Q_{\nu}^{\mu}(z)$ :  
associated Legendre function of the second kind,
- $\backslash\text{JacobiP}{\alpha}{\beta}{n}@{x}$ :  $P_n^{(\alpha,\beta)}(x)$ :  
Jacobi polynomial.

The DLMF web pages, which we use as one of the sources for our dataset, were generated from semantically enriched  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  sources using  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}\mathbb{M}\mathbb{L}$ . Since  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}\mathbb{M}\mathbb{L}$  is capable to interpret semantic macros, generates content MathML that can be controlled with macros, and is easily extensible by new macros, we also used  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}\mathbb{M}\mathbb{L}$  to generate our gold standard. While the DLMF is a compendium for special functions, we need to annotate every identifier in the formula with semantic information. Therefore, we extended the set of semantic macros.

In addition to the special symbols listed in Table 2.3, we created macros to semantically enrich identifiers, operators, and other mathematical concepts by linking them to their Wikidata items. As shown in Figure 2.4, the annotations are visualized using yellow info boxes appearing on mouse over. The boxes show the Wikidata QID, the name, and the description (if available) of the linked concept.

Aside from naming, classifying, and semantically annotating each formula, we performed three other tasks:

- correcting the  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  string extracted from the sources;
- checking and correcting the MathML generated by  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}\mathbb{M}\mathbb{L}$
- visualizing the MathML using VMEXT

Most of the extracted formulae contained concepts to improve human readability of the source code, such as commented line breaks,  $\% \backslash n$ , in long mathematical expressions, or special macros to improve the displayed version of the formula, e.g., spacing macros, delimiters, and scale settings, such as  $\backslash!$ ,  $\backslash$ , or  $\backslash>$ . Since they are part of the expression, all of the tested tools (also  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}\mathbb{M}\mathbb{L}$ ) try to include these formatting improvements into the MathML markup. For our

gold standard, we focus on the pure semantic information and forgo formatting improvements related to displaying the formula. The corrected  $\text{\LaTeX}$  field shows the cleaned mathematical  $\text{\LaTeX}$  expression.

Using the corrected  $\text{\LaTeX}$  field and the semantic macros, we were able to adjust the MathML output using  $\text{\LaTeXML}$  and verify it by checking the visualization from VMEXT.

### 2.3.2.3 Evaluation Metrics

To quantify the conversion quality of individual tools, we computed the similarity of each tool's output and the manually created gold standard. To define the similarity measures for this comparison, we built upon our previous work [336], in which we defined and evaluated four similarity measures: taxonomic distance, data type hierarchy level, match depth, and query coverage. The measures taxonomic distance and data type hierarchy level require the availability of a hierarchical ordering of mathematical functions and objects. For our use case, we derived this hierarchical ordering from the MathML content dictionary. The measures assign a higher similarity score if matching formula elements belong to the same taxonomic class. The match depth measure operates under the assumption that matching elements, which are more deeply nested in a formula's content tree, i.e., farther away from the root node, are less significant for the overall similarity of the formula, hence are assigned a lower weight. The query coverage measure performs a simple 'bag of tokens' comparison between two formulae and assigns a higher score the more tokens the two formulae share.

In addition to these similarity measures, we also included the tree edit distance. For this purpose, we adapted the robust tree edit distance (RTED) implementation for Java [288]. We modified RTED to accept any valid XML input and added math-specific 'shortcuts', i.e., rewrite rules that generate lower distance scores than arbitrary rewrites. For example, rewriting  $\frac{a}{b}$  to  $ab^{-1}$  causes a significant difference in the expression tree: Three nodes ( $\wedge$ ,  $-$ ,  $1$ ) are inserted and one node is renamed  $\div \rightarrow \cdot$ . The 'costs' for performing these edits using the stock implementation of RTED are  $c = 3i + r$ . However, the actual difference is an equivalence, which we think should be assigned a cost of  $e < 3i + r$ . We set  $e < r < i$ .

