

André Greiner-Petter

Master's Thesis

Semantic Preserving Bijective Mappings for Representations of Special Functions between Computer Algebra Systems and Word Processors

Exemplified using the DLMF/DRMF, the Word Processor LaTeX and the Computer Algebra Systems Maple and Mathematica

First Reviewer
Dr. Dr. Howard Cohl

Advisor
Dr. Moritz Schubotz

Second Reviewer
Prof. Dr. Michael Joswig

University
Technische Universität Berlin
Institute of Mathematics

May 22, 2018

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

_____

Berlin, den 22. Mai 2018

Abstract

This Master's thesis presents the development of a translation tool between representations of semantically enriched mathematical formulae in a Word Processor (WP) and its corresponding representations in a Computer Algebra System (CAS). The representatives for the CAS are Maple and Mathematica, while the representative WP is LaTeX.

Semantic information of a mathematical formula in LaTeX is rather given in the context than in the formula itself. Extracting the information is complicated and sometimes even impossible. However, a translation to a CAS is only achievable, if the semantic of the formula is unique. The National Institute of Standards and Technology (NIST) in the U.S. has developed a set of semantic LaTeX macros for all orthogonal polynomials and special functions (OPSF) defined in the Digital Library of Mathematical Functions (DLMF). Using these macros can disambiguate formulae and provide access to the semantic information.

Even if the semantics of a representation of one formula is unique, it does not need to match the semantics in another representation of the formula. For example, there are may be differences in the domains or a function in a CAS is normalized for a better performance. An important concept for complex and multivalued functions are *branch cuts*. However, the positions of these cuts are not consistently defined and can vary from system to system. Therefore, a CAS user needs to fully understand the properties and definitions of the used functions in the CAS. Hence, a manual translation from one system to another is not only laborious, but also prone to errors.

This thesis explains the realization of the translator, and discusses and suggests the approaches to the problems mentioned above.

Zusammenfassung

Im Rahmen der Masterarbeit wurde ein Programm entwickelt, welches semantisch angereicherte mathematische Formeln von einer Darstellung in einem Textverarbeitungssystem (WP) wie LaTeX zu einer entsprechenden Darstellung in einem Computeralgebrasystem (CAS) wie Maple oder Mathematica vor- und zurückübersetzen kann.

Semantische Informationen von mathematischen Formeln in LaTeX ergeben sich vor allem aus dem Kontext der Formeln. Diese Informationen zu extrahieren ist jedoch schwierig und in einigen Fällen bisher sogar unmöglich. Eine direkte Übersetzung zu einem CAS ist jedoch nur möglich, wenn die semantischen Informationen eindeutig sind. Das National Institute of Standards and Technology (NIST) in den USA hat dafür eine Liste von LaTeX Makros erstellt, welche semantische Informationen für jedes orthogonale Polynom und jede spezielle Funktion (OPSF) in der Digital Library of Mathematical Functions (DLMF) bereitstellen. Mit Hilfe dieser Makros können Doppeldeutigkeiten vermieden und der Zugang zu den semantischen Informationen erleichtert werden.

Selbst wenn die Semantik in einer Darstellung eindeutig ist, muss diese nicht mit der Semantik in einer anderen Darstellung übereinstimmen. Es kann Unterschiede in den Definitionsmengen geben oder eine Normalisierung definiert sein, um eine schnellere Berechnung zu ermöglichen. Bei mehrwertigen, komplexen Funktionen spielen sogenannte *Branch-Cuts* eine wichtige Rolle für CAS. Die Position dieser *Branch-Cuts* folgt zwar allgemeinen Konventionen, diese müssen aber nicht zwangsläufig eingehalten werden. Der Umgang mit einem CAS erfordert daher gute Kenntnisse über die speziellen Eigenschaften und Definitionen. Deshalb ist eine Übersetzung von einer Darstellung (z.B. LaTeX) zu einer Anderen (z.B. Maple) per Hand nicht nur mühselig, sondern auch fehleranfällig.

In der vorliegenden Arbeit wird die Umsetzung des Übersetzers erläutert und Lösungsansätze für die oben genannten Probleme werden erörtert.

Table of Contents

Chapter 1

Introduction

A typical workflow of a scientist who writes a scientific publication is to use a Word Processor (WP) to write the publication and one or more Computer Algebra System (CAS) for Verification, Analysis and Visualization, among other tasks. his formulae. Especially in Science, Technology, Engineering and Mathematics (STEM), LaTeX[1] has become the de facto standard for writing scientific publications over the last 30 years [3, 31, p. 559, 32]. LaTeX enables printing of mathematical formulae in a structure similar to handwritten style. For example, consider a Jacobi polynomial [21, (18.3), table 1]

$$P_n^{(\alpha,\beta)}(\cos(a\Theta)). \tag{1.1}$$

This formula is written in LaTeX as

```
P_n^{(\alpha,\beta)}(\cos(a\Theta)).
```
(1.2)

While LaTeX focuses on displaying mathematics, CAS focus on computations and user friendly syntax. Therefore, each system (such as LaTeX, Maple or Mathematica) uses its own representation and syntax. Hence, a writer needs to constantly translate mathematical expressions from one representation to another and back again. Table 1.1 shows four different representations for the same formula (1.1).

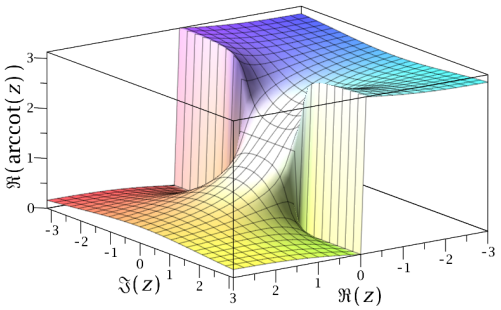
Such translations can become complex, if there are non-obvious differences between the used systems. For example, the CAS could use a different order of the arguments, define a normalization of the function for better performances or even use different domains for the variables. Multivalued functions are particularly difficult [20]. A CAS usually defines so called *branch cuts* to compute principal values of multivalued functions [24]. Thereby, implementations of multivalued functions are discontinuous. In general, positioning branch cuts follow some conventions. They are mostly defined conveniently, such as a straight line. However, since a branch

[1] Note that technically LaTeX is not a WP (https://www.latex-project.org/about/, seen 07/2017) like *Microsoft Word*. However, since LaTeX (an extension of TeX) is the standard to write in STEM and we only focus on mathematical writings in this thesis, we categorize LaTeX as a WP as well.

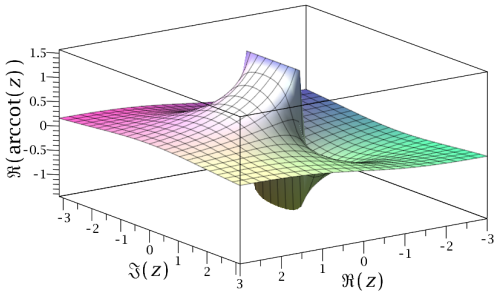
Systems	Representations
Generic LaTeX	`P_n^{(\alpha,\beta)}(\cos(a\Theta))`
Semantic LaTeX	`\JacobiP{\alpha}{\beta}{n}@{\cos@{a\Theta}}`
Maple	`JacobiP(n,alpha,beta,cos(a*Theta))`
Mathematica	`JacobiP[n,\[Alpha],\[Beta],Cos[a \[CapitalTheta]]]`

Table 1.1: Different representations for the same mathematical formula (1.1).

cut is a curve, it also can be defined as a spiral or it can be defined for a different position in the complex plane. The position of branch cuts can therefore varies from CAS to CAS [17]. Figure 1.1 illustrates two examples of different branch cut positioning for the inverse trigonometric arccotangent function. While Maple defines the branch cut at $[-\infty i, -i]$, $[i, \infty i]$ (figure 1.1a), Mathematica defines the branch cut at $[-i, i]$ (figure 1.1b).



(a) The real part of arccotangent with a branch cut at $[-\infty i, -i]$, $[i, \infty i]$.



(b) The real part of arccotangent with a branch cut at $[-i, i]$.

Figure 1.1: Two plots for the real part of the arccotangent function with a branch cut at $[-\infty i, -i]$, $[i, \infty i]$ in figure (a) and at $[-i, i]$ in figure (b), respectively. (Plotted with Maple 2016)

Hence, a CAS user needs to fully understand the properties and special definitions (such as the position of branch cuts) in the CAS to avoid mistakes during a translation [24]. In consequence, a manual translation process is not only laborious, but also prone to errors. Note that this general problem has been named to automatic Presentation-To-Computation (P2C) conversion [63].

An automatic translation process is desirable. Providing translations for mathematical LaTeX expressions is difficult, because the semantic information is absent. Semantics is the meaning of an expression. Consider the example of the Jacobi polynomial in (1.1). A scientific reader might be able to conclude information like P indicating the Jacobi polynomial and n is being a non-negative integer. However, for a computer it is difficult to understand the expression (1.2). Since LaTeX is the de facto standard for mathematical expressions, most CAS provide import and export functions for LaTeX expressions [41, 51, 42, 57]. However, because of the explained difficulties, those functions are very limited. In fact, they only work for expressions with unique semantic information, but fail for more complex expressions, such as our Jacobi polynomial example.

The National Institute of Standards and Technology (NIST) has therefore developed a set of semantic LaTeX macros [43]. Each of these macros is tied with a unique well-defined mathematical object and linked to the corresponding definition in the Digital Library of Mathematical Functions (DLMF) [21, 38, 46]. The semantic macros are used to semantically enhance the DLMF. An outgrowth of the DLMF project is the Digital Repository of Mathematical Formulae (DRMF) [14, 15], which defines more semantic macros, especially for Orthogonal Polynomials and Special Functions (OPSF). In table 1.1 we have already seen an example of a semantic macro. We call LaTeX expressions with semantic macros *semantic LaTeX* and without *generic LaTeX*, respectively. With semantic LaTeX, it is possible to provide translations to CAS.

Note that partial results of this Master's thesis have already been published at the 10th Conference on Intelligent Computer Mathematics 2017 [16]. Techniques, approaches and results have been roughly introduced in the article and are discussed in more detail in this thesis. Pictures and ideas that have already been presented there are not explicitly referenced in the following.

Furthermore, note that the translator we will present in this thesis is based on not published code from the Mathematical Language Parser (MLP) project and the Part-of-Math (POM)-tagger. We will also use the mentioned set of DLMF/DRMF LaTeX macros, which are not published yet as well. Therefore, the current version of the translator is not online available yet. However, we are planning to publish the translator soon on the DRMF website[2].

Moreover, I will use 'we' rather than 'I' in the subsequent chapters of this thesis, since I published and discussed my ideas with others including my advisors Dr. Moritz Schubotz and Professor Abdou Youssef, the hosts during research visits Howard S. Cohl and Jürgen Gerhard, and the students Joon Bang and Kevin Chen.[3]

1.1 Goals of the project

In this thesis, we will present an automatic tool for translations between semantic LaTeX and the CAS Maple and Mathematica. The translation tool should be lite, in a sense that a user does not need a whole software package to use the program. It is also a part of the DRMF project and should provide interactive translations in the DRMF. Furthermore, it should provide additional information about the translation process and present solutions if an appropriate translation is not possible.

Another goal is the translator's extensibility in order to provide the basis for future work.

[2] http://drmf.wmflabs.org

[3] Special thanks to M. Schubotz to give me the permission for using this phrase from his doctoral thesis [52].

1.2 Structure

In the following chapters, we will discuss our translator and verification techniques. Prior to explaining implementation details, we need to introduce some basic background information in chapter 2. We start this chapter with an overview of related work (section 2.1). Multivalued functions and branch cuts, which were already mentioned, will be introduced in section 2.2. Besides the mathematical background, we will discuss semantics in LaTeX in section 2.4. A brief introduction to grammar in languages and how we will use it to understand mathematical expressions will be given in section 2.6. The chapter is closed by some definitions for the following chapters.

The next chapter 3 is the main chapter of the thesis. We will explain implementations and discuss problems and solutions of and for the translator. It starts with an outline of the goals and our approaches in section 3.1. While we will present implementation details for the forward translations in section 3.2, we will present the developed backward translation in the following section 3.3.

In chapter 4 we will discuss, how we can verify or validate a translated expression. The chapter presents some approaches, such as round trip tests (section 4.1) and numerical tests (section 4.3). Extracted formulae from the DLMF and DRMF created a comprehensive test suite for these approaches. Section 4.4 gives a summary of the results.

The results and the current status of the translation tool will be discussed in chapter 5. In conclusion, chapter 6 explains some approaches to further improve the explained techniques in future work.

Chapter 2

Background & Related Work

Before we start to explain the translation process, we need to introduce some basic information. We will see that it is not as easy as it seems to explain what semantic information is, that there can be hidden complexity even behind simple mathematical expressions, and why we need grammar to understand mathematics. All this requires a comprehensive introduction.

In this chapter we will lay the foundation to understand the complexity and our solutions in this project. The first section 2.1 talks about the ideas and problems of related work. In the following section 2.2, we will give an introduction to complex analysis, branch cuts and special functions in general. Sections 2.3 and 2.4 will introduce the DLMF and semantic LaTeX, which is the most important part for our translations to CAS. Section 2.5 will give a brief introduction to CAS and mainly discusses Maple and Mathematica. The MLP is another important part of the translation process, because it gives us the opportunity to parse mathematical expressions. Section 2.6 explains how this works and why this is so important for a translator. This chapter finishes then with some extra definitions for the rest of the thesis.

Later on we will focus on the translation process between mathematical expressions in semantic LaTeX and in CAS, where the source of semantic LaTeX is the DLMF and the source for CAS expressions is the corresponding CAS. Because there is no umbrella term for those representations from different sources, we hereafter call them *systems*. A translation between semantic LaTeX and Maple is therefore in general a translation between two different systems.

2.1 Related Work

Since LaTeX became the de facto standard for writing papers in mathematics, most of the CAS nowadays are able to import and export mathematical LaTeX expressions [41, 51, 42, 57]. Those tools have two essential problems. First of all, they cannot import mathematical expressions where the semantic information is absent, which is the usual case for generic LaTeX expressions. Therefore, they are only able to import simple mathematical expressions, where the semantics

are unique. For example, the internal LaTeX macro `\frac` always indicates a fraction. These import tools fail for more complex expressions, for example the Jacobi polynomial in table 1.1. The second problem appears in the export tools. Mathematical expressions in CAS are fully semantic, otherwise the CAS wouldn't be able to compute or evaluate the expressions. During the export process, the semantic information gets lost, because generic LaTeX is not able to carry semantic information. In consequence of these two problems, an exported expression cannot be imported to the same system again in most cases (except those simple expressions described above). Our tool should solve these problems and provide round trip translations between LaTeX and CAS.

Obviously, the semantics must be well known before an expression can be translated. There are two approaches to solve that problem: someone could specify the semantic information during the writing process (pre-define semantics) or the translator can find out the right semantic information in general mathematical expressions before it translates the expression. The second approach is unreasonably difficult compared to the pre-definition approach. We will talk about those difficulties in the following sections. We will also introduce a new approach in chapter 6 to achieve the second idea.

However, the pre-definition approach is simple and easy to realize, as long as the system can carry semantic information. A typical, not mathematical example are Wikipedia articles [26]. An author of such an article can specify further information about symbols, words and sentences by creating a hyperlink to a more detailed explanation. Usually, the author defines the hyperlink, when he is writing the article or he needs to manually add the hyperlink later on. Similar to this approach are *interactive documents*[1], such as the Computable Document Format (CDF) [50] by Wolfram Research or the *worksheets* of Maple, which are also sometimes referred to as interactive documents. Those documents pre-define the semantics to allow computations. Those complex document formats require specialized tools to show and work with the documents (Wolfram CDF Player, or Maple for the *worksheets*). The `JOBAD` architecture [28] is able to create web based interactive documents and uses Open Mathematical Documents (OMDoc) [33] to carry semantics. The documents can be viewed and edited in the browser. Those `JOBAD`-documents also allow to perform computations during CAS. This gives the opportunity to calculate, compute and change mathematical expressions directly in the document. The translation performs in the background, invisible for the user. Similar to the `JOBAD` architecture other interactive web documents exist, such as *MathDox* [18] and *The Planetary System* [35]. All of them demonstrate the potential for the education system.

All of these systems carry semantic information. Most of them create their own way to make this possible. The web based documents mostly use standardized ways, such as the content Mathematical Markup Language (MathML), a specialization of MathML [5], that focuses on the semantics of an expression rather than any particular rendering for the expression. Since LaTeX is the de facto standard to write mathematical expressions in documents there are projects to extend LaTeX in a way that enable to carry the semantic information directly in LaTeX. There are

[1] There is no adequate definition what interactive documents are. However, this name is widely used to describe electronic document formats that allow interactivity to change the content in real time.

two known projects that created a semantic version of LaTeX: SLaTeX [34] developed by Michael Kohlhase and the DLMF/DRMF LaTeX macros developed by Bruce Miller. Our translator uses the DLMF/DRMF LaTeX macros rather than SLaTeX. A full explanation of this decision will be given in section 2.4.

However, *interactive documents*, web based or not, have an essential problem: all of them create an entire new document format. We want to create an independent lite translation tool for LaTeX. Once we achieve this goal, an integration into *interactive documents* might be possible, besides many other possible applications.

Another approach that tries to avoid the translation problem are LaTeX packages that allow computations directly through the LaTeX compiler (for example *LaTeXCalc* [13]) or allow CAS commands for computations directly in TeX-files. For example, *sagetex* [22] is an interface LaTeX package for the open source CAS *sage*[2]. This package allows *sage* commands in TeX-files and uses *sage* in the background to compute the commands. Obviously, the first example is not as powerful as a CAS would be. However, *sagetex* does not really solve our problem of the scientific workflow. The writer still needs to translate expressions manually. The only difference is that the author can write computations directly into the TeX-file. However, these approaches have the potential for a powerful combination with our translation tool.

In summary, today's translation tools are either specialized for one specific CAS (such as the import and export functions of the CAS) or they completely avoid the translation process by creating a new document format (such as *interactive documents*). Our goal is to provide a lite translation tool for mathematical LaTeX expressions and multiple CAS.

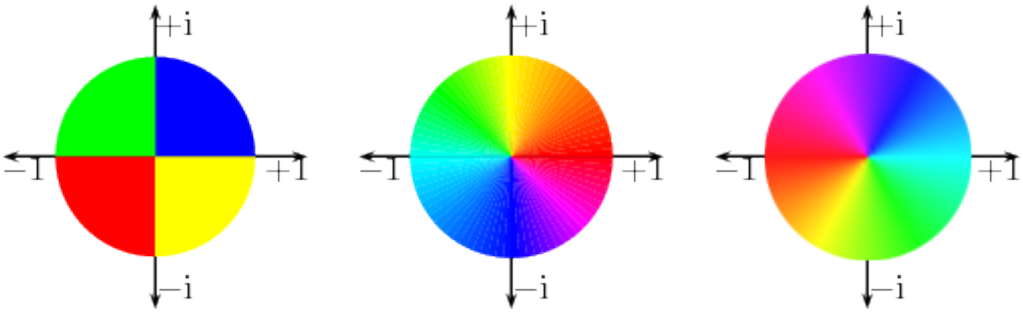
2.2 Mathematical Background

To understand the following chapters in the thesis, we need a brief introduction in complex analysis and multivalued functions. The following mathematical introduction is based on the book *Introduction to Complex Analysis* by H. A. Priestley [49].

A complex number $z \in \mathbb{C}$ is specified by a two real numbers x and y as $z = x + yi$, with i as the imaginary unit, which is defined as a solution of $i^2 = -1$. The two real numbers are called the real part $x = \Re(z)$ and the imaginary part $y = \Im(z)$ of z. It is convenient to represent complex numbers as points in a $\mathbb{R}^2$ plane, which is called **complex plane**. We identify $z = x + yi$ as $(x, y) \in \mathbb{R}^2$ in the complex plane. It is sometimes useful to represent complex numbers by polar coordinates and write $x = r\cos\phi$ and $y = r\sin\phi$, with $r \geq 0$ and $\phi \in \mathbb{R}$. Since $\cos\phi + \sin\phi i$ can be written as $e^{i\phi}$, we write $z = x + yi$ in polar coordinates as $z = re^{i\phi}$. The angle ϕ is often called the **phase** (or argument) of z and r is the absolute value of z. Similar to $\Re(z)$ for the real part and $\Im(z)$ for the imaginary part of z, we can access $r = |z|$ and $\phi = \mathrm{ph}(z)$.[3]

[2] An abbreviation for *SageMath*.

[3] Note that the phase of z is often called the **argument** of z. Therefore, in CAS the phase function is often the `argument` function.



(a) Four color phase mapping.

(b) Continues phase mapping in DLMF.

(c) Continues phase mapping in Maple.

Figure 2.1: Three different domain coloring methods. Mappings (a) and (b) used in the DLMF (Source: DLMF [21] in help page, *About Color Map*). Mapping (c) is the default coloring map in Maple.

Since the same basic arithmetical rules apply in $\mathbb{C}$ as in $\mathbb{R}$, we will not introduce all arithmetical rules for complex numbers here. However, another interesting part are plots of complex functions. Consider a function $f : \mathbb{C} \mapsto \mathbb{C}$. This can be considered as $f : \mathbb{R}^2 \mapsto \mathbb{R}^2$, which shows us four real dimensions that need to be displayed in a three-dimensional space. The most common solution to plot those four dimensions is called domain coloring. With this technique, each value in the complex plane is represented by a color. There is no standard method to map complex values to colors. A simple method is to map a unique color for each of the quadrants in the complex plane (used in the DLMF). Another variation are continues phase mappings, which use color wheels. In those cases, ϕ represents the hue and optionally r defines the intensity of the hue. The second variation is mostly used in CAS, but there is no standard for the orientation of the color wheel. Figure 2.1 draws three different examples.

This thesis mostly focus on the CAS Maple, therefore all of the following complex plots use the default color map in Maple 2.1c.

A three-dimensional plot of a complex function, such as f from above, usually maps the real and imaginary part of the variable to the x and y axes, while the height is defined by the absolute value of the complex variable $|z|$. Figure 2.2 demonstrates a complex plot of the parabolic cylinder function $U(a, z)$ [21, (12.2i)] in Maple.

Another way to plot such complex functions is to plot the real and imaginary parts of the solution separately. Figure 2.3 illustrates this plotting method for the parabolic cylinder function again.

Note that for real function plots, such separate plots for the imaginary and real values of a complex function, will use a color map as well. In that case, the color map is just an adornment and has nothing to do with complex color maps. Figure 2.3 illustrates the separate plots for real and imaginary values for the parabolic cylinder function again.

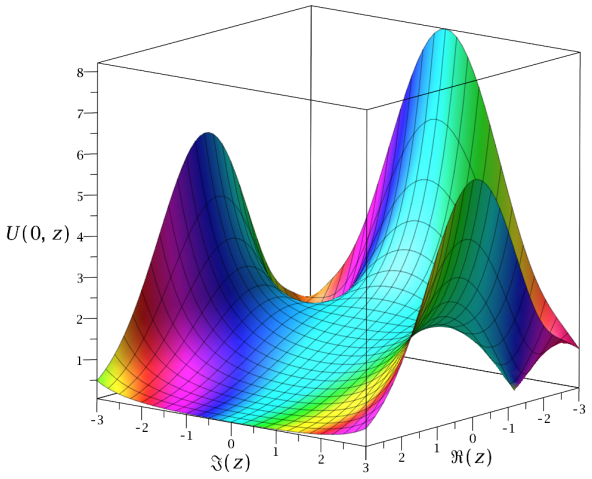


Figure 2.2: A 3-dimensional complex plot of the parabolic cylinder function $U(a, z)$ at $a = 0$ plotted by Maple. [21, (12.2i)]

2.2.1 Multivalued Functions

The following definitions for multivalued functions are based on *Complex Variables* [1] by M. J. Ablowitz and A. S. Fokas.

Unlike a typical[4] function, multivalued functions associate not only one output to any particular input, but a whole set of output values. A real-valued example is the square root function, which associates for each positive real number two real square roots. For example[5], the solution of $4^{\frac{1}{2}}$ is $\{+2, -2\}$.

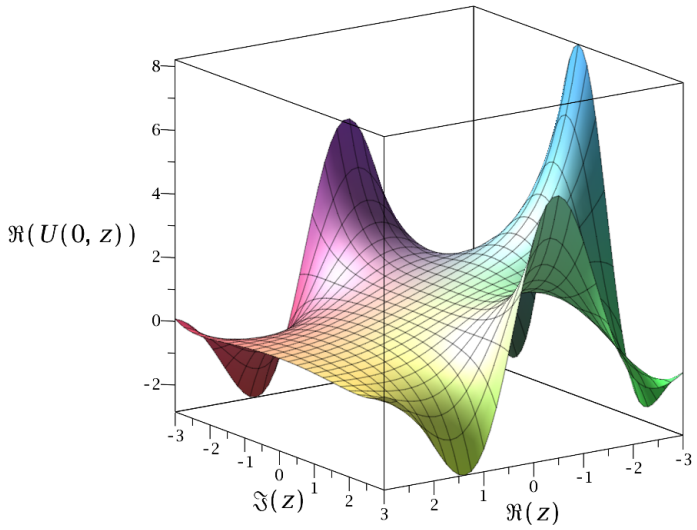
Consider now the natural logarithm with complex arguments and $z = re^{i\phi}$ in polar coordinates. Since it is the natural logarithm, $\ln(z) = \ln(r) + i\phi$ and the imaginary part is just ϕ. Let $r = 1$, so that z is on the unit circle for all ϕ. Let be z_1 the complex number for $\phi = 0$ and z_2 the complex number for $\phi = 2\pi$. Obviously $z_1 = z_2$ but $\ln(z_1) \neq \ln(z_2)$. Therefore, the natural logarithm is also multivalued and has an infinite number of solutions for $\ln(z)$.

The center of the circle we choose (in that case the origin) is called branch point.

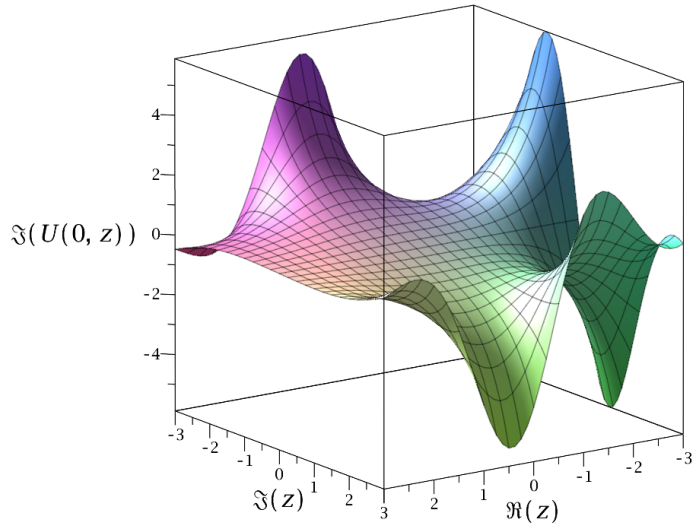
Definition 2.1: *(Branch Point)*

[4] In lack of a better word to distinguish single-valued and multivalued functions, we call single-valued functions here typical functions, because these functions are best known.

[5] Note that the radical sign $\sqrt{}$ is defined for the principal square roots, which are the unique non-negative square roots. Therefore, $\sqrt{4}$ is always 2, and not -2. However, the solutions of the square root of a positive number a are $\left\{+\sqrt{a}, -\sqrt{a}\right\}$.



(a) Plot of the real value of $U(0, z)$.



(b) Plot of the imaginary value of $U(0, z)$.

Figure 2.3: Separately plot of the real (a) and imaginary (b) part of the parabolic cylinder function $U(a, z)$ at $a = 0$.

*A point p is a **branch point** if the multivalued function $w(z)$ is discontinuous upon traversing an arbitrarily small circuit around p.*

The complex square root function has also a branch point in the origin. Consider

$$\sqrt{z} = \sqrt{r}e^{\frac{i\phi}{2}}, \quad \text{with } r \geq 0, \phi \in \mathbb{R}. \tag{2.1}$$

Indeed, at $\phi = 2\pi$, the square root of z is $\sqrt{r}e^{i\pi} = -\sqrt{r}$, but at $\phi = 0$ it is $\sqrt{r}$. Because r was arbitrary, the origin is also a branch point of the complex square root function. Another branch point is $z = \infty$.

There exist two approaches to analytically study multivalued functions. One approach is to express the multivalued function as a single-valued function.

Branch Cuts

Consider a multivalued function in a restricted region and choose for every argument a value, such that the resulting function in the region is continues and single-valued. This regional function of the multivalued function is called **branch** of the multivalued function.

Consider the complex square root function again. If we cut out the region between both branch points, namely $(0, \infty)$, the function is continues and single-valued. This region is one branch of the function and the cut region $(0, \infty)$ is called **branch cut**. Another branch of the function can be reached by further increasing ϕ, such that $\phi \in [2\pi, 4\pi)$. At $\phi = 4\pi$ the square root function returns the same result as for $\phi = 0$. Therefore, $\phi \in [4\pi, 6\pi)$ is again on the first branch. We can define a branch cut for the complex logarithm as well. As already mentioned, the complex logarithm has an infinite number of branches. Each of these branches can be reached by increasing or decreasing ϕ with a multiple of 2π. Consider $\phi + n2\pi$ with an integer number n. The branch for $n = 0$ is called the **principal branch**.

Defining branch cuts to allow analytic computations on the principal branches is a common task for CAS [24]. Unfortunately, the position of the branch cuts is not well-defined. Indeed, consider $\phi \in [-\pi, \pi)$ instead of $\phi \in [0, 2\pi)$ and define the branch cut at $(-\infty, 0)$; than the branch is also continues and single-valued.

In general a branch cut is a curve which ends can be possibly open, half-open or closed. A branch cut can be also defined in a more complicated shape (e.g. spirals). Most of the multivalued functions have standard, convenient positions for the branch cuts, which are commonly accepted. The branch cut at $(-\infty, 0]$ for the complex square root function is one of those standard positions. However, the positions can vary from system to system [17].

Figure 2.4 draws the principal branch of the imaginary part of the complex square root function in Maple.

The plot of the principal branch for the complex logarithm looks similar. The branch cut for the logarithm is in Maple defined at $(-\infty, 0)$. Figure 2.5 illustrates the principal branch of the imaginary part of the logarithm.

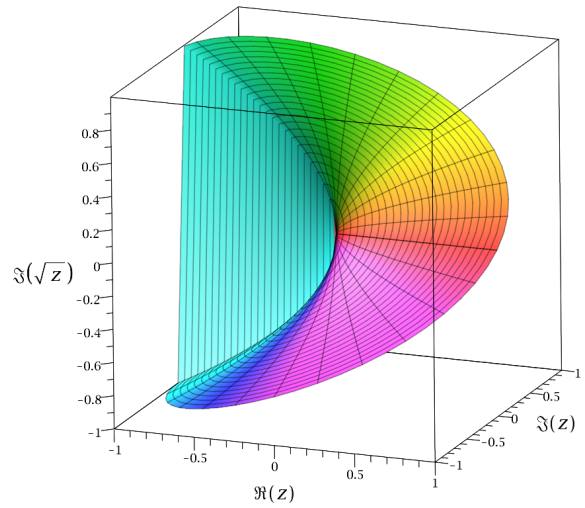


Figure 2.4: The imaginary part of the complex square root function with a branch cut at $(-\infty, 0)$. The resulted function is continues and single-valued for $\phi \in (-\pi, \pi)$.

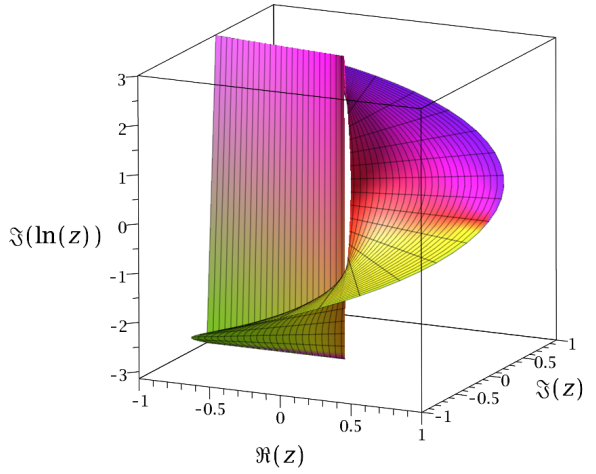


Figure 2.5: The imaginary part of the complex logarithm with a branch cut at $(-\infty, 0]$. The resulted function is continues and single-valued for $\phi \in (-\pi, \pi)$.

Note that in complex plots, a branch cut can also be located in the color map, we were talking about above. In the next example, we use the notation ± 0 to come arbitrarily close to the branch cut from the *upper* and *lower* side. We presume the one-side limits is a well known notation and

define

$$-0 := \lim_{x \nearrow 0} x, \tag{2.2}$$

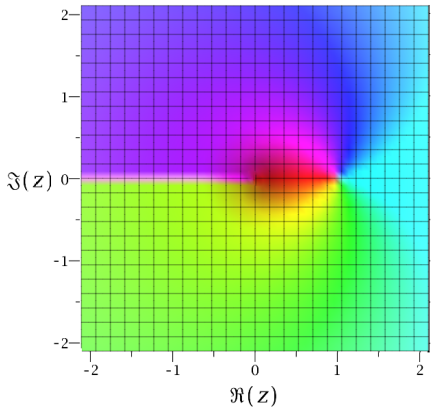
$$+0 := \lim_{x \searrow 0} x. \tag{2.3}$$

The natural logarithm $\ln(z)$ has a branch cut at $(-\infty, 0]$. We have the following values for the *upper side* and *lower side* of the branch cut at $\ln(-1)$

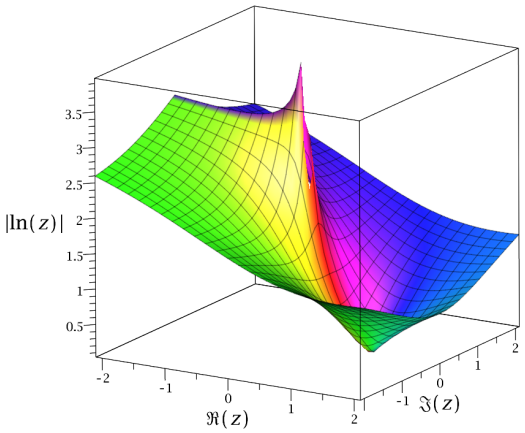
$$\ln(-1 + i0) = \quad i\pi, \tag{2.4}$$

$$\ln(-1 - i0) = -i\pi. \tag{2.5}$$

Figure 2.6 shows two plots of the natural logarithm. The value of (2.4) is represented by a shade of purple and (2.5) is represented by a shade of green, with Maple's continues phase color map from figure 2.1c. The branch cut can be seen by the changing of the color from purple directly to green on $(-\infty, 0]$. The colors are counterparts of each other in the used color map.



(a) The complex plot for the natural logarithm $\ln(z)$. The values of $\ln(z)$ are visualized by colors. The coloring method is the continues phase mapping from Maple.



(b) The complex plot for the natural logarithm $\ln(z)$. The height is the absolute value. The coloring method is the continues phase mapping from Maple.

Figure 2.6: The complex plot of the natural logarithm $\ln(z)$. Figure (a) plots only the colors of $\ln(z)$. The coloring method is Maple's continues phase mapping from figure 2.1c. The branch cut of $\ln(z)$ is at $(-\infty, 0)$ and the values $\ln(1 + i0) = i\pi$ (which is mapped to purple), while $\ln(1 - i0) = -i\pi$ (which is represented by green).

An alternative representation for multivalued functions to branch cuts are Riemann surfaces.

Riemann Surfaces

Assume we would not cut the multivalued functions and allow multiple values for a particular input. For example, if we allow ϕ to constantly increase in the complex logarithm function, we end up in helix structure as plotted in figure 2.7. This representation of a multivalued function is called the Riemann surface. Technically, a Riemann surface is a one-dimensional complex manifold.

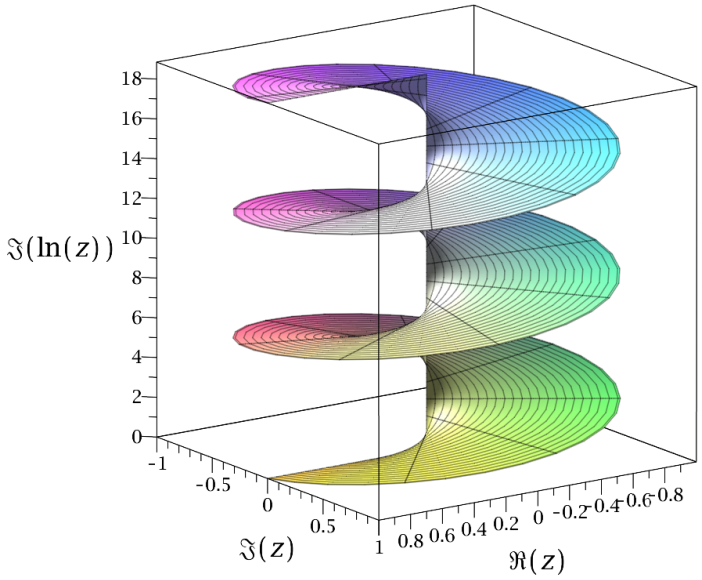


Figure 2.7: The Riemann surface of the imaginary part of the natural logarithm.

In the Riemann surface of logarithm in figure 2.7 we can see the multiple branches of the function. If we use the same method for the square root function, we get figure 2.8, where we can see that the square root function has only two branches.

2.2.2 Elementary & Special Functions

A function is called **elementary** if it can be constructed using only a finite combination of constant functions, field operations, and algebraic, trigonometric, exponential, and logarithmic functions together with their inverses [12, 56, p. 145].

Unlike elementary functions, special functions cannot be listed easily. They are mostly solutions of differential equations or integrals of elementary functions. Almost all of them have established names and notations. Many of them are complex and some are also multivalued. Besides that, special functions are highly related to each other. Therefore, most special functions

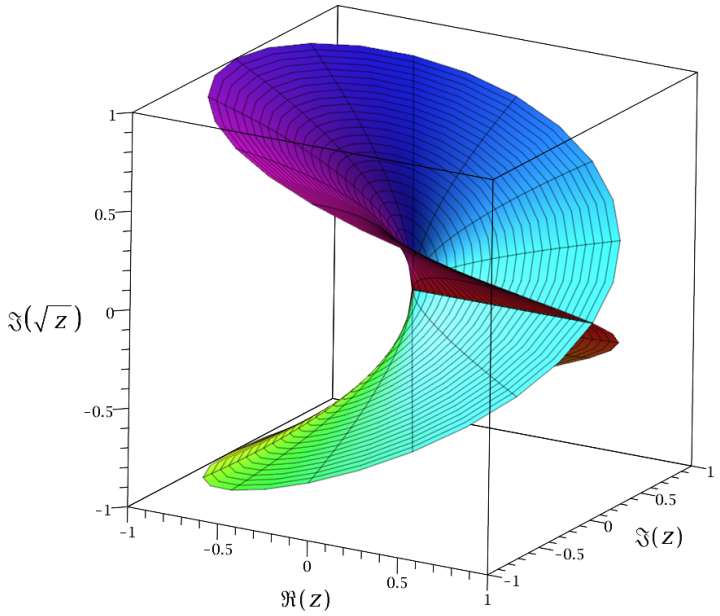


Figure 2.8: The Riemann surface of the imaginary part of the square root function.

can be expressed by other special functions. However, there is no general formal definition, but there are lists and mathematical compendia collecting functions which are commonly accepted as special.

For example, in 1964 Milton Abramowitz and Irene A. Stegun published the *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables* [2] at NIST[6]. The handbook became the most comprehensive source of information on special functions. Therefore, the NIST has published a new version called *NIST Handbook of Mathematical Function* [46] in 2010 to replace the predecessor.

Because there is no formal definition for special functions, some problems appear for our translator project. For example the *Encyclopædia Britannica* wrote:

> *Special function, any of a class of mathematical functions that arise in the solution of various classical problems of physics. [...] Different scientists might not completely agree on which functions are to be included among the special functions, although there would certainly be very substantial overlap.*
>
> *Encyclopædia Britannica - Special function [30]*

[6] In 1988 the *National Bureau of Standards* (NBS) was renamed to the *National Instutute of Standards and Technology* (NIST).

In *Special Functions* [4], G. E. Andrews et al. wrote in the preface:

> *Paul Turán once remarked that special functions would be more appropriately labeled "useful functions".*
>
> *Special Functions - Preface [4]*

Our reference for special functions in this thesis will be the *NIST Handbook of Mathematical Function* and its digital counterpart the DLMF [21]. See the later section 2.3 for a more detailed explanation.

Since most of the special functions represent solutions for other mathematical expressions, they were intensively studied - mostly in applied mathematics. In the past, huge tables of values were used for calculations [11]. Therefore, the book by Abramowitz and Stegun also contains tables with values for several special functions [2]. Obviously, computations of special functions with CAS were desired. Therefore, CAS support numerous of special functions. However, all CAS contain its own ever growing lists of them. Sometimes they use the same definitions, domains and position of branch cuts as defined the handbooks, sometimes not. A translator needs to pay attention to those differences as well. Otherwise the translation process could produce unexpected effects. One of these effects can be produced by different positions of branch cuts in the systems. The next section will give some examples and explanations of these problems.

2.2.3 Problems of Branch Cuts

Obviously, different positions of branch cuts can cause problems, if we transfer a function from one system to the other. Those differences can even appear in well-known elementary functions. For example, the DLMF defines the branch cut for the inverse cotangent function [21, (4.23.9)] at $[-i, i]$, while Maple reasonably [17] defines the branch cut at $(-\infty i, -i]$ and $[i, \infty i)$. Figure 2.9 illustrates the problem. A scientist, who is familiar with the definition for the branch cut by the DLMF would assume an output such as in figure 2.9a, but get an output such as in figure 2.9b in Maple.

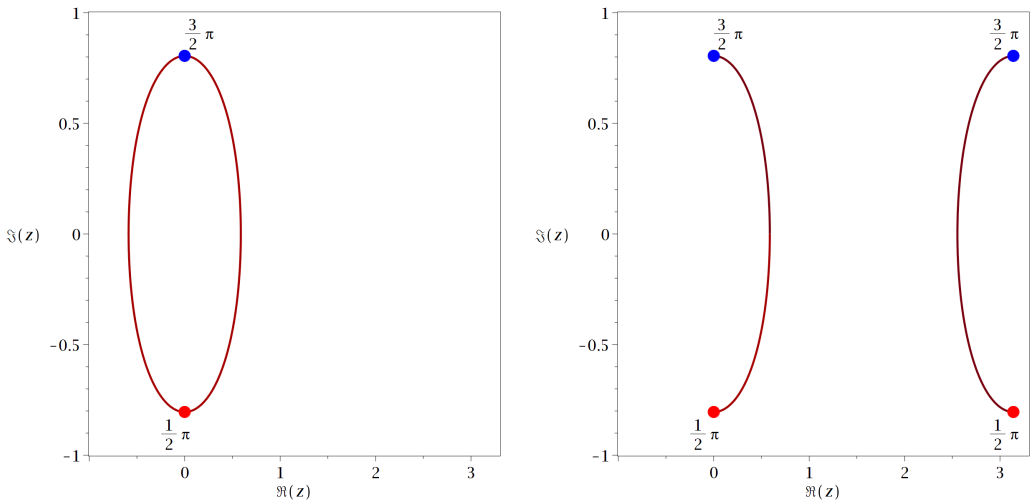
We provide the following solution for such problems. Instead of translating the function itself, the translator uses other equivalent representations for the function. In case of the arccotangent function, we can use the following alternative representations, based on the suggestions in [17].

Note that the following notation uses our standard notation for a translation from semantic LaTeX to Maple. We will introduce this notation in a formal way later in section 2.7.

$$\texttt{\textbackslash acot@\{z\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{arccot(z)} \tag{2.6}$$

$$\overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{arctan(1/z)} \tag{2.7}$$

$$\overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{I/2*ln((z-I)/(z+I))} \tag{2.8}$$



(a) Plot of the arccotangent function with the branch cut defined by DLMF.

(b) Plot of the arccotangent function with the branch cut defined by Maple.