## 2.3.3 Evaluation of Context-Agnostic Conversion Tools

This section presents the results of evaluating existing, context-agnostic conversion tools for mathematical formulae using our benchmark dataset MathMLben (cf. Section 2.3.2). We compare the distances between the presentation MathML and the content MathML tree of a formula yielded by each tool to the respective trees of formulae in the gold standard. We use the tree edit distance with customized weights and math-specific shortcuts. The goal of shortcuts is eliminating notational-inherent degrees of freedom, e.g., additional PL elements or layout blocks, such as `mrow` or `mfenced`.

### 2.3.3.1 Tool Selection

We compiled a list of available conversion tools from the W3C<sup>35</sup> wiki, from *GitHub*, and from questions about automated conversion of mathematical  $\text{\LaTeX}$  to MathML on *Stack Overflow*. We selected the following converters:

<sup>35</sup>[https://www.w3.org/wiki/Math\\_Tools](https://www.w3.org/wiki/Math_Tools) [accessed 2021-08-03]



- $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}\mathbb{M}\mathbb{L}$ : can convert generic and semantically annotated  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  expressions to XML/HTML/MathML. The tool is written in Perl [257] and is actively maintained.  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}\mathbb{M}\mathbb{L}$  was specifically developed to generate the DLMF web page and can therefore parse entire  $\mathbb{T}\mathbb{E}\mathbb{X}$  documents.  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}\mathbb{M}\mathbb{L}$  also supports conversions to content MathML.
- LaTeX2MathML: is a small python project and is able to generate presentation MathML from generic  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  expressions [245].
- Mathoid: is a service developed using Node.js, PhantomJS and MathJax (a javascript display engine for mathematics) to generate SVGs and MathML from  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  input. Mathoid is currently used to render mathematical formulae on Wikipedia [335].
- SnuggleTeX: is an open-source Java library developed at the University of Edinburgh [251]. The tool allows to convert simple  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  expression to XHTML and presentation MathML.
- MathToWeb: is an open-source Java-based web application that generates presentation MathML from  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  expressions<sup>36</sup>.
- TeXZilla: is a javascript web application for  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  to MathML conversion capable of handling Unicode characters<sup>37</sup>.
- Mathematical: is an application written in C and wrapped in Ruby to provide a fast translation from  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  expressions to the image formats SVG and PNG. The tool also provides translations to presentation MathML<sup>38</sup>.
- CAS: we included Mathematica, which is capable of parsing  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  expressions.
- Part-of-Math (POM) Tagger: is a grammar-based  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  parser that tags recognized tokens with information from a dictionary [402]. The POM tagger is currently under development. In this paper, we use the first version. In [3], this version was used to provide translations  $\mathbb{E}\mathbb{T}\mathbb{E}\mathbb{X}$  to the CAS Maple. In its current state, the program offers no export to MathML. We developed an XML exporter to be able to compare the tree provided by the POM tagger with the MathML trees in the gold standard.

### 2.3.3.2 Testing framework

We developed a Java-based framework that calls the programs to parse the corrected  $\mathbb{T}\mathbb{E}\mathbb{X}$  input data from the gold standard to presentation MathML, and, if applicable, to content MathML. In case of the POM tagger, we parsed the input string to a general XML document. We used the corrected  $\mathbb{T}\mathbb{E}\mathbb{X}$  input format instead of the originally extracted string expressions, see 2.3.2.2.

Executing the testing framework requires the manual installation of the tested tools. The POM tagger is not yet publicly available.

### 2.3.3.3 Results

Figure 2.5 shows the averaged structural tree edit distances between the presentation trees (blue) and content trees (orange) of the generated MathML files and the gold standard. To

<sup>36</sup><https://www.mathtowebonline.com> [accessed 2021-08-03]

<sup>37</sup><https://fred-wang.github.io/TeXZilla> [accessed 2021-08-03]

<sup>38</sup><https://github.com/gjtorikian/mathematical> [accessed 2021-08-03]