Figure 2.9: Plots for the arccotangent function arccot(z) with $z = \frac{3}{2}e^{i\phi}$, where $\phi \in [0, 2\pi]$. Figure (a) illustrates a branch cut defined at $[-i, i]$ and figure (b) illustrates the branch cuts defined at $(-\infty i, -i]$ and $[i, \infty i)$. The blue and red dots represents the positions, where the function jumps over the branch cuts in (b).

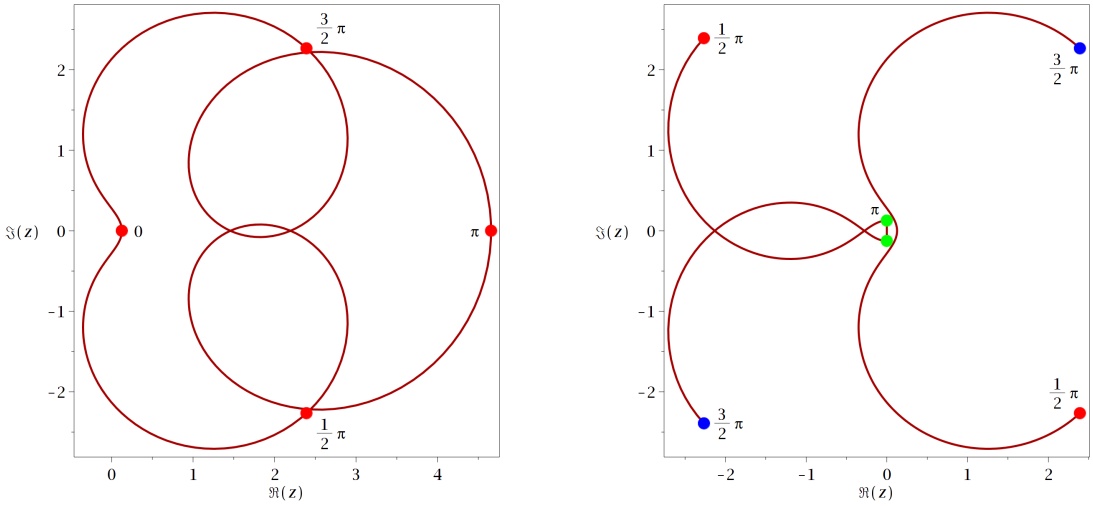
As illustrated above, translation (2.6) contains the branch cut issue. Translation (2.7) is an alternative, because the arctangent function in Maple has the same position for the branch cut of the arctangent function in the DLMF. However, in this case, the arccotangent function would be no longer defined at $z = 0$. Translation (2.8) is another alternative and defined at $z = 0$.

A more complicated example might be the equation of the already introduced parabolic cylinder function $U(a, z)$ and the modified Bessel function of the second kind $K_\nu(z)$. The parabolic cylinder function is a solution of a second order differential equation [21, (12.2i)] and can be represented by $K_\nu(z)$ for $a = 0$. The relation is defined in (2.9).

$$U(0, z) = \sqrt{\frac{z}{2\pi}}\, K_{\frac{1}{4}}\left(\frac{1}{4}z^2\right) \tag{2.9}$$

Following the same approach as for the arccotangent function discovers results in a branch cut problem. Figure 2.10 illustrates that problem for equation (2.9) with $z = 2.5e^{i\phi}$ and letting ϕ increase from 0 to 2π. While the left hand side is an entire function of z (can be seen in figure 2.10a), the right hand side has two functions with a branch cut along the negative real axis each. As already discussed, the branch cut of the square root function is at $(-\infty, 0]$. Therefore, at $\phi = \pi$ the right hand side jumps over the branch cut (green dots) back to the principal branch rather than continue on the other branch. Furthermore, the principal branch of K_ν is defined for $\phi \in (-\pi, \pi]$. With $z^2 = r^2 e^{2i\phi}$, the branch cut of K_ν is reached at $\phi = \left\{\frac{\pi}{2}, \frac{3\pi}{2}\right\}$ (red dots and

blue dots). The branch cut of K_ν "moves" to $(-\infty i, 0]$ and $[0, \infty i)$. Hence, we end up with three branch cuts for the right hand side of (2.9).



(a) The left hand side of (2.9).

(b) The right hand side of (2.9).

Figure 2.10: Two polar plots with $z = 2.5e^{i\phi}$ for $\phi \in [0, 2\pi]$ of left and right hand side of (2.9) in Maple. U is an entire function, while K_ν and the square root function have a branch cut along $(-\infty, 0]$. The colored dots represent the jumps over the branch cuts.

Since CAS do not provide Riemann surfaces, we use another approach to solve this problem, called **analytic continuation**. Analytic continuation is the process of extending the range of validity of a representation or more generally extending the region of definition of an analytic function [1, p. 152]. For the modified Bessel function of the second kind the analytic continuation is defined in the DLMF [21, (10.34.4)]. With analytic continuation, the right hand side of equation (2.9) is illustrated in figure 2.11.

2.2.4 Injective, Surjective & Bijective Mappings

Later in the thesis, we will define our translation as a function. Since our main goal is to provide bijectivity for our translation, we will briefly introduce the terminology.

Consider a function $f : X \mapsto Y$.

Definition 2.2: *(Injective Function)*
*The function f is **injective** if*

$$\forall x, x' \in X : \quad f(x) = f(x') \Rightarrow x = x'. \tag{2.10}$$

Definition 2.3: *(Surjective Function)*
*The function f is **surjective** if*

$$\forall y \in Y, \exists x \in X : \quad y = f(x). \tag{2.11}$$

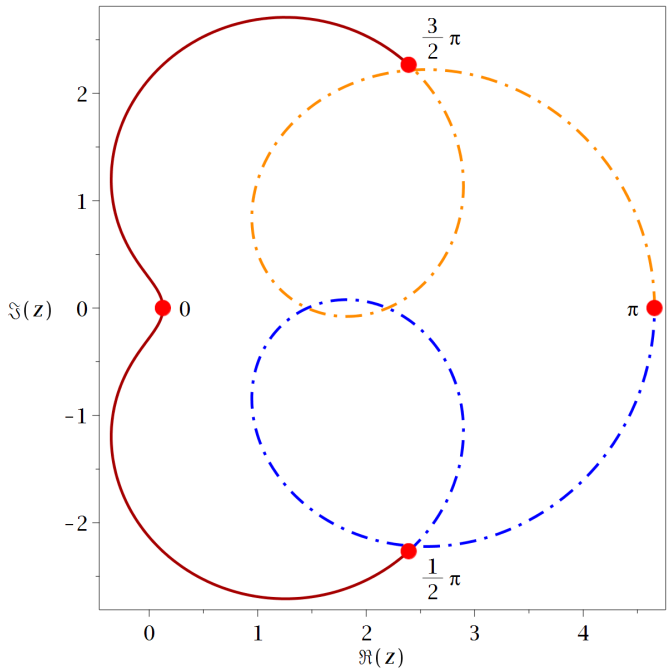


Figure 2.11: The right hand side of (2.9) using analytic continuation, when the function jumps over the branch cut at $\phi = \left\{\frac{\pi}{2}, \pi\right\}$. At $\phi = \frac{3\pi}{2}$, the function jumps back to the principal branch.

Definition 2.4: *(Bijection)*
*The function f is **bijective** if f is injective and surjective.*

The most useful part about bijective mappings is that every bijective function f has an inverse function f^{-1}. Consider a translation between two systems as a function that is bijective. Such a translation has an inverse function, or in other words backward translation. This would allow us to translate each expression in one system to the other and back again.

Obviously, such a translation is desirable. However, we will see that this goal is not achievable for all possible expressions. Even worse, we will see that our translation is neither injective nor surjective.

2.2.5 Graphs and Trees

A typical way to formalize the structure of mathematical expressions are trees. We will briefly introduce what trees are and what an expression tree is.

Definition 2.5: *(Graph)*
*A **graph** is an ordered pair $G = (V, E)$, where V is the set of **vertices** (or **nodes**) and E the set of **edges**, which are a 2-element subset of V.*

Definition 2.6: *(Directed Graph)*

*A **directed graph** (sometimes also referred as **digraph**) $G = (V, E)$ is a graph with oriented edges. That means E is a set of ordered pairs of vertices. Therefore, a graph is **undirected** if all edges $e \in E$ have no orientation.*

Definition 2.7: *(Paths, Cycles and connected Graphs)*
*Let $G = (V, E)$ be a directed or undirected graph with at least three vertices ($|V| \geq 3$). A **path** from one vertex $i \in V$ to another $j \in V$ is a finite sequence of vertices and edges*

$$i = i_1, (i_1, i_2), i_2, \ldots, i_{k-1}, (i_{k-1}, i_k), i_k = j. \tag{2.12}$$

*A path is a **cycle** if $i = j$. If a graph $G = (V, E)$ has no cycles, it is called **acyclic**. Furthermore, if there is a path for every pair of vertices, G is called **connected**.*

A Directed Acyclic Graph (DAG), for example, is used internally to represent mathematical expressions. However, another way to represent mathematical expressions are trees.

Definition 2.8: *(Tree)*
*A **tree** $T = (V, E)$ is an acyclic and connected graph.*

Let $T = (V, E)$ be a tree and $r \in V$. We call r **root** of T and order the other nodes in an ascending ordering according to its length of the path to r. Each node $v \in V$, which is directly connected with r is called *child* of r and r is called *parent* of v. This can be defined for all nodes in the tree which created a hierarchical structure in the tree. A node is in a higher hierarchy, when it is closer to root node. A node can have multiple *children*. Consider a node $a \in V$ as a node with three children u, v, w. We also define a order for the children, so that $u < v < w$. **Siblings** are nodes that share the same parent node. With the order we call u the previous sibling of v and w the following sibling of v. With this additional terminology, we can move through trees easily. Trees with an ordering for all nodes and a root node are also called **ordered trees**.

Mathematical expressions can be represented by ordered trees consisting of terminal symbols, such as identifiers or numbers (leaf nodes), and functions or operators (non-leaf nodes) [55]. Consider the mathematical expression

$$e^x + \frac{x^2}{\pi}. \tag{2.13}$$

The tool Visualizing Mathematical Expression Trees (VMEXT) [55] visualizes expression trees. Figure 2.12 draws the expression tree for (2.13).

2.3 Digital Library of Mathematical Functions

The NIST Digital Library of Mathematical Functions (DLMF) [38] is the digital version of the *NIST Handbook of Mathematical Function*. Internally, the DLMF websites are LaTeX files and all equations are numbered as usual. Therefore, we reference to a formula in the DLMF with the given equation number.

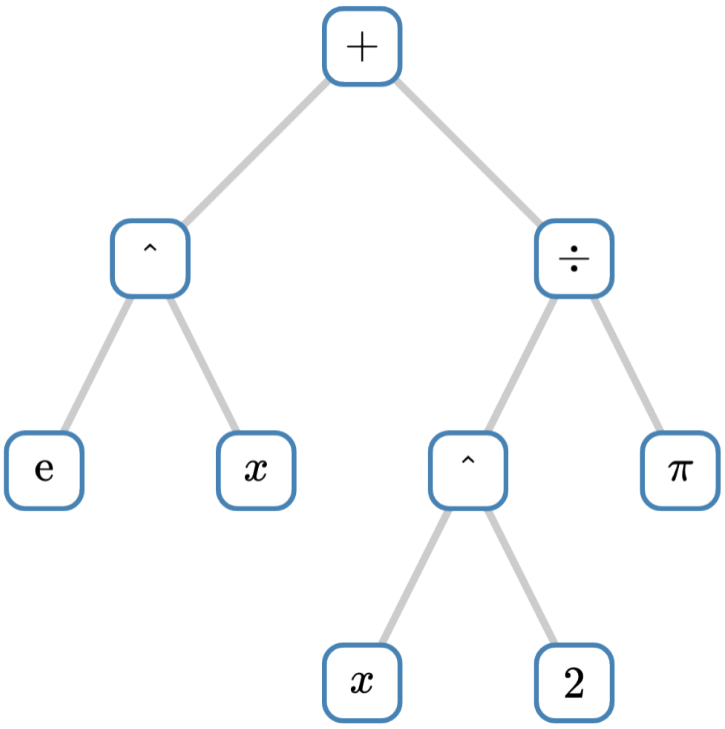


Figure 2.12: The expression tree of (2.13) visualized with VMEXT [55].

The NIST Digital Repository of Mathematical Formulae (DRMF) [14] is an outgrowth of the DLMF and wants to create a digital compendium of mathematical formulae for Orthogonal Polynomials and Special Functions (OPSF). One of the main purposes is to facilitate interaction among a community of mathematicians and scientists [15]. The DRMF also aims at expandability and interactivity. That was also a reason for this translation project, because a goal was to provide a one-click translation on the formulae in the DRMF.

Together, the DLMF and DRMF build a comprehensive compendium for OPSF and they are therefore our main references in this thesis. A definition of a function in any CAS is therefore called different, if it differs from the corresponding DLMF/DRMF definition.

2.4 Semantic & Generic LaTeX

Donald E. Knuth developed the typesetting system TeX in 1977 [31, p. 559], because he was unsatisfied with the typography of his book *The Art of Computer Programming*, Volume 2 [31, p. 5, 32]. Around 1980, Leslie Lamport starts to extend TeX by a set of macros, which became a new version of TeX. According to his name *Lamport*, this new version was called LaTeX [37]. Nowadays, LaTeX has become the de facto standard for scientists to write their publications [3].

This thesis, for example, is written in LaTeX 2_ε, the current version of LaTeX.

It is possible to customize and extend LaTeX, for example through style files, where an editor can define new LaTeX macros, environments and document styles.

2.4.1 Mathematical Generic LaTeX

LaTeX and TeX enables printing of mathematical formulae in a structure similar to handwritten style. For example, the LaTeX expression [21, (4.26.13)]

```
\int_{0}^{\infty} \sin(t^2) dt = \frac{1}{2} \sqrt{\frac{\pi}{2}}
```
(2.14)

will be rendered as

$$\int_0^\infty \sin(t^2)dt = \frac{1}{2}\sqrt{\frac{\pi}{2}}. \tag{2.15}$$

We call the syntax of such mathematical expressions in LaTeX *mathematical LaTeX* and hereafter use the terms *LaTeX expressions*, *mathematical LaTeX expressions* and *mathematical LaTeX* interchangeably.

Since TeX and LaTeX are typesetting systems, there are several ways to change the rendering of such mathematical expressions. However, for a CAS the information about the exact rendering of a mathematical expression is not as important as the meaning of the expression, the semantics.

2.4.2 Semantic Information

Semantics in general is the study of meanings. Semantic information, for example, can be in words, whole sentences, mathematical expressions, and signs.

Consider the equation (2.15). The meaning behind this expression is an equation. The left-hand side uses an infinite integral over the trigonometric sine function, which can be defined with [21, (4.14.1)]

$$\sin(z) := \frac{e^{iz} - e^{-iz}}{2i}. \tag{2.16}$$

The right-hand side of (2.15) contains a Greek letter π, which is known to indicate the mathematical constant for the ratio of a circle's circumference to its diameter. However, π can also indicate the prime-counting function. A human reader might conclude that π cannot be the prime-counting function, because there is no variable given. In (2.15), π is much more likely the mathematical constant, also because the left-hand side contains the sine function and the mathematical constant π belonging to trigonometric functions.

As demonstrated, there are a lot of meanings behind (2.15). Some of them are clear, others can be questionable. For a CAS, the exact meaning of a given input must be unique, otherwise the

CAS cannot compute the input. Therefore, mathematical expressions in CAS contain accessible semantic information. This is realized to connect character sequences with unique internal procedures and classes, which represent specific mathematical objects.

Mathematical LaTeX usually only contains information about the rendering of the expression and not about the meaning of the expression. However, there is semantic information in mathematical LaTeX, but it's often hidden. This was shown in the example discussion of the meaning of equation (2.15). In the plain text of the Greek letter π (`\pi`) is no exact meaning. Or, to be more precisely, there are multiple possible meanings, but there is no way to find the correct meaning. However, together with the left-hand side we can conclude that π is the mathematical constant in that case. As demonstrated, there is semantic information in the expression, but a reader needs to analyze the whole expression and make conclusions to find it. Hence, for a computer it is difficult to understand mathematical LaTeX. To provide a translation from mathematical LaTeX to a CAS, the semantics must be clear and accessible.

We will introduce in the next subsection a semantic version of mathematical LaTeX. To distinguish this version with classical mathematical LaTeX, we call classical LaTeX *generic LaTeX*.

Obviously, there must be different levels of semantics in expressions. We say an expression in generic LaTeX such as

```
P_n^{(\alpha,\beta)}(\cos(a\Theta))
```
(2.17)

contains less semantic information (more semantics are absent) than an appropriate representation in Maple

```
JacobiP(n,alpha,beta,cos(a*Theta)).
```
(2.18)

This is because the meaning of (2.18) is unique and accessible, while the exact meaning of (2.17) must be concluded from the structure and context of the formula. Unfortunately, there is no formal definition for grading semantics or a **fully semantic** expression. However, we use this phrase, when there is sufficient semantic information accessible to provide an appropriate translation.

2.4.3 DLMF/DRMF LaTeX Macro Set

Bruce Miller at NIST has created a set of semantic LaTeX macros [43]. Each macro ties specific character sequences to a well-defined mathematical object and is linked with the corresponding definition in the DLMF or DRMF. Therefore, we call these semantic macros DLMF/DRMF LaTeX macros. These semantic macros are internally used in the DLMF, DRMF and also in this thesis. They are defined in multiple style files and allow optional parameters and multiple renderings. Table 2.1 show how a semantic macro is defined in a style file.

The `\defSpecFun` is a defined macro to easily define new macros for special functions. It takes a unique character sequence for the function name. The first optional argument `nparams` defines the number of parameters of the special function. Some special functions, for example the associated Legendre functions, can define optional parameters. The general structure of

`\defSpecFun{function}`	Define a macro `\function`.
`[nparams]`	Number of parameters of the macro.
`[optparams]`	The number of optional parameters of the macro.
`{format}`	Defines the format of the macro. (How it will be displayed).
`[keys]`	Attributes of the function. For example, *meaning* or *role*.
`{arity}`	Number of arguments. Defines how many {•} blocks come after the @ symbol.
`[argformat]`	Format of arguments.
`[altargformat]`	Alternative formats depends on the number of @ symbol.

Table 2.1: Overview for the definition of a semantic macro.

a semantic macro has two parts to display the formula. The first part defines the display for the function symbol (including parameters) in `{format}`. This format is fixed and cannot be changed. In contrast, the display for the arguments of the function can be controlled by the number of @ symbols and is defined in multiple optional `[argformat]` blocks.

Consider the *hypergeometric function* [21, (15.2.1)]. The semantic macro

```
\HypergeoF@{a}{b}{c}{z}
```
(2.19)

is linked to the definition (15.2.1) in the DLMF. The *hypergeometric function* provides three different representations. Table 2.2 gives an overview for all of them.

Semantic macro	Display
`\HypergeoF@{a}{b}{c}{z}`	$F(a,b;c;z)$
`\HypergeoF@@{a}{b}{c}{z}`	$F\left(\begin{smallmatrix}a,b\\c\end{smallmatrix};z\right)$
`\HypergeoF@@@{a}{b}{c}{z}`	$F(z)$

Table 2.2: The semantic macro for the *hypergeometric function* [21, (15.2.1)] has three different ways to display the function. The different representations are controlled by the number of @ symbols.

The display of the function name F is fixed, but the number of @ symbols control the display of the arguments. The third representation displays only the variable of the function, but hide the given parameters. Therefore, the semantic LaTeX source in the background provide more semantic information than the displayed version. However, all of them represent the *hypergeometric function* defined in the DLMF.

The sets of macros from the DLMF and the DRMF are mostly different. The DRMF uses the macros from the DLMF and developed new macros as well. However, it happens that there are

two different macros for the same mathematical object defined. One example is the *Gegenbauer polynomial*[7]

```
\Ultra{\lambda}{n}@{x},                    (2.20)
\Ultraspherical{\lambda}{n}@{x}.           (2.21)
```

There are currently 685 semantic macros defined. Note that the number of macros is constantly changing, because the project of semantic macros is still work in progress.

2.4.4 Weakness of DLMF/DRMF Macros

The set of DLMF/DRMF LaTeX macros gives the opportunity to semantically enhance mathematical expressions. However, there are mostly macros for OPSF. It is difficult to foresee if a semantic macro provides sufficient semantic information for an appropriate translation, or, as we formulated above, if a semantic macro is really fully semantic.

One example is the *Wronskian*. For example, for two differentiable functions the *Wronskian* is defined as [21, (1.13.4)]

$$\mathscr{W}\{w_1(z), w_2(z)\} = w_1(z)w_2'(z) - w_2(z)w_1'(z). \tag{2.22}$$

In semantic LaTeX it is used with

```
\Wronskian@{w_1(z), w_2(z)}.               (2.23)
```

It turned out, that for an appropriate translation to a CAS, we need access to the variables of $w_1(z)$ and $w_2(z)$. Therefore, we improved the macro and replaced it to a new version

```
\Wron{z}@{w_1(z)}{w_2(z)}.                 (2.24)
```

This new macro is displayed in the same way as the old version was displayed, but it provides more information.

Another version of semantic LaTeX is sTeX, developed by Michael Kohlhase [34]. This version of semantically enhanced LaTeX is more comprehensive for general mathematics. It is not particularly useful for OPSF, and it doesn't link macros to unique mathematical definitions. Therefore, we did not use sTeX in this project, although it provides more semantic information for simple arithmetic expressions. This part is almost completely missing in the DLMF/DRMF LaTeX macros in its current state. One example is the question of white spaces. Scientists frequently uses white spaces to indicate multiplications. Most CAS are familiar with this convention and are able to understand such inputs. However, in Maple's 1-D input notation, an asterisk is mandatory for multiplications. Therefore, we also added a macro `\idot` (for *invisible dot*), to provide a multiplication symbol which is not displayed in LaTeX, but semantically enhances the LaTeX expressions.

[7] Also known as the *ultraspherical polynomial*.

2.5 Computer Algebra Systems

A Computer Algebra System (CAS) is a mathematical software, which is, for example, able to analyze, simplify and compute mathematical expressions. It is based on the theory of computer algebra. Computer Algebra is a part of computer science that designs, analyzes, implements and applies algebraic algorithms [9]. In 1963, *Schoonschip* was one of the first CAS developed by Martinus J. G. Veltman [58]. It was mainly used for high-energy physics.

Using CAS was highly desirable, because it was a faster and easier alternative to calculation tables [11]. Another CAS *MATLAB*[8] was initially developed by Cleve Moler in the late 1970s to allow students to use Fortran[9] libraries without learning Fortran itself [44]. *MATLAB* was constantly updated[10] and is still widely used for education and in image processing, because of its performance in linear algebra and numerical analysis.

Nowadays, there are several different CAS available. From proprietary solutions, such as *MATLAB*, Maple, Mathematica, to freely available software packages, such as *SageMath*. Most CAS are more or less specialized for fields in mathematics, such as *MATLAB* for linear algebra and numerical computations. However, nowadays the most popular CAS are Mathematica and Maple.

2.5.1 Maple

Maple was initially released in 1982 and is written in *C*, *Java* and its internal programming language, also called *Maple*. The main purposes was to create an CAS, which would be able to run on lower cost computers. Maple bases on a small kernel, which is written in *C*, that provides the Maple language. Additional libraries are written in the Maple language, while the Graphical User Interface (GUI) is written in *Java*.

In this thesis, we refer to version *Maple 2016* and all information about internal data structures and the usage of Maple is described in the manual [8].

Internally, symbolic expressions are stored in the memory as DAG data structures. This avoids multiple blocks in the memory for the same object in an expression.

Furthermore, Maple allows different input notations: the `1D` input and the `2D` input. Consider the infinite integral

$$\int_0^\infty \frac{\pi + \sin(2x)}{x^2} dx. \tag{2.25}$$

[8] *MATLAB* is an abbreviation for ***mat****rix* ***lab****oratory*.

[9] Originally *FORTRAN*, was the first widely used high-level programming language for general purposes and is still very popular for high-performance computing.

[10] Newest version is *MATLAB 9.2* from march 2017.

The corresponding 1D input notation in Maple is

```
int((Pi+sin(2*x))/x^2, x=0..infinity).
```
(2.26)

Historically, the 1D format was used for inputting commands and expressions into Maple, while the 2D input is more used for interaction with human users. In fact, the internal 2D representation is similar to MathML [5] and its display is similar to (2.25).

We consider Maple's 1D input as syntactically correct and our translation process always refers to the 1D input representation rather than the 2D input representation.

Another interesting characteristic, which becomes import for our translator, is determinism.

Definition 2.9: *(Deterministic Algorithm)*
An algorithm is called ***deterministic*** *if it produces always the same results on two identical inputs.*

Maple is in general non-deterministic. That means, that two identical inputs possibly produce two different outputs depending on the run time environment. The reason is, that for example the order of the elements of a set[11] depends on the addresses of the elements in the memory. Mostly the background operation systems determine the positions of variables in the memory rather than the application Maple does.

However, Maple works in sessions. When a user starts a worksheet a new Maple-session will be created. It is also possible to start a new session in a worksheet with the command `restart`. Within such sessions Maple is deterministic as long as the input doesn't use non-deterministic algorithms, such as probabilistic algorithms[12].

2.5.2 Mathematica

Wolfram Mathematica was conceived by Stephen Wolfram and initially released in 1988 [61]. It is also written in *C/C++*, *Java* and its internal programming language *Wolfram Language*. It is one of the most comprehensive CAS nowadays and an ever growing software. The previously mentioned interactive document CDF uses Mathematica. Also the knowledge engine[13] *Wolfram|Alpha*[14] is based on Mathematica.

In 2008 an error [23] in Mathematica shows a problem of confidence in CAS. As we will explain later, our translator uses symbolic simplification[9, section 2] techniques in Maple. Implemented

[11] Mathematically, the elements in a set are not ordered. This is technically different in computer algebra, because each element is somewhere located in the memory. In a scope of a low-level programming environment, the elements of sets always have an order.

[12] Probabilistic algorithms use randomness to find approximate solutions. Such algorithms are obviously non-deterministic.

[13] Also known as answer engine, is a computer science discipline which tries to automatically answer questions posed by humans in natural language.

[14] https://www.wolframalpha.com/, seen 08/2017

symbolic simplification algorithms are rarely[15] proven and may therefore produce errors. We can only presume that the simplifications are correct.

2.6 Mathematical Language Processor

The Mathematical Language Parser (MLP)[16] project was developed to parse any mathematical expressions [63].

According to Natural Language Processing (NLP) [40], the MLP understood mathematical expressions as a word of a formal language and parses expressions to so called parse trees. To understand the MLP, we need to understand the concept of formal languages first. In the following, we will briefly introduce formal languages [25].

Definition 2.10: *(Alphabet)*
*An **Alphabet** Σ is a finite, non-empty set. The elements of Σ are called **letters** (or **symbols**).*

For example, the alphabet of a computer is simply the binary alphabet $\Sigma = \{1, 0\}$.

Definition 2.11: *(Word)*
*A **word** (or string) w over an alphabet Σ is a finite sequence of letters of Σ. The set of all words over Σ is denoted as Σ^* (known as the Kleene star of Σ). The empty word ϵ (sequence of length 0) is always an element of Σ^*.*

Definition 2.12: *(Formal Language)*
*A subset $L \subseteq \Sigma^*$ is called **formal language** over an alphabet Σ.*

Hereafter we use the term *formal language* and *language* interchangeably. Usually, a formal language is constructed by a set of rules. In our case we construct a formal language by a context-free grammar [29, p. 79, 39].

Definition 2.13: *(Context-Free Grammar)*
A context-free grammar $G = (V, \Sigma, R, S)$ is a 4-tuple, where

1. *V is a finite set. The elements of V are called **non-terminal symbols**.*
2. *Σ is a finite set with $\Sigma \cap V = \varnothing$. The elements of Σ are called **terminal symbols**.*
3. *R is a finite relation from V to $(\Sigma \cup V)^*$, thus $R \subseteq V \times (\Sigma \cup V)^*$. The members of R are called **rules**.*
4. *$S \in V$ is called the **start symbol**.*

[15] In fact, there is no proof of correctness for symbolic simplifications in any CAS known.

[16] Note that the official publication refers MLP to *Mathematical Language **Processor***, rather than to the **parser**. However, the *Mathematical Language Processor* project [47] is slightly different and trying to analyze the natural language context of mathematical expressions. However, both projects are highly coupled and uses the same abbreviation. Since we focus on the parser, we use MLP to refer to the *Mathematical Language Parser*, even the official publication understood it as a **processor**.

A rule $(\alpha, \beta) \in R$ can be understood as a substitution. We denote the rule also as $\alpha \longrightarrow \beta$.

Definition 2.14: *(Derived Words & Applied Rules)*
*Let $G = (V, \Sigma, R, S)$ be a context-free grammar. Let $A \in V$ and $u, v, w \in (\Sigma \cup V)^*$, such that $(A, w) \in R$. Than we say a word 'uwv' can be **derived** in one step from the word 'uAv' by **applying** the rule (A, w).*

$$uAv \Rightarrow uwv. \tag{2.27}$$

We can also apply multiple rules sequentially.

Definition 2.15: *(Chain Derivation)*
Let $G = (V, \Sigma, R, S)$ be a context-free grammar and $u, v \in (\Sigma \cup V)^*$, with $u \neq v$. We say v can be derived from u, if

$$\exists\, k \geq 2 \;\; \exists u_1, u_2, \ldots, u_k \in (\Sigma \cup V)^* : \quad u = u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k = v, \tag{2.28}$$

and write

$$u \overset{*}{\Rightarrow} v. \tag{2.29}$$

With a context-free grammar, we can define a context-free language over the grammar.

Definition 2.16: *(Context-Free Language)*
Let $G = (V, \Sigma, R, S)$ be a context-free grammar. G defines a language $L(G)$ by

$$L(G) = \left\{ w \in \Sigma^* : \; S \overset{*}{\Rightarrow} w \right\}. \tag{2.30}$$

A language L is called **context-free language** if there is a grammar G, such that $L = L(G)$. In other words, a context-free language $L(G)$ is a set of all strings that can be derived from the start variable S of G.

Example 2.1:
A simple example for a context-free grammar are well-formed parentheses. Let $V = \{S\}$, $\Sigma = \{(,)\}$ and S be the start symbol. Furthermore, let R contain the rules

$$S \longrightarrow SS \tag{2.31}$$

$$S \longrightarrow (S) \tag{2.32}$$

$$S \longrightarrow () \tag{2.33}$$

Let $G = (V, \Sigma, R, S)$. Than, for example

$$'(()(()))' \in L(G). \tag{2.34}$$

Proof of 2.34. We can produce the following chain of rules from R.

$$\begin{aligned} S &\Rightarrow (S) \\ &\Rightarrow (SS) \\ &\Rightarrow (()S) \\ &\Rightarrow (()(S)) \\ &\Rightarrow (()(())). \end{aligned} \tag{2.35}$$

Therefore

$$S \overset{*}{\Rightarrow} \text{'(()(()))'} \tag{2.36}$$

and '(()(()))' $\in L(G)$. □

On the other hand

$$\text{'(()'} \notin L(G). \tag{2.37}$$

The chain of rules can also be represented in a tree structure. Such trees are called **parse trees**. The tree for (2.35) is visualized in figure 2.13. Note that a parse tree is different to an expression tree. However, a parse tree can become an expression tree if the appropriated mathematical rules are defined.

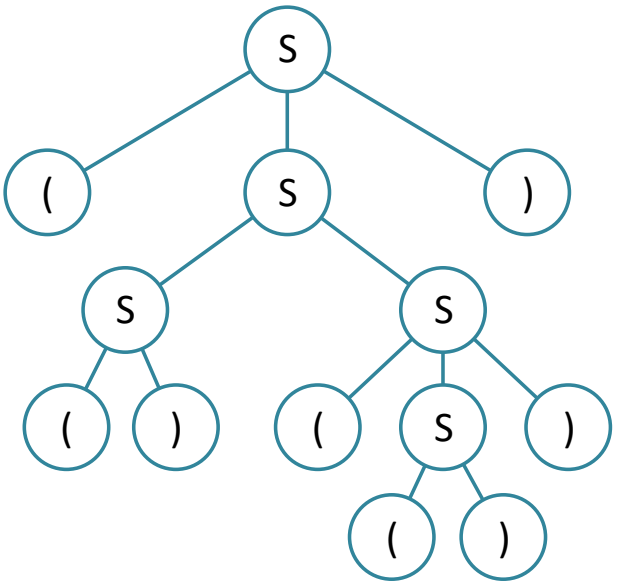


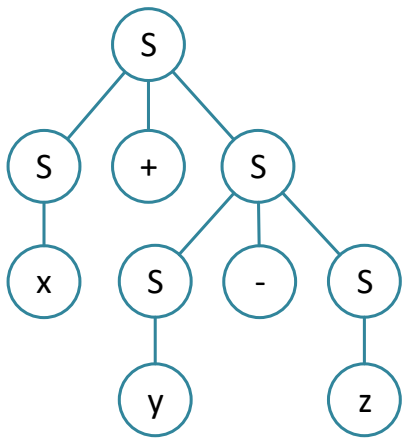
Figure 2.13: The parse tree of the expression '(()(()))' for the well-formed parenthesis grammar $G = (V, \Sigma, R, S)$. Produced by the chain of rules 2.35.

There are several more definitions around formal languages, which we will not mention here. Note, for example, that a parse tree such as in figure 2.13 is not necessarily unique and can depend on the choice of a rule where more than one is *applicable*. Consider a context-free grammar for simple arithmetic expressions with sums and subtractions of three variables x, y

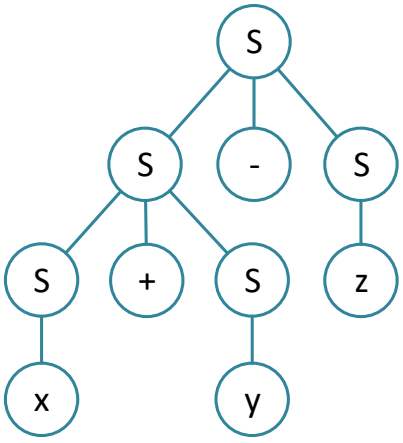
and z. Let $V = \{S\}$, $\Sigma = \{x, y, z, +, -\}$, S be the start symbol and define the rules

$$S \longrightarrow x \tag{2.38}$$
$$S \longrightarrow y \tag{2.39}$$
$$S \longrightarrow z \tag{2.40}$$
$$S \longrightarrow S + S \tag{2.41}$$
$$S \longrightarrow S - S. \tag{2.42}$$

Now an expression such as $x+y-z$ is an element of the context-free language of $G = (V, \Sigma, R, S)$. However, there are two possible parse trees for this expression, tree 2.14a and 2.14b, depending on the order of which rule gets applied first. Such grammars are also called ambiguous.



(a) The parse tree of $x + y - z$ if rule (2.41) gets applied first.



(b) The parse tree of $x + y - z$ if rule (2.42) gets applied first.

Figure 2.14: Two different parse trees for $x + y - z$. (a) applies rule (2.41) before (2.42), while (b) applies the rules vice versa.

A parser can define now in which order the rules get applied. For example, an **LL**-Parser parses the input from **L**eft to right and performing the **L**eftmost derivations. Leftmost in this terminology means, the parser takes the first applicable rule. In consequence, the internal ordering of rules is no longer irrelevant. The MLP is such an **LL**-Parser. In case of the expression $x + y - z$ from above with the rules (2.38 - 2.42), an **LL**-Parser would generates tree 2.14a.

2.6.1 BNF Grammar

The Backus-Naur Form (BNF) is a notation technique for context-free grammars in computers. For example, a BNF specification is a set of rules, written as

```
<symbol> ::= exp,
```
(2.43)

where `<symbol>` is a non-terminal element and `exp` are one or more sequences of symbols. Symbols that never appear on the left side are terminals.

The BNF is often used to describe the syntax of programming languages. There are several tools that automatically generates a parser from a given grammar in BNF. One of those tools is *JavaCC* (for Java Compiler Compiler).

2.6.2 POM-Tagger & MLP

The MLP defines a BNF grammar to parse mathematical LaTeX expressions. The Part-of-Math (POM)-tagger (according to Part-of-Speech tagger in NLP) categorizes math tokens. The main purpose of the POM-tagger is the extraction of semantic information from mathematical LaTeX. The POM tagger is designed to perform multiple *scans*. Given an input LaTeX math document, the first scan of the tagger examines *terms*, and groups them into sub-expressions where suitable. For instance `\frac{1}{2}` consists of two sub-expression of numerator and denominator. A *term* is a non-terminal symbol defined in the BNF. In the definition of a LaTeX grammar this can be LaTeX macros, environments, reserved symbols (such as the LaTeX line break command \\), or numerical or alphanumerical expressions. The first scan of the POM-tagger adds information to each term. The information is defined in lexicon files, which provide meanings, descriptions, categories and many more for a large variety of symbols. For example, table 2.3 gives an example of the lexicon entry for the plus symbol.

Symbol: +
Feature Set: universal
Category: operation
Description: plus
Source: unicode-math
TeX-equivalent: `\mathplus`
Unicode: u+0002b

Table 2.3: The lexicon entry for the + symbol.

A symbol can be tagged with multiple feature sets according to its ambiguous semantics. Tagged terms are called math terms. Math terms are rarely distinct at this stage and often have multiple features. All information are centralized in the `global-lexicon.txt`.

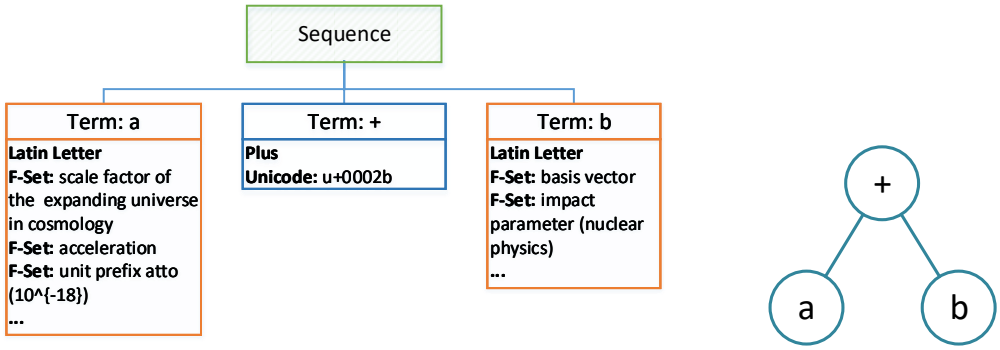
Further scans will try to reduce the number of alternative feature sets by scanning the context of the math expression and concluding possible feature sets from the structure of the expression. See section 6.1 for more information about this approach.

In its current state, the POM-tagger only performs the first scan. Note that, technically, the MLP parses mathematical expressions while the POM-tagger categorizes elements of the parse tree. However, the MLP also acts as an interface to interact with the parse tree and the POM-tagger is a kind of an intermediate step of the whole parsing process. Therefore, we will refer to the MLP in following chapters rather than to the POM-tagger.

2.6.3 MLP-Parse Tree

Hereafter we consider the MLP-Parse Tree (PT) as the parse tree, which is produced by the MLP. The BNF of the MLP has no mathematical rules implemented in its current state. For example, there is no rule, similar to (2.41) for the binary operator +. Therefore, the PT is not an expression tree. The POM project was designed to extract semantic information from mathematical expressions. Certainly, the semantics of mathematical symbols is rarely unique. Consider the − symbol as an example. As a binary operator it indicates the subtraction of two arguments. However, it is also a unary operator as a leading sign.

In its current stage, it is futile to implement arithmetical rules into the MLP grammar. The next scan of the MLP will reduce ambiguous meanings from the PT. It can be anticipated that the PT will become an expression tree when the planned next scans are implemented.



(a) The MLP-PT for $a + b$. The POM-Tagger adds information to all terminals. F-Set are feature sets. The terms a and b have multiple feature sets. An F-set represents another semantic meaning. In the figure are not all F-sets listed.

(b) The expression tree for $a + b$.

Figure 2.15: Comparing the MLP-Parse tree with an expression tree for $a + b$.

2.7 Translator Definitions

To explain translations mathematically, we need to define some techniques first. Mainly, a translation process is a function that maps an input expression to an output expression. The input and output expressions are written in two different formal languages. For a translation, the exact types of input and output language are insignificant. Therefore, we consider mathematical LaTeX, semantic LaTeX, Maple's `1D` input and Mathematica input as formal languages.

Definition 2.17: *(Translation)*
Let $\mathcal{L}$ be a set of formal languages and $L_1, L_2 \in \mathcal{L}$ be two formal languages. A ***translation*** tr *is*

a mapping from an input language L_1 to an output language L_2, denoted as

$$\mathrm{tr}_{L_2}^{L_1} : L_1 \to L_2.$$

Definition 2.18: *(Forward & Backward Translation)*
*Let $\mathcal{L}$ be a set of formal languages. Let $L_1 \in \mathcal{L}$ be the language of semantic LaTeX and $L_2 \in \mathcal{L}$ the language of any CAS. The function $\mathrm{tr}_{L_2}^{L_1}$ is called **forward translation to** L_2 and $\mathrm{tr}_{L_1}^{L_2}$ is called **backward translation from** L_2 respectively.*

Therefore, we define $\mathfrak{sL}$ as the formal language of semantic LaTeX, $\mathfrak{M}_{aple}$ as the formal language of Maple and $\mathfrak{M}_{athematica}$ as the formal language of Mathematica. Since we will only translate expressions between $\mathfrak{sL}$ and a CAS, we will use a more convenient notation for forward and backward translations. Let $t \in \mathfrak{sL}$ be an arbitrary expression in semantic LaTeX. A forward translation to Maple is denoted as

$$t \overset{\mathfrak{M}_{aple}}{\mapsto} c, \quad \text{with } c \in \mathfrak{M}_{aple}, \tag{2.44}$$

and a backward translation from Maple as

$$t \overset{\mathfrak{M}_{aple}}{\mapsfrom} c, \quad \text{with } c \in \mathfrak{M}_{aple}. \tag{2.45}$$

Note that there is an identity translation between two formal languages L_1, L_2 defined if there is a word $w \in L_1 \cap L_2$. We specially denote the identity translation with

$$\begin{aligned} &\mathrm{id}_{L_2}^{L_1} : \ L_1 \to L_2, \\ &\mathrm{id}_{L_2}^{L_1}(w) := w, \quad \text{with } w \in L_1 \cap L_2. \end{aligned} \tag{2.46}$$

In this thesis, we will translate expressions from one system to another. There is no formal definition to compare mathematical expressions in two different systems. As an example, consider the trigonometric cosine function $\cos(z)$. There is a formal definition in the DLMF given [21, (4.14.2)]

$$\cos(z) = \frac{e^{iz} + e^{-iz}}{2}. \tag{2.47}$$

The internal definition of the cosine function in Maple is given by

$$\cos(z) = \frac{1}{2}e^{iz} + \frac{1}{2e^{iz}}. \tag{2.48}$$

Technically, we would agree that both definitions are equivalent. However, Maple is limited by the possibilities of a computer system. For example, the mathematical constant e is irrational and can therefore never be *absolutely equivalent* to the abstract mathematical construct of Euler's number. Instead of defining a new equivalence term for our purpose, we conveniently look for an **appropriate** translations between two formal languages. Obviously, this is not a formal

definition, but should prevent us from translating one mathematical object[17] into a completely different object in another language.

Therefore, we would label a translation such as

$$\texttt{\textbackslash cos@\{z\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{cos(z)} \tag{2.49}$$

as *appropriate*, while a translation such as

$$\texttt{\textbackslash cos@\{z\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{sin(z)} \tag{2.50}$$

would be *inappropriate*. However, note that it is not always as easy as in this example to decide if a translation is appropriate or not. Therefore, we developed validation techniques to automatically check if a translation can be considered appropriate. In addition to this terminology, we introduce *direct translations*. Later in the thesis, we will explain that a translation from one specific mathematical object to its *appropriate* counterpart in the other system is not possible. We call a translation to the *appropriate* counterpart **direct**. For example, the translation (2.49) is *direct*, while a translation to the definition of the cosine function

$$\texttt{\textbackslash cos@\{z\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{(exp(I*z)+exp(-I*z))/2} \tag{2.51}$$

is not a *direct* translation.

[17] We are aware that it could be even more difficult to formally define what a *mathematical object* is. We consider a mathematical object to be a function or symbol with a formal mathematical definition behind it.