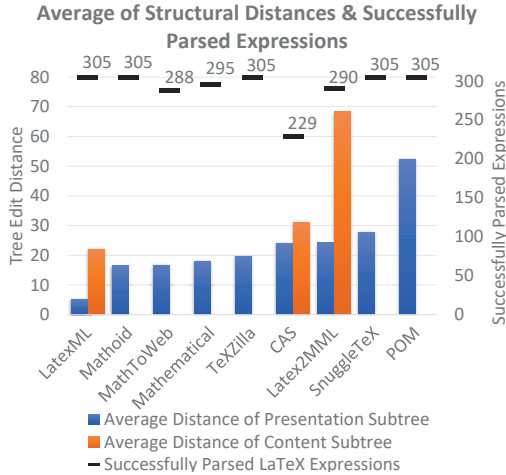


Figure 2.5: Overview of the structural tree edit distances (using  $r = 0$ ,  $i = d = 1$ ) between the MathML trees generated by the conversion tools and the gold standard MathML trees.

calculate the structural tree edit distances, we used the RTED [288] algorithm with costs of  $i = 1$  for inserting,  $d = 1$  for deleting and  $r = 0$  for renaming nodes. Furthermore, the Figure shows the total number of successful transformations for the 305 expressions (black ticks). Note that we also consider differences of the presentation tree to the gold standard as deficits, because the mapping from  $\text{\LaTeX}$  expressions to rendered expressions is unique (as long as the same preambles are used). A larger number indicates that more elements of an expression were misinterpreted by the parser. However, certain differences between presentation trees might be tolerable, e.g., reordering commutative expressions, while differences between content trees are more critical. Also note that improving content trees may not necessarily improve presentation trees and vice versa. In case of  $f(x + y)$ , the content tree will change depending whether  $f$  represents a variable or a function, while the presentation tree will be identical in both cases. In contrast,  $\frac{a}{b}$ ,  $a/b$ , and  $a/b$  have different presentation trees, while the content trees are identical.

Figure 2.6 illustrates the runtime performance of the tools. We excluded the CAS from the runtime performance tests, because the system is not primarily intended for parsing  $\text{\LaTeX}$  expressions, but for performing complex computations. Therefore, runtime comparisons between a CAS and conversion tools would not be representative. We measured the times required to transform all 305 expressions in the gold standard and write the transformed MathML to the storage cache. Note that the native code of LaTeX2MathML, Mathematical and  $\text{\LaTeX}$ ML were called from the Java Virtual Machine (JVM) and Mathoid was called through local web-requests, which increased the runtime of these tools. The figure is scaled logarithmically. We would like to emphasize that  $\text{\LaTeX}$ ML is designed to translate sets of  $\text{\LaTeX}$  documents instead of single mathematical expressions. Most of the other tools are lightweight engines.

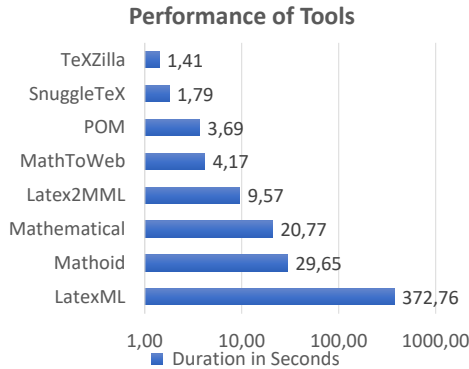


Figure 2.6: Time in seconds required by each tool to parse the 305 gold standard  $\text{\LaTeX}$  expressions in logarithmic scale.

In this benchmark, we focused on the structural tree distances rather than on distances in semantics. While our gold standard provides the information necessary to compare the extracted semantic information, we will focus on this problem in future work.

### 2.3.4 Summary of MathML Converters

We make available the first benchmark dataset to evaluate the conversion of mathematical formulae between presentation and content formats. During the encoding process for our MathML-based gold standard, we presented the conceptual and technical issues that conversion tools for this task must address. Using the newly created benchmark dataset, we evaluated popular context-agnostic  $\text{\LaTeX}$ -to-MathML converters. We found that many converters simply do not support the conversion from presentation to content format, and those that did often yielded mathematically incorrect content representations even for basic input data. These results underscore the need for future research on mathematical format conversions.