Chapter 3

Semantic LaTeX Translator

The overall goal of the whole project was to provide an interactive one click-translation for each formula in the DRMF to a specified CAS. The program mainly supports translations from and to the CAS Maple. However, the program is designed to support other CAS. In the current stage it also supports translations to Mathematica, but the backend library needs to get extended for a full support.

In this chapter, the functionality and the design of the translator will be explained. The structure follows the development process and explains our decisions, problems and solutions. It starts with an overview of our approaches in 3.1 and solutions for the problems. We will also focus on the theory of our backend libraries in 3.1.2 and how to extend them in 3.1.3. The sections 3.2 and 3.3 explains the forward and backward translator separately.

3.1 General Approach and Goals

Initially this project aimed for translations not only for semantic LaTeX expressions but for generic LaTeX as well. However, as discussed before, the semantic information in generic LaTeX is hardly accessible (as seen in 2.4.2). Supporting generic LaTeX translations would go beyond the scope of a Master's thesis. Hence, we focused on translations for semantic LaTeX expressions. Furthermore, the translator should support multiple CAS and because of the work in progress state of the DLMF/DRMF LaTeX macro it should be easily extendable and changeable. Due to the time limitations of the project we focused our implementations on the support for Maple and Mathematica.

Another strive for a goal was to keep a translation as straight forward as possible. In other words, the structure of the translated expression should be similar to the structure of the input expression. This makes it easier for a user to follow the translation process and debug the program. This intention should not to be confused with the task for *appropriate* translations. We will see that a translation process (especially the backward translation) can easily change

expressions without changing the semantics. We want to avoid that behavior in our translations as often as possible.

Furthermore, the translation process should solve problems of existing technologies, presented in section 2.1. In the following subsection, we will focus on known problems and present solutions before we start to explain implementation details.

3.1.1 Translation Problems

Most of the existing translation tools cannot preserve the semantic information during the translation process (see section 2.1). Therefore, our program should not lose semantic information during the process. However, the semantics can be changed during a translation. This happens when a definition or property of a mathematical object varies from system to system. Our reference systems for all mathematical LaTeX expressions are the DLMF and the DRMF.

Let us focus on translations of mathematical objects that are not defined in one of the languages, first. Obviously, we cannot translate these objects *directly*. Our workaround is to translate not the mathematical object itself, but the definition of the object. For example, the *Gudermannian* [21, (4.23.10)] $\mathrm{gd}(x)$ function can be defined by

$$\mathrm{gd}(x) := \arctan(\sinh(x)) \quad x \in \mathbb{R}. \tag{3.1}$$

This function is not implemented in Maple. Therefore, we translate its definition (3.1)

$$\texttt{\textbackslash Gudermannian@\{x\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{arctan(sinh(x))}. \tag{3.2}$$

We want to avoid such translations, because they are not intuitive and therefore can confuse a user. However, providing such translations is still better than providing no translation.

Instead of none possible translation, how can we handle multiple alternative translations? We already discussed the problem of the inverse trigonometric function $\mathrm{arccot}(x)$. Because of a different branch cut in Maple (compared to the DLMF definition), there are alternative translations available (see (2.6)). In such cases, we define the most intuitive, the direct translation. In addition, we inform the user about the available alternative translations and the branch cut issues. Hence,

$$\texttt{\textbackslash acot@\{x\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{arccot(x)}. \tag{3.3}$$

Since we have these alternative translations, we also use the same technique for the previous Gudermannian example. We provide one direct translation and possible alternative translations for a mathematical object.

These two problems already explained, why our translation process is in general not bijective. However, we will explain later in a more detailed way that the translation is only bijective for a subset of possible expressions. Thereby, a bigger goal is to increase this subset as much as possible.

3.1.2 Libraries

The first step for providing an automatic translation tool is to defining the translation for each function. We manually create a table of translations for each DLMF/DRMF LaTeX macro. Translations for generic LaTeX functions, mathematical symbols, Greek letters and constants are organized in JavaScript Object Notation (JSON) files for a better performance.

For each translation, we design a translation pattern with placeholders to define the correct position of each argument. The placeholders are defined as `$i`, where `i` is a non-negative integer specifying the position, starting at 0. Table 3.1 illustrates the translation patterns for the trigonometric sine function in semantic LaTeX, Maple and Mathematica.

Semantic LaTeX	`\sin@{$0}`
Maple	`sin($0)`
Mathematica	`Sin[$0]`

Table 3.1: Translation patterns for the sine function in the DLMF, Maple and Mathematica.

Placeholders gives us the opportunity to define the order of the arguments and handle nested function calls. The order of arguments can differ between the systems. For example, if a function possesses parameters written in sub- and superscripts, Maple and Mathematica usually put the parameters in the subscript before the parameters in the superscripts. The notation in the DLMF/DRMF LaTeX macros uses the reversed order. Table 1.1 in the introduction shows an example of such different orders. The translation patterns for the Jacobi polynomial are illustrated in table 3.2.

Forward Translation:	
Maple	`JacobiP($2, $0, $1, $3)`
Mathematica	`JacobiP[$2, $0, $1, $3]`
Backward Translation from Maple/Mathematica:	
Semantic LaTeX	`\JacobiP{$1}{$2}{$0}@{$3}`

Table 3.2: Forward and backward translation patterns of the Jacobi polynomial. The pattern for the backward translation is the same for Maple and Mathematica.

Note that Maple also uses the symbol `$` in few functions. This can cause trouble in the replacement process. However, we can handle all of the few cases by adding addition parenthesis. Consider the differentiation function in Maple

```
diff(f, [x$n]),
```
(3.4)

with f as an algebraic expression or an equation, x the name of the differentiation variable and n for the n-th order differentiation. Therefore

$$\frac{d^2x^2}{dx^2} = \texttt{\textbackslash deriv[2]\{x\^{}2\}\{x\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{diff(x\^{}2, [x\$2])}. \tag{3.5}$$

But the translation pattern in Maple would be

```
diff($1, [$2$$0]).
```
(3.6)

Replace the placeholders sequentially by the arguments ends in

		`diff($1, [$2$$0])`
Replace index 0 by 2	→	`diff($1, [$2$2])`
Replace index 1 by x^2	→	`diff(x^2, [$2$2])`
Replace index 2 by x	→	`diff(x^2, [xx]).`

We use parenthesis to solve this issue and define the translation pattern with

```
diff($1, [$2$($0)]).
```
(3.7)

A special problem can be binary operators. A good example is the multiplication sign. In LaTeX most scientist would use white spaces or `\cdot` to indicate a multiplication (using none of them can cause ambiguities, see 3.2.2). Similar to that approach, Mathematica also understands white spaces and asterisks as multiplications. When we translate semantic LaTeX expressions to Maple, we use the 1D Maple representation (see subsection 2.5.1), where an asterisk becomes mandatory for multiplications. White spaces would produce a syntactical error in this input format. On the other hand, white spaces in LaTeX do not necessarily indicate a multiplication. It could also be used to improve the readability. Whether a white space indicates a multiplication or not is a semantic information in the expression. We introduce a new DLMF macro that indicates a multiplication, but not gets rendered. This macro is `\idot`, for an invisible dot.

As already mentioned, the information is stored in three different file types. Together they form the backend library.

CSV Tables

The Comma-Separated Values (CSV) file represents the translation tables generated by MS Excel. Although CSV is an abbreviation for *comma-seperated*, one can specify the split symbol. Most common are semicolons. In our case we use semicolons, because commas are frequently used in string representations of mathematical functions to separate multiple parameters and variables. Note that semicolons are also sometimes used, but CSV files are able to use quotation marks to separate a value with semicolon from the split symbols.

Note that CSV files are databases. While we do not want to introduce the theory of databases in this thesis, note that our databases contain a *primary key*. A primary key uniquely identifies and provides access to information in the database. The main CSV file is called `DLMFMacro.csv`. It provides information about each macro. The primary key of this database is the macro itself. Further information consists of the name of the mathematical function, the hyperlink to the DLMF definition, the numbers of parameters, variables and the maximum number of @

Semantic Macro	Entry in the CSV file
`\LegendreP{\nu}@{x}`	`\LegendreP{\nu}@{x}`
`\LegendreP[\mu]{\nu}@{x}`	`X1:\LegendrePX\LegendreP[\mu]{\nu}@{x}`

Table 3.3: CSV entry example of the Legendre and associated Legendre function of the first kind.

symbols. The maximum number of @ symbols is mostly used for backward translations. Additionally, the CSV file can contain information about the constraints, branch cuts and a role. The additional role can be used to ignore the macro in the translation process, or to specify that this macro is a mathematical constant. For example, mathematical constants are ignored, because they are organized in the JSON files. This avoid multiple definitions in the database.

Since a translation is not necessarily bijective, the translations are defined in two separate CSV files per CAS. For Maple those files are `DLMF_Maple.csv` for the forward translation, and `CAS_Maple.csv` for the backward translation. Both are organized in the same already described logic.

In 3.2 we defined the definition for the Gudermannian function. The translated expression contains two function calls: the inverse tangent function and the hyperbolic sine function. Instead of providing a hyperlink to the definitions for both functions on the Maple help page, we only provide one hyperlink. We specify the hyperlink by a special prefix in the translation pattern. To explain this prefix, we need to take a look to optional parameters before. While the primary key in the CSV files is only the macro name, we need to specify the number of optional parameters in front of the macro name. Otherwise, for example, the Legendre and the associated Legendre function of the first kind would be linked to the same definition. Our prefix notation solve both problems. It links to one specific function for the translation and specify the correct version of the macro, if there are optional parameters included. The prefix starts and ends with an X and contains two information. The number of optional parameters (if none, than this number will be 0) and the name of the used function. Table 3.3 illustrates the lexicon entry of the Legendre and the associated Legendre function of the first kind.

In Maple the number of optional arguments is not stored in the dictionaries. Therefore, we link to the total number of arguments. Consider Olver's associated Legendre function. A forward translation can be defined with

$$\texttt{\textbackslash LegendreBlackQ\{\textbackslash nu\}@\{z\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{LegendreQ(\textbackslash nu,z)/GAMMA(\textbackslash nu+1).} \tag{3.8}$$

In this case, we would prefer to provide the hyperlink to the definition of Maple's `LegendreQ` function rather than to the definition of the `GAMMA` function. The used function has two arguments in Maple. Hence, the lexicon entry for the translation pattern is

$$\texttt{X2:LegendreQXLegendreQ(\$0,\$1)/GAMMA(\$1+1).} \tag{3.9}$$

All of the CSV files just store the information to provide an easy access for the developer. Nevertheless, the program itself works only with the *lexicon files*. Therefore, each CSV file

needs to get converted into a lexicon file. Subsection 3.1.3 explains how to do that and how to update the database.

JSON Libraries

Some translations are defined in separate JSON files. The reason is to separate translations for OPSF from translation for single symbols and generic LaTeX macros. These files therefore define translations for the following types expressions.

- **Generic LaTeX macros for functions**:
  For example, the macros `\frac{a}{b}`, `\binom{a}{b}` and `\sqrt{a}`. This section also defines translations for special symbols for mathematical functions such as the exclamation mark '`!`' to indicate the factorial function or `\mod` for the modulo function.
- **Mathematical Symbols**:
  For example, the symbol to indicate multiplications, sums, inequalities.
- **Greek Letters**:
  For example, `\alpha`, `\beta` or `\Theta`.
- **Mathematical Constants**:
  For example `\cpi` for the constant π or `\EulerConstant` for the Euler-Mascheroni constant γ.

Lexicon Files

The knowledge of the MLP is based on lexicon files, as explained in subsection 2.6.2. Since the MLP project was not a part of the DLMF/DRMF LaTeX macros project, the BNF grammar defines no rules about semantic macros. Furthermore, the knowledge database has no information about the semantic macros. Therefore, we created a new lexicon file that defines each semantic LaTeX macro and provides further information about it. This information is stored in the previously mentioned CSV files.

Table 3.4 presents the entry for the sine function in the `DLMF-macros-lexicon.txt`. The `global-lexicon.txt` file is the main dictionary of the MLP and combines the whole knowledge of the parser in one file. This lexicon files also help to translate ambiguous expressions. We will discuss some of these decision in the following sections. However, it is important to mention that a successful translation strongly depends on the comprehensive database of the lexicon files.

The concept of the lexicon files is also adapted to store the backward translations from Maple. Since the MLP project is developed to parse LaTeX expressions, it could be confusing to use the original lexicon files to also store names of Maple functions. Therefore, we create another type of lexicon files, which are very similar to the original lexicon files. In Maple a function can have multiple entries differing in the number of arguments. Therefore, the primary key of

Symbol: \sin
Feature Set: dlmf-macro
DLMF: \sin@@{z}
DLMF-Link: dlmf.nist.gov/4.14# E1
Meanings: Sine
Number of Parameters: 0
Number of Variables: 1
Number of Ats: 2
Maple: sin($0)
Maple-Link: www.maplesoft.com/support/ help/maple/view.aspx?path=sin
help/maple/view.aspx?path=sin
Mathematica: Sin[$0]
Mathematica-Link: reference.wolfram.com/ language/ref/Sin

Table 3.4: The entry of the sine function in the lexicon file.

the backward translation database is the combination of the function name and the number of arguments. For example, the name GAMMA in Maple is reserved for the Euler Gamma function in case of one argument, but also for the incomplete Gamma function in the case of two arguments. Therefore the lexicon file for the backward translation also contains the number of arguments to specify a translation.

3.1.3 Extending the Forward Translator

The whole project is still work in progress. Therefore, it is an important goal to keep it easy to update the backend libraries. This section will explain the typical workflow to add new translations for semantic LaTeX macros and Maple functions or how to support a forward translation to another CAS.

Adding Translations to Existing CAS

Since CSV files are not as easy to maintain as MS Excel[1] worksheets, the workflow starts with MS Excel. Once a user updates the existing tables, the MS Excel file needs to get exported as a CSV file. The translator project contains the `lexicon-creator.jar` file to convert the CSV files to the backend lexicon files. Of course, these steps are not necessary if the user just changes a translation in the JSON files.

Once the `lexicon-creator.jar` finished the conversion process, the translator is aware of all changes and ready to use.

[1] Microsoft Excel is widely used to work with tables and is able to import, export and manipulate CSV files.

Supporting New CAS

To support a complete new CAS for the forward translation, the user needs to add the CAS in the JSON files and create two new CSV files. Even if there is no backward translation implemented yet, the CSV file for the backward translation is needed.

Once all of the files are updated, the `lexicon-creator.jar` can be used to update the lexicon files.

3.2 The Forward Translation

The translation process starts with semantic LaTeX expressions. These expressions get analyzed and parsed by the MLP, which produces a PT of the input expression with additional information about each symbol. An abstract translator class analyzes each node and delegates them to specialized subtranslators. In this process an object called Translated Expression Object (TEO) is build. This TEO is needed to rebuild a string representation of the mathematical expression given by the PT.

The overall forward translation process is explained in figure 3.1. All translation patterns and related information are stored in the DLMF/DRMF tables. These tables are converted by the `lexicon-creator.jar` to the `DLMF-macros-lexicon.txt` lexicon file. Together with the `global-lexicon.txt` file, the PT will be created by the MLP. The `latex-converter.jar` takes a string representation of a semantic LaTeX expression and uses the MLP as well as our `Translator` to create an appropriate string representation for a specified CAS. The following sections 3.2.1 to 3.2.4 will focus on the `Translator`.

3.2.1 Analyzing the MLP-Parse Tree

The main task is to analyze the PT. Remember that the PT is different to expression trees as described in section 2.6.2.

Since the BNF does not define rules for semantic macros, each argument of the semantic macro and each @ symbol are following siblings of the semantic macro node. That is the reason, why we stored the number of parameters, variables and @ symbols in the lexicon files. Otherwise, the translator could not find the end of a semantic macro in the PT.

Figure 3.2 visualizes the PT of the Jacobi polynomial example from table 1.1. Because of these differences to expression trees, a backward conversion of the PT to a string representation can be difficult, especially for finding necessary or unnecessary parentheses. Therefore we create the TEO. Of course, the TEO is deeply incorporated into the translation process, but in this subsection we will focus on the general idea and process of the translation. We will focus on the usage of TEO in subsection 3.2.3.

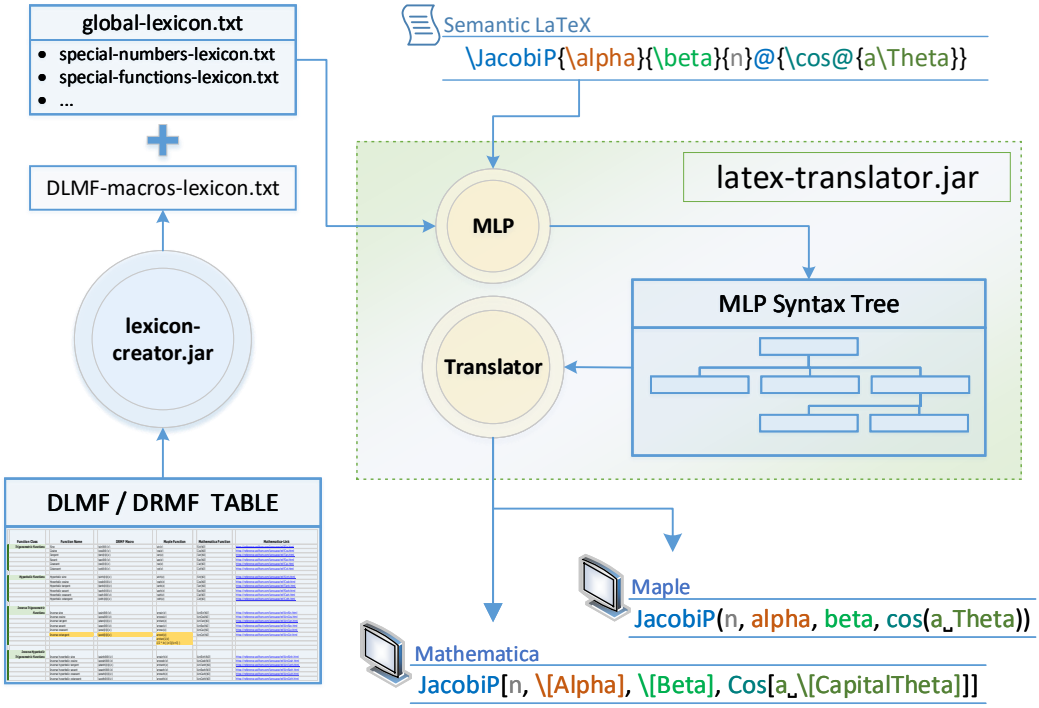


Figure 3.1: Process diagram of a forward translation process. The MLP generates the PT based on lexicon and JSON files. The PT will be translated to different CAS.

The general idea of a translation of a PT is a simple recursive algorithm, such as algorithm 1. Whenever the algorithm finds a leaf, it can translate this single term. If the node is not a leaf, it starts to translate all children of the node recursively.

But this approach is not completely feasible, because sometimes the algorithm needs to look ahead and check the following siblings for a valid translation, for example in case of a semantic macro with arguments (see figure 3.2). Therefore, the `TRANSLATE_LEAF` function also needs to know the following siblings of the current node. To avoid multiple translations of the same node, a list representation would be a better solution. Algorithm 2 explains an abstract version of the final algorithm for our translation tool.

Algorithm 1 Simple translation algorithm for the MLP-Parse Tree

Input: Root r of the parse tree T

```
1: procedure TRANSLATE(r)
2:     if r is leaf then
3:         TRANSLATE_LEAF(r);
4:     else
5:         for all children v_n of r do
6:             TRANSLATE(v_n);
7:         end for
8:     end if
9: end procedure
```

If the root r is a leaf, it can be translated as a leaf. Eventually, some of the following siblings are needed in order to translate r correctly. If r is not a leaf, it contains one or more children. Therefore, we can call the `ABSTRACT_TRANSLATOR` recursively for the children. Once we have translated r, we can go a step further and translate the

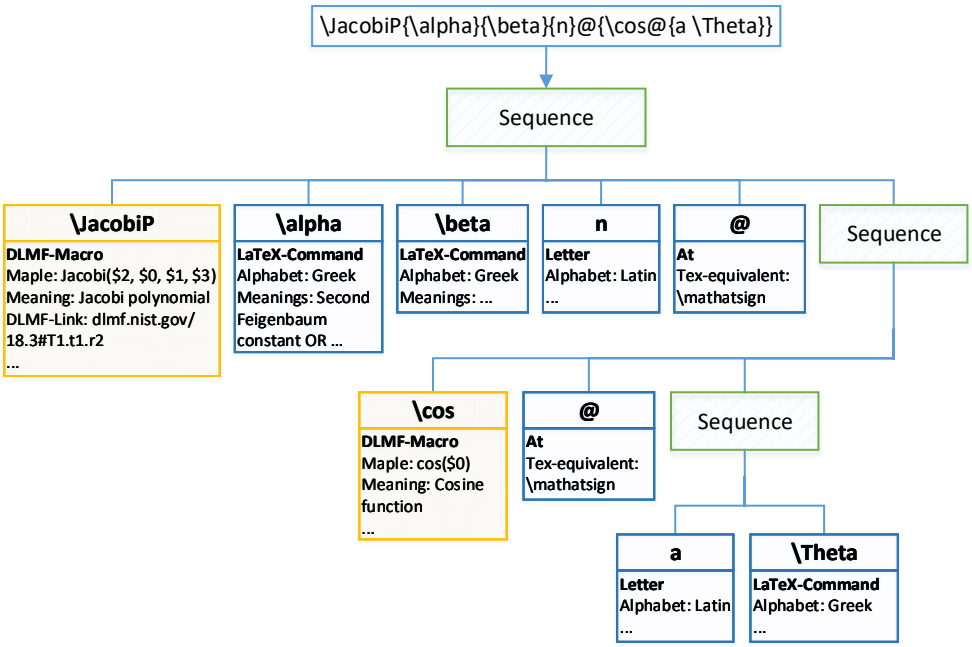


Figure 3.2: MLP-PT for a Jacobi polynomial using the DLMF/DRMF LATEX macro. Each leaf contains information from the lexicon files.

next node. Line 8 checks if there are following siblings left and calls the `ABSTRACT_TRANSLATOR` recursively in that case.