Of the tools we tested,  $\text{\LaTeX}ML$  yielded the best conversion results, was easy to configure, and highly extensible. However, these benefits come at the price of a slow conversion speed. Due to its comparably low error rate, we chose to extend the  $\text{\LaTeX}ML$  output with semantic enhancements.

## 2.4 Mathematical Information Retrieval for LaTeX Translations

In the following, we will briefly discuss related work in the Mathematical Information Retrieval (MathIR) arena in order to find existing practical approaches for a translation from presentational to computable formats. MathIR is the research area that aims to retrieve additional (generally semantic) information about mathematical content [141]. In turn, the task of translating mathematical presentational formats to computable formats is part of this research area

since it requires a context-dependent semantification<sup>39</sup>, i.e., the semantic enhancement or enrichment of mathematical objects with additional information. One of the most well-studied tasks in MathIR<sup>40</sup> is searching for relevant mathematical expressions or content [21, 22, 241, 346, 405, 408]. However, successful solutions in this area focus on similarity measures and do not necessarily require a deep understanding of the meaning and content of a formula. Likewise, other tasks in MathIR, such as entity linking, use similarity measures to retrieve connections between entities rather than semantic relatedness [208, 319, 321]. Thus, many related work in MathIR is not particularly beneficial for translating presentational encodings to computable formats. One of the reasons for this research gap is presumably a semantic version of the *chicken or the egg* causality dilemma. On the one hand, semantically enriching mathematical objects in an expression require identifying the meaningful objects. On the other hand, identifying those meaningful objects requires semantic information about those objects. In other words, if we want to annotate  $P_n^{(\alpha,\beta)}(x)$  with *Jacobi polynomial* in our use case equation (1.1), we need to know that  $P_n^{(\alpha,\beta)}(x)$  refers to the Jacobi polynomial.

Figure 2.7 illustrates this issue by splitting a math expression into four layers of mathematical objects. The identifier layer contains all identifiers (which may include general symbols and numbers too). The arithmetic layer contains arithmetic structures that combine tokens from the identifier layer to mathematical terms. This layer may include logic terms, sets, and other mathematical concepts with specific notations. The function layer combines elements from the lower layers to entire function calls. The top expression layer contains entire expressions in documents which are often a composition of elements in the previous layers. The difference of elements in the function and arithmetic layer is the ambiguity of the notations. Elements in the arithmetic layer generally do not need to be mapped to specific keywords in CAS because they are often semantically unique. In contrast, elements in the function layer are potentially ambiguous. However, a clear distinction between both layers is not always necessary and may even confuse in other MathIR related scenarios. For our task, the distinction is beneficial because elements in the function layer must be mapped to specific keywords in the CAS syntax, while elements in the arithmetic layer can be mostly ignored.

Existing MathIR tasks focus on semantically enhancing either the expression [208, 209, 215], arithmetic [93, 242, 339], or the identifier [121, 279, 329, 330, 339, 400] layer, missing the important function layer entirely. An algorithm needs to understand the involved functions to identify objects in the function layer. This dilemma is usually avoided in MathIR tasks since objects in the other layers can be extracted primarily context-independently. The meaning of arithmetic operators usually does not change (e.g., +, −, or /) and math identifiers can often be presumed to be Latin or Greek letters. The function layer, however, contains the most crucial objects for the translation task. Identifiers generally represent mutable objects, such as variables or parameters, and do not require specific mapping rules. Similarly, arithmetic operations are natively supported by most mathematical software. Finally, objects in the expression layers are often too abstract (because they are compositions of multiple objects) and cannot be mapped as a whole to a single logic procedure in a computable format.

There are approaches available that try to semantically enrich elements in the function layer. However, most of these semantic enrichment approaches focus solely on mathematical expressions themselves and do not analyze textual information [159, 259, 270, 339, 364, 374].

<sup>39</sup> Also often called *semantic enrichment*.

<sup>40</sup> For an extensive review of retrieval approaches for mathematical formulae, see also [326, Chapter 2].