With this approach we are able to translate most of the semantic LATEX expressions. To take a look at the problems with this approach, we need to think about the operators and their standard notations in mathematics. There are many types of notation used to represent formulae. For example, the Normal Polish Notation (NPN)[2] (hereafter called prefix notation) places the operator to the left of/before its operands. The Reverse Polish Notation (RPN)[3] (hereafter called postfix notation) does the opposite and places the operator to the right of/after its operands. The infix notation is commonly used in arithmetic and places the operator between their operands, which only makes sense as long as the operator is a binary operator. For example, table 3.5 shows an expression in different notations.

In mathematical expressions, notations are mostly mixed, depending on the case and number of operands. For example, infix notation is common for binary operators ($+$, $-$, $\cdot$, mod etc.), while functional notations are obviously used for any kind of functions (*sin*, *cos*, etc.). Sometimes the same symbol is used in different notations to identify a different meaning. For example, the '$-$' as an unary operator is used in prefix notation to indicate the negative value of its operand, such as in '-2'. Of course, $-$ can also be the binary operator for subtraction, which is

[2] Also known as *Warsaw Notation* or *prefix notation*
[3] Also known as *postfix notation*

Algorithm 2 Abstract translation algorithm to translate MLP-Parse trees.

Input: Root r of a MLP-Parse tree T. List *following_siblings* with the following siblings of r. The list can be empty.

```
1:  procedure ABSTRACT_TRANSLATOR(r, following_siblings)
2:      if r is leaf then
3:          TRANSLATE_LEAF(r, following_siblings_of_r);
4:      else
5:          siblings = r.getChildren();          ▷ siblings is a list of children
6:          ABSTRACT_TRANSLATOR(siblings.removeFirst(), siblings);
7:      end if
8:      if following_siblings is not empty then
9:          r = following_siblings.removeFirst();
10:         ABSTRACT_TRANSLATOR(r, following_siblings);
11:     end if
12: end procedure
```

Notation	Expression
Infix	$(a + b) \cdot x$
Prefix	$\cdot + a\ b\ x$
Postfix	$a\ b + x \cdot$
Functional	$\cdot(+(a, b), x)$

Table 3.5: The mathematical expression '$(a + b) \cdot x$' in infix, prefix, postfix and functional notation.

commonly used in infix notation. An example for the postfix notation is factorial, such as '2!'.

Most programming languages (and CAS as well) internally use prefix or postfix notation and do not mix the notations in one expression, because it is easier to parse those notations. However, the common practice in science is to use mixed notations in expressions. Since the MLP has barely implemented mathematical grammatical rules, it takes the input as it is and does not build an expression tree. Therefore, it parses all four examples from table 3.5 to four different PTs rather than to one unique expression tree. In general, this is not a problem for our translation process because most CAS are familiar with the most common notations. Therefore, the translator does not need to know that a and b are the operands of the binary operator '+' in '$a + b$'. We could just translate the symbols in '$a + b$' in the same order as they appear in the expression and the CAS would understand it. However, this simple approach generates two problems.

1. The translated expression is only syntactically correct if the input expression was syntactically correct.
2. We cannot translate expressions to a CAS which uses a different notation.

Problem 1 should be obvious. Since we want to develop a translation tool and not a verification tool for mathematical LaTeX expressions, we can assume syntactically correct input expressions

and produce errors otherwise. Problem 2 is more difficult to solve. If a user wants to support a CAS that uses prefix or postfix notation by default, the whole translator would fail in its current state. Supporting CAS with another notation would be a part of future work and will be mentioned in chapter 6.

Nonetheless, changing a notation could also solve ambiguities in some situations. Consider the two ambiguous examples in table 3.6. While a scientist would probably just ask for the right interpretation of the first example, Maple automatically computes the first interpretation. On the other hand, LaTeX automatically disambiguate the first example by only recognizing the very next element (single symbols or sequence in curly brackets) for the superscript and therefore displays the second interpretation. The second example is already interpreted as the double factorial function of n, since this notation is the standard interpretation in science. We wrote the second interpretation as the standard way in science to make it even more obvious. However, surprisingly, Maple computes the first interpretation again rather than the common standard interpretation.

	Text Format Expression	First Interpretation	Second Interpretation
1:	4 ^ 2!	$4^{2!}$	$4^{2}!$
2:	n!!	$(n!)!$	$(n)!!$

Table 3.6: Ambiguous examples of the factorial and double factorial function. One expression in a text format can be interpreted in different ways.

In most cases, parentheses can be used to disambiguate expressions. We used them in table 3.6 to clarify the different interpretations in example 2. But sometimes, even parentheses cannot solve a mistaken computation. For example, there is no way to add parentheses to force Maple to compute $n!!$ as the double factorial function. Even $(n)!!$ will be interpreted as $(n!)!$. Rather than using the exclamation mark in Maple, one could also use the functional notation. For example, the interpretations (2!)! and (2)!! can be distinguished in Maple by using `factorial(factorial(2))` and `doublefactorial(2)` respectively.

As already mentioned, the structure of the PT makes it difficult to change the notation for all kind of operators. Therefore, and especially because of the examples above, we change the notation during the forward translation process to the functional notation only for the factorial and double factorial function. Thus,

$$\texttt{n!} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{factorial(n)},$$
$$\texttt{n!!} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{doublefactorial(n)}.$$

The translator needs to presume some properties for the functions similar to LaTeX, which only recognizes the very next element right after the caret for the superscript.

The biggest problem about translating the factorial (or double factorial) function is the common postfix notation for them. In this stage, algorithm 2 only translates the current node and if

necessary the following siblings, but not the preceding siblings. Consider an expression such as '4!'. When the variable *r* reaches the exclamation mark, the argument '4' of the function has already been translated. The current version of the algorithm has no access to this previously translated expression. Even more complicated, would be the case of the double factorial '4!!'. The following two approaches would solve this problem.

1. The translator always checks if the next two following siblings are one or two exclamation marks.
2. The current state of program has access to the previously translated expressions and can modify them retrospectively.

Approach 1 would be unnecessarily inefficient. Therefore we implement approach 2. That is another reason why we implement an extra object to organize translated expressions, the TEO. This object stores all parts of the previously translated expressions. The translator has access to these parts and is able to modify them afterwards. A more detailed explanation about the TEO follows in subsection 3.2.3.

	Node type	Explanation	Example
***r* has children**	Sequence	Contains a list of expressions.	$a + b$ is a sequence with three children (a, + and b).
	Balanced Expression	Similar to a sequence. But in this case the sequence is wrapped by `\left` and `\right` delimiters.	`\left(`$a + b$`\right)` is a balanced expression with three children (a, + and b).
	Fraction	All kinds of fractions, such as `\frac`, `\ifrac`, etc.	`\ifrac{a}{b}` is a fraction with two children (a and b).
	Binomial	Binomials	`\binom{a}{b}` has two children (a and b).
	Square Root	The square root with one child.	`\sqrt{a}` has one child (a).
	Radical with a specified index	n-th root with two children.	`\sqrt[a]{b}` has two children (a and b).
	Underscore	The underscore '_' for subscripts.	The sequence a_b has two children (a and '_'). The underscore itself '_' has one child (b).
	Caret	The caret '^' to for superscripts or exponents. Similar to the underscore.	The sequence $a\hat{}b$ has two children (a and '^'). The caret itself '^' has one child (b).
***r* is a leaf**	DLMF/DRMF LaTeX macro	A semantic LaTeX macro	`\JacobiP`, etc.
	Generic LaTeX macro	All kinds of LaTeX macros	`\Rightarrow`, `\alpha`, etc.
	Alphanumerical Expressions	Letters, numbers and general strings.	Depends on the order of symbols. $ab3$ is alphanumerical, while $4b$ are two nodes (4 and b).
	Symbols	All kind of symbols	'@', '$*$', '+', '!', etc.

Table 3.7: A table of all kinds of nodes in a MLP syntax tree. Note that this table groups some kinds for a better overview. For a complete list and a more detailed version see [63].

3.2.2 Delegation of Translations

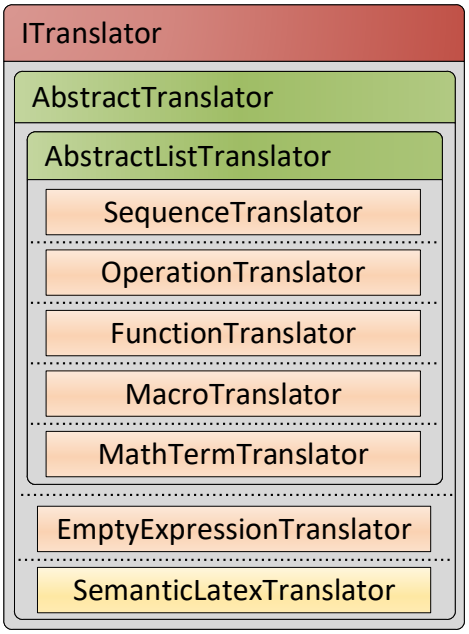


Figure 3.3: A scheme of the forward translator and its specialized subtranslators.

Algorithm 2 is a simplified version of the translator process. The main progress happens in lines 3 and 6. There are several cases what r can be. Table 3.7 on page 50 gives an overview of all different types in a PT. A more detailed explanation of the types can be found in [63].

The BNF grammar defines some basic grammatical rules for generic LaTeX macros, such as for `\frac`, `\sqrt`. Therefore, there is a hierarchical structure for those symbols, similar to the structure in expression trees. As already mentioned, some of these types can be translated directly, such as Greek letters, while others are more complex, such as semantic LaTeX macros. Therefore, the translators delegates the translation to specialized subtranslators. This delegation process is implemented in lines 3 and 6 of algorithm 2.

Figure 3.3 is a scheme of the forward translator and its subtranslators. The entry point for each translation process of the program is the `SemanticLatexTranslator`. It delegates the translations for the expression through the `AbstractTranslator` to the other subtranslators. The `AbstractTranslator` is the interface of all other translators and coordinates the translation process by delegating translations for specific nodes to specialized classes. For example, the `MacroTranslator` only translates semantic LaTeX macros. The `EmptyExpressionTranslator` is used to delegate the so called *empty expressions* to other classes. An *empty expression* is a node with children, but without a symbol itself, such as *sequences* and *balanced expressions*.

Subtranslators

The most interesting translators are the `MacroTranslator` (translating semantic LaTeX macros) and the `SequenceTranslator` (translating all kinds of *sequences*).

The `SequenceTranslator` translates the *sequence* and *balanced expressions* in the PT. If a node n is a leaf and the represented symbol an open bracket (parentheses, square brackets and so on) the following nodes are also taken as a *sequence*. Hence, combined with the recursive translation approach, the `SequenceTranslator` also checks balances of parentheses in expressions. An expression such as '$(a]$' is producing a mismatched parentheses error. On the other hand, this is a problem for interval expressions such as '$[a, b)$'. In the current version, the program cannot distinguish between mismatched parentheses and an half-opened, half-closed interval. Whether

an expression is an interval or another expression is difficult to decide and can depend on the context. In addition, the parenthesis checker could simply be deactivated to allow mismatched parentheses in an expression. But this functionality is not implemented yet.

Another problem that the `SequenceTranslator` solves is the position of multiplication signs in an expression. There are a couple of obvious choices to translate multiplication signs. For example, using `\cdot` or `\idot` will be obviously translated as a multiplication symbol. However, `\idot` is fairly new and therefore not frequently used yet. Furthermore, the most common symbol for multiplications is still the white space or none symbol at all, as explained above. Consider the simple expression $2n\pi$. The PT generates a sequence node with three children, namely 2, n and π. This sequence should be interpreted as a multiplication of the three elements. Therefore, the `SequenceTranslator` checks the types of the current and next nodes in the tree to decide if there should be a multiplication symbol or not. For example, if the current or next node is an operator, a relation symbol or an ellipsis there will be no multiplication symbol added. However, this approach implies an important property. The translator interprets all sequences of nodes as multiplications as long as it is not defined otherwise. This potentially produces strange effects. Consider an expression such as $f(x)$. Translate this to Maple will be `f*(x)`. But we do not consider this translation to be wrong, because there is a semantic macro to represent functions. In this case, the user should use `\f{f}@{x}` instead of `f(x)`.

Besides the translation of sequences, the translation process of the DLMF/DRMF LaTeX macros can be complex too; not only because of the structure of the PT, but also because of the complex definition of the semantic macros. As explained in 2.4.3, they can contain optional parameters, exponents are allowed before and after the arguments and the number of @ symbols varies. All these different cases will be solved by the `MacroTranslator`.

Algorithm 3 is the translation function of the `MacroTranslator` without error handling. It analyzes the next element right after the macro first. As mentioned, this can be one of the following:

- an exponent such as '`^2`'.
- an optional parameter in square brackets.
- a parameter in curly brackets (a *sequence* node in the PT) if none of the above.
- an @ symbols if none of the above.
- a variable in curly brackets (a *sequence* node) if none of the above.

An exponent will be translated and shifted to the end of the translated semantic macro, since it is more common to write the exponent after the arguments of a function in CAS. Therefore, the function translates and stores the exponent in line 5. One could ask what happens when there is an exponent given before and after the arguments. The `MacroTranslator` only translates the following siblings until each argument is translated. The first exponent will be shifted to the end. If right after the translated macro (with all arguments) follows another exponent, we interpret it as another exponent for the whole previous expression. In that case, it would be the macro with

Algorithm 3 The translate function of the MacroTranslator. This code ignores error handling.

Input:
macro - node of the semantic macro.
args - list of the following siblings of *macro*.
lexicon - lexicon file
Output:
Translated semantic macro.

```
1:  procedure TRANSLATE_MACRO(macro, args, lexicon)
2:      info = lexicon.getInfo(macro);
3:      argList = new List();              ▷ create a sorted list for the translated arguments.
4:      next = args.getNextElement();
5:      if next is caret then
6:          power = translateCaret(next);
7:          next = args.getNextElement();
8:      end if
9:      while next is [ do                 ▷ square brackets indicate optional arguments.
10:         optional = TRANSLATE_UNTIL_CLOSED_BRACKET(args);
11:         argList.add(optional);
12:         next = args.getNextElement();
13:     end while
14:     argList.add( TRANSLATE_PARAMETERS(args, info) );
15:     SKIP_AT_SIGNS( args, info );
16:     argList.add( TRANSLATE_VARIABLES(args, info) );
17:     pattern = info.getTranslationPattern();
18:     translatedMacro = pattern.fillPlaceHolders(argList);
19:     if power is not null then
20:         translatedMacro.add(power);
21:     end if
22:     return translatedMacro;
23: end procedure
```

the first translated exponent. Table 3.8 shows an example for the trigonometric cosine function with multiple exponents.

As you can see, the input expression in semantic LaTeX is interpreted as (3.10).

$$\left(\cos(x)^2\right)^2 \tag{3.10}$$

An open square bracket right after the semantic macro (after the exponent respectively) is an optional parameter. Since expressions in square brackets are not considered to be a *sequence* in the PT, the optional parameter is not grouped in a single node. Therefore, every node after the opening square bracket forms an expression for just one optional parameter until the algorithm reaches the closing square bracket. Such expressions in square brackets will be also translated by

Displayed As	$\cos^2(x)^2$
Semantic LaTeX	`\cos^2@{x}^2`
Translated Maple Expression	`((cos(x))^(2))^2`

Table 3.8: A trigonometric cosine function example with exponents before and after the argument.

the `SequenceTranslator` as described above. A semantic macro could have multiple optional parameters. Therefore, the function translates all optional parameters and stores them in line 9.

The lines 14 to 16 translates the parameters and variables. The @ symbols will be skipped. Nevertheless, the current program has an error handling implemented in line 15 if the maximum number of @ symbols is exceeded. In the following lines, the function takes the translation pattern from the lexicon and removes each placeholder by the corresponding argument. In addition, if there is an exponent right after the semantic macro, it will be attached to the translated semantic macro (see table 3.8). The parameters and variables are always grouped by curly brackets in *sequence*-nodes in the PT. Therefore, every argument after the optional parameters and potential exponent cannot be longer than one node in the PT.

Handle Ambiguities

Our semantic LaTeX expressions are not full semantic with respect to possible ambiguities in the expression. We already mentioned some techniques to solve ambiguities in semantic LaTeX expressions, for example how to handle double factorials. In table 3.9 are four examples of ambiguous expressions. LaTeX implemented the rules that the superscript or subscript is only one symbol (or a balanced expression in curly brackets). Therefore, each of these input expressions are not ambiguous in LaTeX. Since we talking about the forward translation, we should follow the rules of LaTeX to interpret ambiguous expressions.

Ambiguous Input	LaTeX Output
`n^m!`	$n^m!$
`a^bc^d`	$a^b c^d$
`x^y^z`	Double superscript error
`x_y_z`	Double subscript error

Table 3.9: Ambiguous LaTeX expressions and how LaTeX displays them.

Another more questionable translation decision are alphanumerical expressions. As explained in table 3.7, the MLP handles strings of letters and numbers differently, depending on the order of the symbols. The reason is, that an expression such as '$4b$' is usually considered to be a multiplication of 4 and 'b', while '$b4$' looks like indexing 'b' by 4. While the first example produces two nodes, namely 4 and 'b', the second example '$b4$' produces just a single alphanumerical node in the PT. The first expression would be translated as a multiplication of 4 and 'b'.

Because '$4b$' looks like '$4 \cdot b$' and the multiplication is commutative, we would assume that '$4b$' and '$b4$' are equivalent. This is one reason why we interpret such alphanumerical expressions as multiplications of the symbols. On the other hand, an expression such as '*energy*', would be an alphanumerical node as well and could be interpreted as a single variable named 'energy'. In that case, someone could wonder why the variable '*energy*' was translated as '$e \cdot n \cdot e \cdot r \cdot g \cdot y$'. While in physics or engineering variables possibly appear with longer names, such as the 'energy' example, it is more common to use single symbols for variables in mathematics [10]. Therefore, we interpret such alphanumerical expressions as multiplications of variables rather than one variable with an alphanumerical name.

Another, more elegant way to solve this problem would be to use the newly created semantic macro `\idot` (see subsection 2.4.4) to define an invisible multiplication symbol. However, because `\idot` is such a new semantic macro, it is not used in the DLMF or DRMF quite often yet. Therefore, our program still interprets alphanumerical expressions as multiplications.

Other ambiguities can appear, when the input expression contains a symbol, which is typically used to represent a specific mathematical object, but the user did not use the semantic macro to represent it. For example, the input expressions '$3i$' contains an 'i', which is usually associated with the imaginary unit. Since there is a semantic macro for the imaginary unit, namely `\iunit`, we do not translate the letter to the imaginary symbol in Maple and use the identity translation instead. Furthermore, we inform the user about the potential misusage of `i` (see 3.2.4).

In general the translator is drafted to solve ambiguous expressions or automatically find a workaround to disambiguate the expression. Only if there is no way to solve the ambiguity with the defined rules, the translation process stops.

3.2.3 Translated Expression Objects

It is sometimes necessary to provide access to already translated symbols or subexpressions. A typical example is the (double) factorial function, where the argument was translated before the function was registered by the translator (postfix notation). To realize this, the Translated Expression Object (TEO) is implemented. It is mainly a list object of translated expressions. It also groups subexpressions when it becomes clear that multiple symbols belong together. This typically happens when something is wrapped in parentheses but also when a function is completely translated.

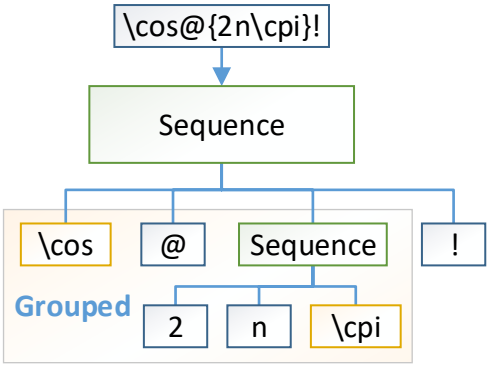


Figure 3.4: The MLP-PT for '$\cos(2n\pi)!$' and the grouped argument of the factorial function.

Figure 3.4 shows an example for the expression '$\cos(2n\pi)!$'. If we just analyze the PT, the previous node of the exclamation mark (which indicates the factorial function) is the sequence.

Now it could be possible to misinterpret the sequence as the argument of the factorial function. But the translator will first translate the cosine function, which has one argument. In that case the argument is the sequence node. Once the cosine function has been completely translated (which means the whole argument has been translated as well) the TEO groups the translated cosine and the sequence. Therefore, when the translator reaches the exclamation mark and asks for the last translated object in the list, it is not the translated sequence but the whole cosine function.

Table 3.10 shows some examples of the TEO list after the translation process was finished. Note that a translation of '$a + b$' contains three translated objects, while '$(a + b)$' contains just one.

Input Expression	TEO List
$a + b$	`[a, +, b]`
$(a + b)$	`[(a+b)]`
$\frac{a}{b} - 2$	`[(a)/(b), -, 2]`

Table 3.10: How the TEO-list groups subexpressions.

Local & Global Translated Expressions

During the translation process multiple local TEO are created by the subtranslators. Besides the local TEO there is just one global TEO. The reason to create local objects is originally motivated by the recursive structure of the translator. A subtranslator should only handle the current subexpression. However, the global TEO becomes necessary because of the functions in postfix notation such as the factorial function. For example, when a subtranslator tries to translate the exclamation mark as the factorial function, the local TEO is empty, because the argument of the function has been previously translated by another subtranslator and stored in another local TEO. Therefore, we implement a global TEO, which contains all of the previous translated subexpressions in each state of the translation process. With this implementation the local TEO becomes more and more obsolete. In the current version, the program stores redundant translated expressions. For debugging reasons, a translation process still handles multiple local and one global TEO. Starting the forward translation with the flag `--debug` (for the debugging mode) shows the elements of the global TEO after the translation has been finished.

3.2.4 Additional Information

As already mentioned, a translation is not always straight forward. In those cases, the user should get informed about the translation and the reason for it. Consider the translation

$$\texttt{3i} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{3*i}, \tag{3.11}$$

Since the imaginary unit in Maple is associated with '`I`' instead of '`i`', a user might be confused by 3.11. Therefore, we created the `InformationLogger` class to organize further information

about the translation process. In the case of 'i', the user is informed about the semantic macro \iunit and the potential misusage of 'i'.