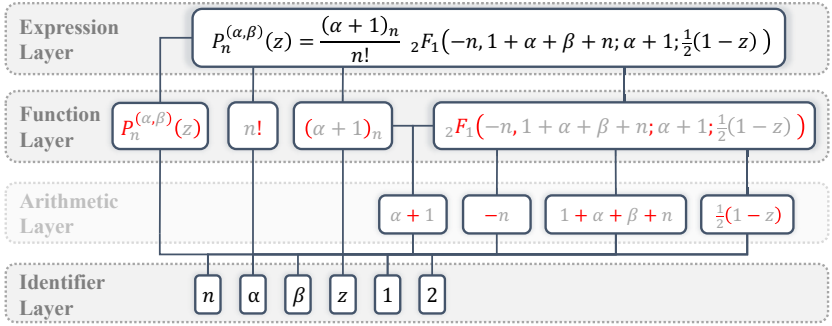


Figure 2.7: Four different layers of math objects in a single mathematical expression. The red highlights in the function and arithmetic layer refer to the fixed structure (or stem) of the function or operator. Gray tokens are mutable. Elements in the arithmetic layer are generally understood without further mappings and are mostly context-independent while elements in the function layer must be mapped to specific procedures in CAS and require disambiguation. However a strict distinction is not always required and might be even confusing. For example,  $n!$  is mostly understood by CAS and context-independent but can (and sometimes should) be mapped to the specific factorial procedure making it more to an element of the function layer.

Approaches that take the textual context of a formula into account, on the other hand, do not semantically enrich objects in the function layer. Instead, they focus on other specific applications, including math embeddings with the goal of a semantic vector representation [121, 215, 360, 400, 404], entity linking [208, 212, 316, 321], math word problem solving [285, 409], semantic annotation [183, 214, 279, 329, 330], and context-aware math search engines [93, 122, 124, 145, 210, 211, 232, 273, 314, 315, 366]. Regarding translating mathematical expressions from a lower level of semantics to a higher level, relevant literature is limited. The main relevant related literature for our task include semantic tagging [71, 402], annotations [139, 183, 214, 279, 329, 330], and term disambiguations [339]. In the following, we distinguish semantic tagging (the task of precisely tagging math objects with a pre-defined set of semantic tags) and semantic annotation (the task of adding any number of relevant descriptions to math objects).

**Semantic Tagging and Term Disambiguation** Semantic tagging of mathematical tokens has rarely been studied in the past and has not reached a well-established reliability level yet. To the best of our knowledge, only Chien et al. [71] (2015) and Youssef [402] (2017) addressed the issue for semantic tokenization of math formulae. Youssef [402] created the POM tagger, which tags tokens in the  $\LaTeX$  parse tree with additional information from a manually crafted lexicon. The POM tagger is still a work in progress and does not perform disambiguation steps yet. In the future, it is planned to reduce the number of possible tags for a token by analyzing the textual context and eliminating false tags. Ideally, the extracted context information results in a single, unique tag for each token. However, no update of the POM tagger, including the disambiguation steps, has been published so far. Recently, however, Shan and Youssef [339] presented several machine learning approaches as the first step towards disambiguation of mathematical terms. They trained different models on the semantic DLMF dataset and successfully disambiguated

prime notations with an  $F1$  score of 0.83. However, if the models only adapted the relatively strict DLMF notation style for primes or if they are also able to disambiguate other real-world data has not been discussed.

Chien et al. [71] proposed a probabilistic model on entire document collections to conclude semantic tags of mathematical tokens. They focused on tagging single identifiers (i.e., no groups of tokens). They constituted that the *consistency property* and *user habits* are critical aspects for successful tag disambiguation. With *user habits*, the authors referred to the different education levels and expertise of users so that a model can predict the preferred notation for specific semantics. The *consistency property* refers to the assumption that the meaning of a single term does not change within a certain context, e.g., a document. Recent efforts on annotating mathematical symbols by Asakura et al. [1], however, indicate that the scope of consistent tags could be significantly smaller than an entire document or a document collection. The semantics of frequently used symbols, such as  $x$  or  $t$ , may even change within single paragraphs. Another interesting counterexample is the connection between Euler numbers and Euler polynomials [98, (24.2.9)] in

$$E_n = 2^n E_n \left( \frac{1}{2} \right). \quad (2.3)$$

While clearly connected, the first  $E$  refers to the Euler number but the second  $E$  refers to Euler polynomials. This underlines that under special circumstances, even within the scope of a single equation, an identifier may refer to two different mathematical concepts. Chien et al. reported a maximum accuracy of 0.94.