Essentially, the InformationLogger provides further information about a translation process. This information is defined in the lexicon files. Consider the example of the arccotangent function, which is defined with a different branch cut in DLMF and in Maple. As already mentioned, we provide alternative translation patterns to solve these problems. These alternative translations, explanations, branch cuts, domains etc. will be automatically stored in the InformationLogger when an expression contains a mathematical object with further information.

3.3 Maple to Semantic LaTeX Translator

The backward translation is based on the internal data structure of Maple. Therefore, a license of Maple is mandatory to perform backward translations. The following subsections will explain why this approach was necessary and how Maple works internally. Using the internal data structure also requires to handle the internal properties of Maple. This can causes some problems. Those problems and our solutions will be explained in subsection 3.3.3. In the last subsection 3.3.4, we will focus on the implementation of the backward translation.

Note that all explanations in this section is based on Maple's programming guide [8] for Maple 2016.

3.3.1 Internal Data Structure

While our forward translation is realized for the 1D input representation in Maple, this representation is not suitable for a backward translation. Since Maple uses it's own syntax and programming language, we would need a new parser for Maple expressions to realize a backward translation tool. Of course, Maple has this parser implemented. But it is not possible to use the parser without using Maple itself. Therefore, an installed version of Maple is mandatory for our backward translation.

As already mentioned, Maple has several ways to represent an expression. The 1D and 2D representations are used as input representations. Besides those, there are internal data structures to perform all kinds of computations on the expression. Internally, each expression is handled and stored as a DAG. The Maple DAG is very similar to an expression tree. But for efficiency, it does not store copies of an object. Consider the integral from (2.25)

$$\int_0^\infty \frac{\pi + \sin(2x)}{x^2} dx. \tag{3.12}$$

As already mentioned, the Maple DAG has no copies of objects. Therefore, the 'x' in (3.12) only appears once in the DAG.

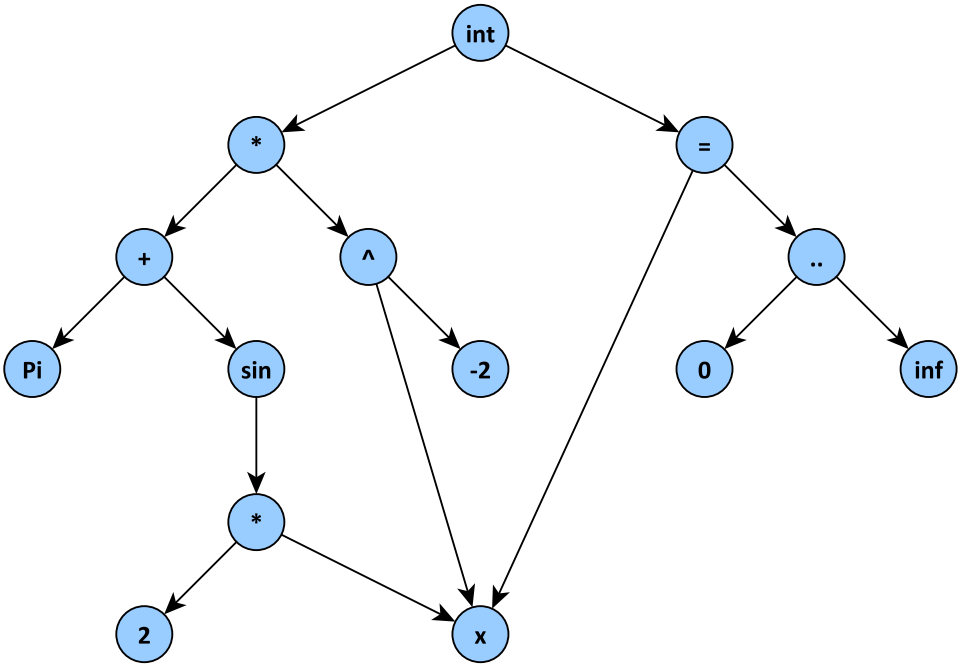


Figure 3.5: The Maple DAG of equation (2.25)

The Maple DAG can be visualized in a graph, such as in figure 3.5. But the internal representation looks a bit different. Each node in the DAG stores its children and has a header, which defines the type and the length of the node. Consider the polynomial $x^2 + x$. Figure 3.6 illustrates the internal DAG representation with headers and arguments.

Maple has also another representation, similar to the DAG, that allows multiple copies of the same object in its representation. This representation is called `InertForm` and is more similar to an expression tree than Maple's DAG. The `InertForm` represents the DAG as a tree by splitting object copies and displays the tree in a list structure. Each node (or element in the list) has the prefix `_Inert_XXX`, that indicates the node is in the `InertForm`. The suffix `XXX` is the type of the node. Some of the important types are specified in table 3.11. Note that this is only a subset of all possible types.

3.3.2 Maple's Open Maple API

Maple provides an Application Programming Interface (API), called `OpenMaple` [8, §14.3] for interaction with the Maple kernel. The `OpenMaple` API is implemented for different programming languages such as `C`, `Java` and `Visual Basic`. Since our forward translation and the MLP is implemented in `Java`, we use the `Java` API for the backward translation. The `Java` API has some limitations to interact with Maple's DAG.

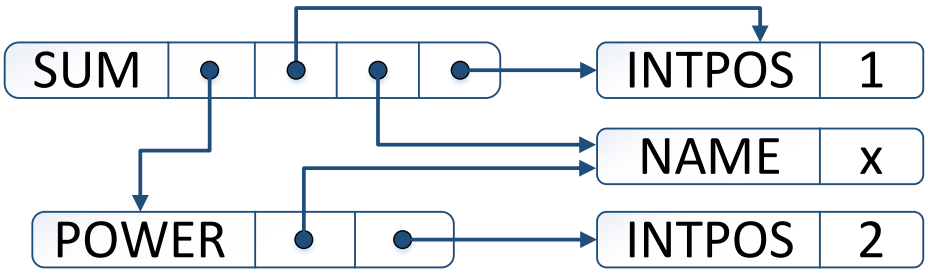


Figure 3.6: The internal Maple DAG representation of $x^2 + x$.

Type	Explanation
SUM	Sums.
PROD	Products.
EXPSEQ	Expression sequence is a kind of list. The arguments of functions are stored in such sequences.
INTPOS	Positive integers.
INTNEG	Negative integers.
COMPLEX	Complex numbers with real and imaginary part.
FLOAT	Float numbers are stored in the scientific notation with integer values for the exponent n and the significand m in $m \cdot 10^n$.
RATIONAL	Rational numbers are fractions stored in integer values for the numerator and positive integers for the denominator.
POWER	Exponentiation with expressions as base and exponent.
FUNCTION	Function invocation with the name, arguments and attributes of the function.

Table 3.11: A subset of important internal Maple structures. See [8] for a complete list.

For the backward translation, we use the `InertForm`. This representation can be converted to a nested list version and the `OpenMaple` Java API can handle lists. Therefore, the backward translation firstly converts the `InertForm` to a nested list structure before any translation process starts.

The following figure 3.7 shows the `InertForm` and the nested list of the polynomial $x^2 + x$.

In the nested list, the first element specifies the type of the current list while the following arguments store the arguments and optional attributes.

```
_Inert_SUM(
    _Inert_POWER(
        _Inert_NAME("x"),
        _Inert_INTPOS(2)
    ),
    _Inert_NAME("x")
)
```

```
[_Inert_SUM,
    [_Inert_POWER,
        [_Inert_NAME, "x"],
        [_Inert_INTPOS, 2]
    ],
    [_Inert_NAME, "x"]
]
```

Figure 3.7: `InertForm` and the nested list representation of $x^2 + x$.

3.3.3 Workarounds for Problems

Note that the backward translations usually do not have problems with ambiguous expressions. Every CAS needs to solve ambiguities in its own representation, otherwise it could not compute the expression. Our backward translation from Maple back to semantic LaTeX is therefore based on already disambiguated expressions.

The general walkthrough of our backward translation starts with a string of a `1D` Maple input expression. As described above, this string is entered in Maple to parse the expression and work on the nested list version of the `InertForm` afterwards. This causes some problems. Mostly because Maple automatically evaluates input expressions immediately - even before we have a chance to take a look at the internal data structure of the input. For example, an input such as $\sin\left(\frac{\pi}{2}\right)$ is immediately evaluated to 1 and the internal data structure just shows us the positive integer value 1 instead of the trigonometric function.

Furthermore, Maple also changes input expressions slightly, mostly because of missing data types for a specific expression. For example, the internal data structure has no type to represent fractions except the `RATIONAL` type, which only allows integer values for the numerator and denominator. Thereby, an expression like $\frac{x}{2}$ is automatically changed to $x\frac{1}{2}$. There is no way no avoid such simple arithmetic changes in the expression, but we can avoid the automatic evaluations.

Firstly, we summarize all known changes and evaluations Maple automatically performs on input expressions.

- Maple evaluates input expressions immediately.
- There is no data type to represents square roots such as $\sqrt{x}$ (or n-th roots). Therefore, Maple stores roots as an exponentiation with a fractional exponent. For example, $\sqrt{x}$ is stored as $x^{\frac{1}{2}}$.
- There is no data type for subtractions, only for sums. Negative terms are changed to absolute values times '-1'. For example, $x - y$ is stored as $x + y \cdot (-1)$.
- Floating point numbers are stored in the scientific notation with a mantissa and an exponent in the base 10. For example, 3.1 is internally represented as $31 \cdot 10^{-1}$.

- There is only a data type for rational numbers (fractions with integer numerator and positive denominator), but not for general fractions, such as $\frac{x+y}{z}$. This will be automatically changed to $(x+y)\cdot z^{-1}$.

There are unevaluation quotes implemented to avoid evaluations on input expression. Table 3.12 gives an example how those unevaluation quotes work.

	Without unevaluation quotes	With unevalation quotes
Input expression:	`sin(Pi)+2-1`	`'sin(Pi)+2-1'`
Stored expression:	1	`sin(Pi)+1`

Table 3.12: Example of unevaluation quotes for `1D` Maple input expressions.

Since we want to keep a translated expression similar to the input expression, we use unevaluation quotes during the whole translation process. Unevaluation quotes also solve the problems with square roots and n-th roots. In Maple's `1D` input representation, a square root is a function call (`sqrt(x)`) and the unevaluation quotes prevent evaluations on functions. Therefore, a backward translation simply needs to translate a square root as a normal function. Hence, we do not have problems with the internal representation of roots in Maple.

Because of the absent data type for subtractions, a long sum with negative terms could be difficult to read. We change the order of constants (such as an integer value) to be in front of products. Therefore, an expression such as '$-y$' is not represented by '$y\cdot(-1)$', but by '$(-1)\cdot y$'. The translator now needs to check if there is a leading '-1' and can translate it to '$-y$' rather than to '$(-1)\cdot y$'.

Floating point numbers in scientific notation also causes a problem. Consider the input expression 0.41 as the nested list `InertForm` representation

```
[_Inert_FLOAT, [_Inert_INTPOS, 41], [_Inert_INTNEG, 2]].        (3.13)
```

The backward translator is implemented in `Java`. To translate such an expression back to 0.41, computations become necessary. Of course, it makes sense to perform those computations in the CAS (in that case, in Maple) rather than in the translation program. Therefore, we implemented a procedure[4] to automatically convert `FLOAT` nodes to a string representation of the floating point number. We created a new type, called `MYFLOAT` to represent such numbers. With this approach, we avoided computations in the translation process.

For fractions, we use a similar approach and introduce a new internal data type `DIVIDE`. The implementation of `DIVIDE` was difficult. On the one hand, some expressions with negative exponent should be displayed as a fraction while others should should be displayed with the negative exponent. We follow the same rule as Maple follows internally to display fractions. When the exponent is a negative numeric element, the expression should be displayed as a fraction. If the exponent is something else, we uses the negative exponent representation. Table 3.13

[4] *Procedures* in Maple are small programs similar to methods and functions in Object-Oriented Programming (OOP) languages.

shows two examples with negative exponents. One of them is displayed as a fraction (exponent '-2') and the other is not (exponent '$-2y$'). We implement the same rule for our newly developed `DIVIDE` data type. Note that with the new created data type, we are still not able to avoid changes from $\frac{x}{2}$ to $x\frac{1}{2}$, but we are able to work with more general fraction expressions.

	Negative numeric exponent	General negative exponent
`1D` Maple input expression	`(x-1)^(-2)`	`(x-1)^(-2*y)`
How Maple displays the input	$\frac{1}{(x-1)^2}$	$(x-1)^{-2y}$

Table 3.13: Different styles to display negative exponents depending on the type of the exponent. Maple displays expressions with negative numeric exponents as fractions, while other negative exponents are not displayed as fractions.

We implement all of these changes before the actual translation process starts. This helps keeping the translated expression similar to the input expression.

3.3.4 Implementation Details

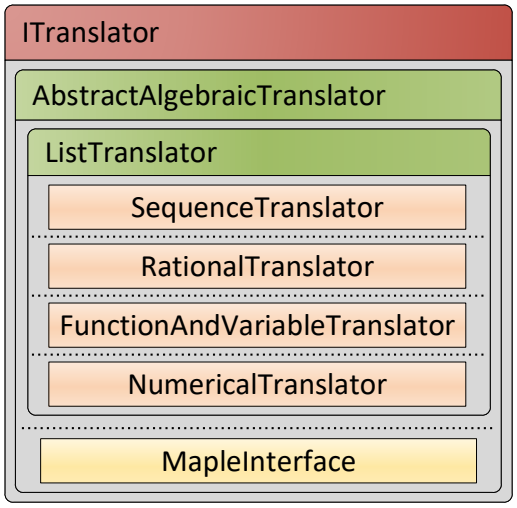


Figure 3.8: A scheme of the backward translator and its specialized subtranslators.

The general translation process runs through different states. Each translation process starts with the string representation of a `1D` Maple input. This string will be converted to the internal nested list representation of our custom `InertForm` (see the previous subsection 3.3.3). Since the `InertForm` is similar to an expression tree, it can be translated as the PT in the forward translation. Hence, we use the same approach of multiple subtranslators for the backward translation process.

Figure 3.8 shows the subtranslators of the backward translation process. The structure is the same as for the forward translation, seen in figure 3.3. The `MapleInterface` represents the entry point of a translation process. Before the translation starts, the Maple kernel needs to be initialized and prepared for the translation process by loading all custom procedures. Once everything is loaded, an expression is delegated to specialized subtranslators depending on the internal data type, defined by custom procedures and Maple's `InertForm`.

The following figure 3.9 illustrates the backward translation process for the Jacobi polynomial example $P_n^{(\alpha,\beta)}(\cos(a\Theta))$ from table 1.1. The input expression is converted to the nested list representation of the `MyInertForm` (the customized `InertForm` of Maple). Afterwards, each node of the tree is translated separately by specialized subtranslators (visualized by blue arrows). The translation of functions (bold blue arrows) is again realized by translation patterns, which define the correct position for each argument. The last step is used to put all translated arguments to the right position (visualized by red arrows).

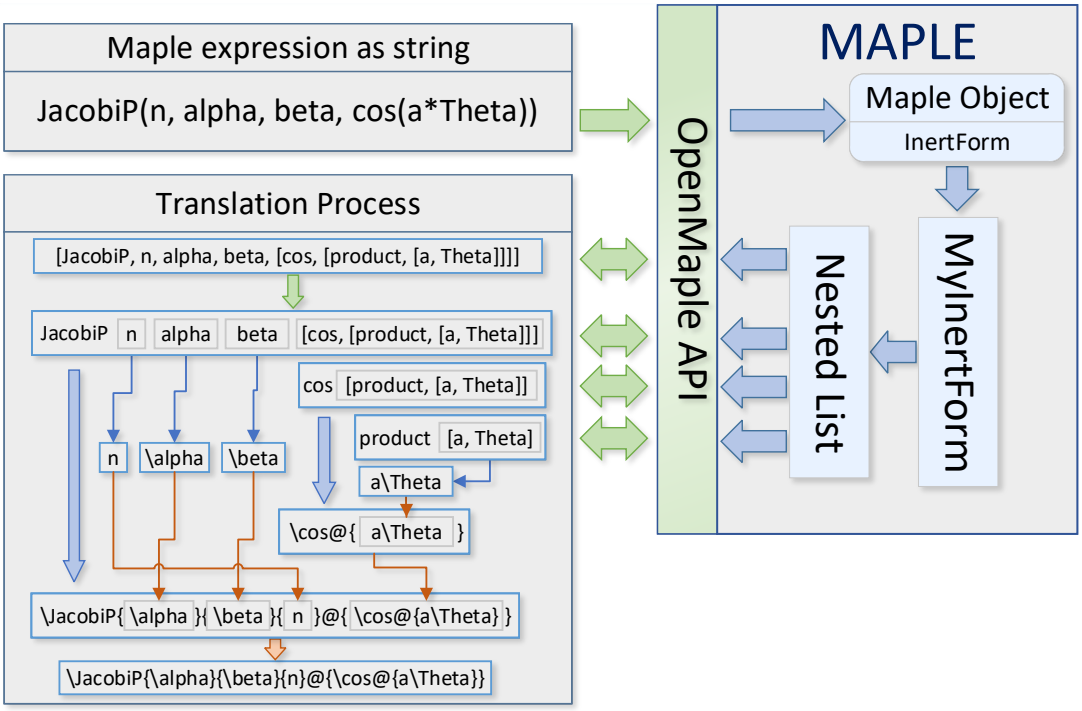


Figure 3.9: A scheme of the backward translation process from Maple for the Jacobi polynomial $P_n^{(\alpha,\beta)}(\cos(a\Theta))$. The input string is converted by the Maple kernel into the nested list representation. This list is translated by subtranslators (blue and red arrows). A function translation (bold blue arrows) is again realized by translation patterns to define the position of the arguments (red arrows).

Chapter 4

Evaluation

This project aims for a translation of mathematical expressions from LaTeX sources to CAS and back again. Essentially, the translator translates only string representations. We need to verify that such translations are *appropriate*. Therefore, we develop verification and validation techniques for translations and present them in this chapter.

First of all, each semantic LaTeX macro ties a specific character sequence to a well-defined mathematical object. This definition can be found at the DLMF or DRMF. Therefore, our reference source for semantic LaTeX expressions is always the DLMF or DRMF. This gives us the opportunity to compare more than the string representations of the mathematical objects in a translation process. For example, consider the simple translation of the cosine function

$$\texttt{\textbackslash cos@\{x\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{cos(x)}. \tag{4.1}$$

Our verification and validation techniques, to proof for example that (4.1) is an *appropriate* translation, can be grouped into three parts. We present in the following sections these three parts and conclusively, we use all these techniques to actually verify translations for a test suite directly extracted from the DLMF.

4.1 Round Trip Tests

Round trip tests are our first approach to verify translations. The idea is to translate an expression back and forth again and check if the representation changes during this process. The translator is able to translate expressions from and to Maple. Therefore, we implement the following round trip tests only for Maple.

Furthermore, we take advantage of Maple's symbolic simplification function to compare a Maple expression in the beginning of an round trip test with the Maple expression after the translations. If Maple's *simplify* function returns zero for the difference between two representations, both

expressions are symbolically equivalent. Note that this relies on the correctness of the simplification, which is not proven (see section 2.5.2). On the other hand, the equivalence is not disproven if the simplification returns something different to zero.

A round trip test always starts with a valid expression either in semantic LaTeX or in Maple. A translation from one system to another is called **a step**. A complete round trip translation (two steps) is called **one cycle**.

Definition 4.1: *(Fixed Point Representation)*
*A **fixed point representation** (or short fixed point) in a round trip translation process is a string representation that is identical to all string representations in the following cycles.*

Obviously, there may appear two fixed point representations in round trip tests, one for each system. We will see that most of the round trip translations reach fixed points after a certain amount of steps.

Theorem 4.1: *(Fixed Point Properties)*
Consider two languages $A, B \in \mathcal{L}$. A fixed point representation is reached if the current representation is just identical to the next representation in the same language (after one cycle). Furthermore, if $a \in A$ is a fixed point representation, the translation of a to $b \in B$ is also a fixed point representation.

Proof. Let $A, B \in \mathcal{L}$ be languages. Consider a round trip translation test between the languages A and B. Without loss of generality, let $a \in A$ be a fixed point representation and let a be the *first* fixed point representation that appears in this test. The MLP as well as Maple are deterministic, because a test case works in only one Maple-session and we do not compute the inputs in Maple. Therefore, the translation process is not able to call any probabilistic algorithms. Hence, our translator is also deterministic.

Let $b \in B$ with

$$\mathrm{tr}_B^A(a) = b. \tag{4.2}$$

Since the translation process is deterministic, a will be always translated to b. Consider $a' \in A$ as the backward translation from b

$$\mathrm{tr}_A^B(b) = a'. \tag{4.3}$$

Now a was defined to be a fixed point representation in A. Therefore, a is identical to all string representation in the following cycles. Thus $a = a'$.

Furthermore, because the translations are deterministic, b will be always backward translated to $a' = a$ and a will be always forward translated to b. Therefore, b is identical to the representation after one cycle. Per definition, b is therefore a fixed point in B.

Remember, a was chosen arbitrarily as the first fixed point representation that appears in the round trip test. Therefore, whenever a fixed point appears in a round trip test, the next translation to the respective other language will be a fixed point as well. Consequently, a fixed point is already reached, when just the next representation (after one cycle) is identical to the current representation. □

With theorem 4.1, we are able to find fixed points in round trip tests by only testing the equality to the next representation (after one cycle). Furthermore, we only need to find the first fixed point representation in either system, because the next step will be a fixed point representation for the respective other system. When a round trip test reaches such a fixed point in Maple, the *simplify* function tries to simplify the difference between the first expression in Maple and the fixed point expression. It is not feasible to simplify or compute LaTeX expressions yet (neither generic nor semantic LaTeX) - actually, our translation tool just makes this possible. Therefore, we check the equivalence between a fixed point and its first expression in semantic LaTeX manually.

Table 4.1 illustrates an example of a round trip test which reaches a fixed point. The test formula is

$$\frac{\cos(a\Theta)}{2}. \tag{4.4}$$

Steps	semantic LaTeX/Maple representations
0	`\frac{\cos@{a\Theta}}{2}`
1	`(cos(a*Theta))/(2)`
2	`\frac{1}{2}\idot\cos@{a\idot\Theta}`
3	`(1)/(2)*cos(a*Theta)`
4	`\frac{1}{2}\idot\cos@{a\idot\Theta}`

Table 4.1: A round trip test reaching a fixed point.