**Semantic Annotation Task** While the task of semantic annotation has been studied more comprehensively, none of these existing approaches tried to convert the source expressions into a computable format [139, 183, 214, 279, 329, 330]. Grigore et al. [139], Nghiem et al. [269], Pagel et al. [279], Schubotz et al. [329, 330], and Kristianto et al. [214] analyze nouns or noun phrases in the surrounding context of a formula to semantically annotate an entire expression or parts of an expression. Only Grigore et al. [139] tried to use this information to perform a translation to a semantically enhanced format, here content MathML. The authors deduced a CD entry for a math symbol by calculating the similarity of the nouns surrounding the symbol and the textual description (or more precisely: the cluster of nouns in that description) of the CD entry. They measured the similarity with distributional properties from WordNet [261]. The other approaches either use the gained semantic information to improve search engines [214, 269] or enable entity linking [279, 329, 330]. While other semantification approaches exist that elevate source presentational formats to a semantically enriched format [245, 251, 257, 270, 271, 364, 391], none of them take the textual context into account. Some of them, however, perform disambiguation steps by considering other mathematical expressions in the same document (again presuming a semantic consistency of math notation within a single document as proposed by Chien et al. [71]) [270, 271]. None of the previous work considered the possibility of an identifier that has multiple meanings within a single formula, as shown in equation (2.3).

**Summary** In summary, semantic enriching approaches avoid the essential function layer [159, 259, 270, 364, 374], ignore the textual context surrounding a formula [71, 245, 251, 257, 270, 271, 296, 364, 391], or does not use the extracted information for a translation towards a semantic enhanced format [183, 214, 279, 329, 330, 402]. Nonetheless, the related work underlines the benefits of analyzing the textual context of a formula. More importantly, the research has

shown that even simple noun phrase extraction provide viable information for numerous of applications [139, 183, 214, 279, 329, 330]. This motivated us to apply these promising approaches for our semantification pipeline too.

Regarding the final translations towards computable formats, our comprehensive analysis of  $\text{\LaTeX}$  to MathML conversion tools in the previous section revealed that we probably gain no benefits from translating  $\text{\LaTeX}$  to MathML in an intermediate step. While many CAS provide import functions for MathML, there is no substantial support for OpenMath CDs. Another option would be OpenMath, since the SCSCP protocol uses OpenMath for inter-CAS communications. However, the SCSCP is relatively complex for our task and difficult to extend for new CAS if we do not have access to the internal libraries. Additionally, there are no translation tools from  $\text{\LaTeX}$  to OpenMath even though  $\text{\LaTeXML}$  can be exploited to realize rule-based translations.

In a previous research project, we developed  $\text{\LaTeX}$  to CAS translator, specifically for the DLMF [3, 13]. The goal of  $\text{\LaTeX}$  was to translate DLMF formulae, given in semantic  $\text{\LaTeX}$ , to the CAS Maple. The semantic  $\text{\LaTeX}$  macros reduced the ambiguity in mathematical expressions and enabled  $\text{\LaTeX}$  to focus on other translation issues, such as definition disparity between the DLMF and Maple. Hence, we already established a reliable and expandable translation pipeline from semantic  $\text{\LaTeX}$  to Maple. As a consequence, we focus our efforts on the more promising semantification of  $\text{\LaTeX}$  to semantic  $\text{\LaTeX}$  rather than from  $\text{\LaTeX}$  to content MathML in this thesis<sup>41</sup>.

This Chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>).




---

<sup>41</sup>Since the original development of  $\text{\LaTeX}$  was part of my Master's thesis, the content of the associated early publications [3, 13] is not reused in this thesis. For more details about  $\text{\LaTeX}$ , see [13].