Step 4 is identical to step 2. Theorem 4.1 implies that step 2 is the fixed point representation for semantic LaTeX and step 3 is the fixed point representation for Maple. The structure of the expression changes between the steps 1 and 2. The reason are internal changes made by Maple. These problems have been discussed in the previous section 3.3.3.

Almost all test cases are reaching a fixed point representation. Starting from semantic LaTeX, the first fixed point is reached at step 2 in semantic LaTeX, because of Maple's internal changes. If the structure of the semantic LaTeX input is already identical to the internal structure of Maple's representation and the expression only contains functions and symbols with an available direct translation from and to each system, the input expression is already the fixed point representation. For example, consider the semantic LaTeX expression in step 2 in table 4.1 as the input expression. In that case the input is already a fixed point. Starting a round trip test from Maple, the first fixed point representation is reached in step 1 in semantic LaTeX, because Maple's changes has already been performed after the first translation. Again, this is only true if the expression only contains functions and symbols with a direct translation from and to each system. Furthermore, if a test case reaches a fixed point, it is equivalent (checked by Maple's *simplify* function or manually) to the input expression.

There is currently only one exception known where a round trip test does not reach a fixed point representation at all: Legendre's incomplete elliptic integrals [21, (19.2.4-7)] are defined with the amplitude ϕ in the first argument in the DLMF, while Maple takes the trigonometric sine of the amplitude as the first argument. Therefore, the forward and backward translations are

defined as

$$\texttt{\EllIntF@\{\phi\}\{k\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{EllipticF(sin(phi),k)}, \tag{4.5}$$

$$\texttt{\EllIntF@\{\asin@\{\phi\}\}\{k\}} \overset{\mathfrak{M}_{aple}}{\mapsfrom} \texttt{EllipticF(phi,k)}. \tag{4.6}$$

A round trip translation for this function produces an infinite chain of sine and inverse sine calls. The problem is that the round trip tests prevent simplifications in Maple during the translation process (see section 3.3.3). This problem could be resolved by allowing automatic evaluations in the round trip tests. However, currently we do not allow those evaluations because the changes can be serious, which makes it hard to follow possible errors.

The round trip tests are very successful, but they only detect errors in string representations. However, because of the simplification techniques of fixed points, we are able to at least detect logical errors in one system: Maple. On the other hand, these tests cannot determine logical errors in the translations between the two systems - but this is the critical part we want to verify. Consider defining a forward and backward translation for the sine function obviously wrong as

$$\texttt{\sin@\{\phi\}} \overset{\mathfrak{M}_{aple}}{\leftrightarrow} \texttt{cos(phi)}, \tag{4.7}$$

$$\texttt{\cos@\{\phi\}} \overset{\mathfrak{M}_{aple}}{\leftrightarrow} \texttt{sin(phi)}. \tag{4.8}$$

In that case the round trip test would not detect any errors and reach a fixed point representation, because the simplification techniques only simplify two representations in the same system but cannot compare the representation in one system to those in the other. Therefore, we implemented tests for equations rather than general formulae.

4.2 Function Relation Tests

In addition to the correctness of string representations, it is more important to validate whether a translation is *appropriate*. Special functions are highly related to each other. The theory of special functions researches these relations and other properties of the functions. The DLMF is a compendium of special functions and lists a lot of those relations. The translation of such relations to another system and the following analysis of the relations with the translated formulae would form a very effective test.

With this technique we can detect translation errors such as in (4.7 and 4.8). Consider the DLMF equation for the sine and cosine function [21, (4.21.2)]

$$\sin(u+v) = \sin u \cos v + \cos u \sin v. \tag{4.9}$$

Assume the translator would forward translate the expression based on (4.7, 4.8). Than

$$\texttt{\sin@\{u + v\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{cos(u + v)}, \tag{4.10}$$

$$\texttt{\sin@@\{u\}\cos@@\{v\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{cos(u)*sin(v)}, \tag{4.11}$$

$$\texttt{\cos@@\{u\}\sin@@\{v\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{sin(u)*cos(v)}. \tag{4.12}$$

This produces the equation in Maple

$$\cos(u+v) = \cos(u)\sin(v) + \sin(u)\cos(v), \tag{4.13}$$

which is wrong. Since the expression is correct before the translation, we conclude an error during the translation process.

However, there are two essential problems with this approach. Testing the mathematical equivalence of expressions is hard to solve and CAS have trouble to test even simple equations. Furthermore, this approach only checks forward translations because there is no way to check equivalence of expressions in LaTeX automatically (again this could become feasible with our translator). We use Maple's *simplify* function to check if the difference of the left-hand side and the right-hand side of the equation is equal to zero. In addition, we use *simplify* and check if the division of the right-hand side by the left-hand side returns a numerical value or not. This simplification function is the most powerful function to check the equivalence in Maple. However, there are several cases where the simplification fails. We implement some tricks to help *simplify*: pre-conversion of the functions of the expression to other special functions is able to help *simplify*, because the algorithms in the background can use different simplification methods for different functions. For example, the user can use the Maple function *convert* to convert the expression

$$\sinh(x) + \sin(x) \tag{4.14}$$

to an equivalent representation using the exponential function:

$$\frac{1}{2}e^{x} - \frac{1}{2}e^{-x} - \frac{1}{2}i\left(e^{ix} - e^{-ix}\right). \tag{4.15}$$

With such pre-conversions we are able to improve the simplification process in Maple. However, the limitations of the *simplify* function are still the weakest part of this verification approach. Consider the equation (2.9) from section 2.2.3.

$$U(0,z) = \sqrt{\frac{z}{2\pi}}\, K_{\frac{1}{4}}\left(\frac{1}{4}z^2\right) \tag{4.16}$$

As already discussed, the modified Bessel function of the second kind [21, (12.7.10)] has its branch cut in Maple at $z < 0$. However, the argument of K contains z^2. For now, just focus on

$$K_{\frac{1}{4}}\left(\frac{1}{4}z^2\right) \tag{4.17}$$

Because of z^2, if $|\operatorname{ph}(z)| \in \left(\frac{\pi}{2}, \pi\right)$ the value of (4.17) would be no longer on the principal branch. However, Maple will still compute (4.17) on the principal branch independently of the value of z. Hence, a translation

$$\texttt{\textbackslash BesselK\{\textbackslash frac\{1\}\{4\}\}@\{\textbackslash frac\{1\}\{4\}z\^{}2\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{BesselK(1/4,(1/4)*z\^{}2)} \tag{4.18}$$

is incorrect if $|\operatorname{ph}(z)| \in \left(\frac{\pi}{2}, \pi\right)$ and one has to use analytic continuation for the right-hand side of equation (4.16). In subsection 2.2.3 we also shown that the complete right-hand side of

equation (4.16) has three branch cuts, caused by the square root function and the z^2 argument of the modified Bessel function of the second kind. Such complex examples cannot be simplified yet.

However, this validation approach is powerful and runs automatically. We use this approach to verify a large set of test cases extracted directly from the DLMF, see section 4.4.

4.3 Numerical Tests

Another approach of verification are numerical tests. In principle, we follow the same approach as the relation tests in section 4.2, but rather than using the simplification functions we compare the left-hand side and right-hand side for actual values.

For example, reconsider equation (4.16) from [21, (12.7.10)]. Define the difference of right-hand side and left-hand side

$$D(z) := U(0, z) - \sqrt{\frac{z}{2\pi}}\, K_{\frac{1}{4}}\left(\frac{1}{4}z^2\right), \tag{4.19}$$

and compute $D(z)$ for some values of z. Table 4.2 presents four computations for $D(z)$, one value for each quadrant in the complex plane.

z	$D(z)$
$1 + i$	$2 \cdot 10^{-10} - 2 \cdot 10^{-10} i$
$-1 + i$	$2.222121916 - 1.116719816 i$
$-1 - i$	$2.222121916 + 1.116719816 i$
$1 - i$	$2 \cdot 10^{-10} + 2 \cdot 10^{-10} i$

Table 4.2: Four computations of $D(z)$ in Maple.

The values are computed with the default precision of 10 significant digits. Caused by the machine accuracy, the first and last values are different from zero. In such cases, we can increase the precision to figure out if it is an accuracy issue or $D(z)$ is really different from zero at those points. Increasing the precision of the computation for the first value $z = 1 + i$ up to 100 significant digits returns $3 \cdot 10^{-100} i$ and can therefore be considered as zero (we get the same results for $z = 1 - i$). Hence, table 4.2 shows us what we already know. The second and third values are computed on the principal branch of the modified Bessel function of the second kind and therefore produces a difference between the left-hand side and right-hand side of equation (4.16).

This approach of verification is very powerful, because it can verify not only our translations but also equations in a mathematical compendium. This has been demonstrated by discovering an overall sign error of the DLMF equation (4.20), see section 4.4.

However, this approach has also a significant problem. It is easy to see that something went wrong, but very hard to proof equivalence. Obviously, only four tested values are not sufficient

to proof the equivalence. How many and which values do we need for a comprehensive result? Reformulating this question to a general problem brings us to:

> *How can we discretize equations down to a finite set of test points such that the probability of equivalence for the continues equation is high if the equation is valid for the finite set of test points?*

As far as we know, there is no answer to this question yet. This is one reason why we do not implement an automatic process of numerical tests yet. Another reason are constraints. Constraints are not as easy to translate as general formulae. They often contain quantifiers or logical operators which are difficult to translate in the current state of the program. However constraints are necessary for numerical tests.

4.4 DLMF Test Suite

To verify our translations with the approaches described above we create test suites by extracting equations and formulae from the sources of the DLMF. We start with a small test suite of 110[1] equations and provide a more detailed look to those tests. We are able to translate 108 of those 110 cases.

Two of the test cases (lines 42 and 43) contain macros we are not able to translate yet. Those macros are `\subplus` and `\CFK`, which are used for continued fractions. They have been created for another translation project [16, §5] for continued fractions defined in the *Handbook of Continued Fractions for Special Functions* (2008) [19]. The conversion process for the Continued Fractions for Special Functions (CFSF) [6] dataset from Maple and the Encoding Continued Fraction Knowledge (eCF) [59, 60] dataset from Wolfram uses special file types. Therefore, our translator is not able to handle those macros yet.

However, we translate 108 test cases and for 84 of them the simplification function of Maple returns zero as the difference of the left-hand side and right-hand side. The 24 remaining cases are translated, but the simplification function is not able to simplify the difference to zero. Therefore, we try to improve this part. Line 46 [21, (18.5.10)] for example, can be simplified to zero, if n is previously set to an arbitrary integer value. For all those 24 test cases, we perform also manual numerical tests, which reveal an overall sign error in equation [21, (14.5.14)][2]

$$P_\nu^{1/2}(\cosh\xi) = -\left(\frac{2}{\pi\sinh\xi}\right)^{1/2}\cosh\left(\left(\nu+\tfrac{1}{2}\right)\xi\right). \tag{4.20}$$

Besides this wrong equation, all of these manual numerical tests return zero for the difference between the left-hand side and right-hand side. Of course it is still not possible to conclude the equivalence based on our manual numerical tests, as explained above.

[1] The set of test cases is available at
https://gist.github.com/AndreG-P/f48121d70fbd2a57b2cf6d88019ddadf

[2] The equation had originally been stated as shown in equation (4.20). The DLMF has already updated the wrong equation.

Those 110 test cases are not sufficiently comprehensive for all of the functions of the DLMF. Therefore, we create a larger dataset[3] of 4,165 semantic LaTeX formulae, which is extracted from all chapters of the DLMF.

We are able to translate 2,232 (approx. 53.59%) of those test cases. The *simplify* function of Maple returns zero for the difference for 490 of the translated 2,232 test cases. We increase the number to 713 with automatic pre-conversions helping the simplification process. With simplification over the division of the left-hand side by the right-hand side (rather than simplify the difference) we have three more verified translations and 716 in total. Therefore, 1,516 test cases were translated but the translation cannot be verified automatically yet.

The translator cannot translate 1,933 test cases. First of all, 32 lines contain neither an equation nor a relation. Because we have created this dataset for our verification approaches, we do not translate those 32 lines at all. Most of the other failures are produced because of missing semantic information in the LaTeX expression or the formulae contains a macro that cannot be translated in the current state of the program, such as in 978 lines that contain a macro that cannot be translated. For example, in 171 lines one of the symmetric elliptic integrals [21, (19.16.1-6)] appears and we do not provide translation patterns for those integrals, because there is no direct translation possible yet. A workaround would be to allow translations based on the definitions, see (3.1).

Other errors appear in 914 cases, where the semantic information is missing or is not unique. An example are 284 lines that use a potentially ambigious single quote (or prime) notation ’’’ to identify derivatives of functions.

[3] We are planning to publish this dataset at `http://drmf.wmflabs.org`.

Chapter 5

Results & Current Status

The goal is to develop an independent, lite translation tool between mathematical LaTeX expressions and CAS. The semantic LaTeX macros are constantly changing. Therefore, the translator can easily be updated as well, without implementing new code. Furthermore, we have implemented forward translations to the CAS Maple and Mathematica and a backward translation from Maple. The translator is therefore able to allow complete round trip translations between Maple and semantic LaTeX. We have summarized the results of those validation techniques in chapter 4.

The translator is based on not published code from the MLP and POM-tagger. Furthermore, the DLMF/DRMF LaTeX macros are not published yet as well. Therefore, the current version of the translator is not online available yet. However, we are planning to publish the translator soon on the DRMF website[1].

Our translator knows 665 DLMF/DRMF LaTeX macros. We have defined forward translations for 201 of those to Maple and 195 functions for backward translation from Maple, plus translations for all Greek letters, constants and generic LaTeX macros. The small percentage of covered macros is the translator's most important weakness. The reason is that most of the functions which are represented by a DLMF/DRMF LaTeX macro are not defined in Maple. For example, the multivalued versions of the inverse trigonometric functions [21, (4.23.1-6)] and the q-generalizations are not defined in Maple. A possible workaround is to define alternative translations based on the definition or other representations of the function, see the equations 2.6 as an example.

Nonetheless, the translator is able to handle a wide range of input expressions. This is shown by the test case of several formulae directly taken from the DLMF. Out of 4165 test equations, 716 have been successfully forward translated to Maple and proven to be correct by the internal simplification functions of Maple. In addition, 1519 have been successfully forward translated to Maple, but all validation methods failed. Thus, the translator is able to translate approximately 53.59% of all test cases. Numerical test cases can advance the validation process and have been

[1] http://drmf.wmflabs.org

proven to be very powerful, but implementations of such automatic numerical tests are very complex and therefore not realized yet.

5.1 Bijection of Translations

One of the main goals has been the provision of bijective mappings between semantic LaTeX and CAS. This goal has not been achieved, because of the large number of functions that are only defined in one system, but not in the other. Our translations are in general neither injective nor surjective.

The forward translation is not injective, because there exist multiple macros for the same mathematical object. For example, the ultraspherical orthogonal polynomial (or Gegenbauer polynomial) has two macros and both are translated to the same function in the CAS. One of the macros is defined in the DLMF, while the other is defined by the DRMF. For the background translation we choose the DLMF representation.

$$\begin{aligned}
\texttt{\textbackslash Ultra\{\textbackslash lambda\}\{n\}@\{x\}} &\overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{GegenbauerC(n, lambda, x)},\\
\texttt{\textbackslash Ultraspherical\{\textbackslash lambda\}\{n\}@\{x\}} &\overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{GegenbauerC(n, lambda, x)},\\
\texttt{\textbackslash Ultraspherical\{\textbackslash lambda\}\{n\}@\{x\}} &\overset{\mathfrak{M}_{aple}}{\mapsfrom} \texttt{GegenbauerC(n, lambda, x)}.
\end{aligned}$$

Deleting those multiple definitions from the set of macros would make a forward translation injective. Note that this is only true if two expressions are defined as identical as soon as their semantic meanings are identical. For example, formulae with a different number of white spaces are technically not identical, but will be translated to the same expression. However, those formulae can be defined as identical.

The forward translation is not surjective either, because obviously there are expressions in CAS which the translator would never produce. The same problems appear for the backward translation. For example, a backward translation from Maple can never produce $\frac{x}{2}$, because of Maple's internal changes it will only produce $x\frac{1}{2}$. Therefore, a backward translation is not surjective. Furthermore, it is not injective because of functions that cannot be translated yet.

However, for a subset of functions and expressions a bijective mapping is possible and has been proven by round trip tests. The main goal is therefore to enlarge these subsets. Mathematically, we looking for subsets $L_1 \subseteq \mathfrak{sL}$ and $L_2 \subseteq \mathfrak{M}_{aple}$, such that $|L_1|$ and $|L_2|$ are maximal and $\mathrm{tr}_{L_2}^{L_1}$ is bijective and its inverse function is $\mathrm{tr}_{L_1}^{L_2}$.

5.2 Supported CAS

We support a forward translation to Maple and Mathematica. Nevertheless, the forward translation to Mathematica is in an early state and only support a few translations for functions. Since the main work is already done for Maple, one only needs to update the libraries for a better support of Mathematica.

In theory it is possible to support even more CAS, but this has not been tested yet. Especially difficult are CAS that use different mathematical notations in their input format, such as the prefix notation. Those CAS cannot be supported as long as the MLP does not produce expression trees.

A backward translation has only been implemented for Maple yet and it needs to have a licensed version of Maple installed on the system to work, because access to the internal expression tree of Maple expressions is necessary for the translator. Therefore, possible backward translations from other CAS also need access to the internal data structure.

5.3 Open Problems

One still unsolved problem are different branch cuts in two systems for the same mathematical function. Our workaround only provides additional information to the user, who needs to handle this issue on his own. However, we have presented an approach to solve this problem. Instead of translating the function itself, the translator could use equivalent formulae for the translation that only contain functions that use the same positions for branch cuts in both systems.

On the other hand, this approach does not solve the problem of the relation (4.16), between the modified Bessel function of the second kind and the parabolic cylinder function.

$$U(0,z) = \sqrt{\frac{z}{2\pi}}\, K_{\frac{1}{4}}\left(\frac{1}{4}z^2\right)$$

In this case, the problem is not an inconsistent definition of the position of the branch cuts, but that the variable z jumps over branch cuts. While this is generally an issue of CAS, the translator could solve it automatically by using analytic continuation if necessary. This presumes that the translator can compute semantic LaTeX expressions to detect critical values of z^2. Together with a CAS, this approach is feasible.

Another open problem are the verification techniques for translated expressions. While the backward translations are mostly verified by round trip tests, almost all of the translations need a powerful equivalence checker in the CAS. The limitations of this functionality are an important open problem for our validations. Furthermore, we currently need to assume the correctness of the used simplification functions. A bug in the simplification process spuriously verifying or mistakenly refusing the translation must be taken into consideration.

An approach to solve this problems are the described numerical tests. However, the verification of equivalence with a finite set of discrete values in a sufficient way, is an open problem.

Besides all of this, the translator still has problems with unclear semantics. One example are the already explained prime symbols to indicate derivatives. Another example are mathematical conventions for notations. For example, an expression

$$\sin^{-1}(x) \tag{5.1}$$

is mostly interpreted as the arcsine function $\arcsin(x)$. However, our translator translates it as

$$\texttt{\sin\^{}\{-1\}@\{x\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{1/(sin(x))}. \tag{5.2}$$

Another problem are mismatched parenthesis. An expression for an half-opened interval $(a, b]$ currently returns an error because of the mismatch of the parenthesis. Nonetheless, this expression is valid for the representation of an interval.

We have also discovered a bug in the grammatical rules for the square root LaTeX macro. The expression

$$\texttt{\sqrt \frac\{a\}\{b\}} \tag{5.3}$$

produces an error. The generic LaTeX macros always taken a fixed number of following expression as arguments. It is possible to group an expression with curly brackets. However, if there are no curly brackets used, a LaTeX compiler interprets just one symbol as the argument. Therefore, the expression '`\frac{a}{b}`' and '`\frac ab`' produces the same output: $\frac{a}{b}$. The next symbol can also be a LaTeX macro, such as in expression (5.3). Because the BNF rule of the MLP defines curly brackets currently as mandatory around the argument of the square root function, the MLP produces an error during the parsing process.

Chapter 6

Conclusion & Future Work

The results and conclusions of the translations of the formula set from the DLMF are diverse. Some test cases have shown us that the set of semantic macros is not perfect yet and needs to be improved, seen in the example of the macro definition for the Wronskian symbol. However, simply adding more and more macros to improve the set is not the right decision. In that case we would end up in something similar to STEX, which is overloaded and too complicated to use. Obviously, if the system is no longer handy, it will not be used.

On the other hand, the concept of the translator has been proven. For example, one test case has revealed an overall sign error in an equation in the DLMF [21, (14.5.14)] and therefore the validation system has shown the potential to become a strong verification tool for mathematical compendia. The test cases have also shown how difficult it is to validate a translated expression and have uncovered the problems of translations between two systems with different sets of supported functions. Our validation techniques also assume the correctness of the simplification and computational algorithms in CAS. However, combining those techniques and automatically running translation checks can potentially not only discover errors in mathematical compendia, but also detect errors in simplifications or computations in CAS.

Finally we can conclude that the project has been successful and the translator has great potential to become a handy translation tool. Nonetheless, it is without any doubts an ever-growing project and needs to be improved constantly.

6.1 Approaches for Further Improvements

As already discussed, the percentage of defined translations is small. Therefore, extending the backend libraries with additional translations for more semantic macros is the first obvious task to improve the translator. Adding more CAS would also be worthwhile for forward and backward translations. Some CAS or other software use other notations than the infix notations. Therefore, another improvement for the translator would be the support for other types of no-

tations. This could be achieved by using expression trees during the translation process rather than the PT generated by the MLP.

Another minor issue is the way of representing functions after a backward translation. The DLMF/DRMF LaTeX macros allow different representations and control the different styles by the number of @ symbols. Currently the backward translation simply adds the largest possible number of @ symbols, which has undesirable effects. Categorizing the functions (Possible categories may be *trigonometric functions* or **polynomials**) can be used to define the right typical number of @ symbols for the most common representations.

Besides the extensions for the backend libraries, we saw in section 4.3 that automatic numerical tests are not implemented yet. Certainly, automatic numerical tests are a powerful approach to verify translations and also to find errors in the mathematical sources. However, the realization of such powerful numerical tests is very difficult. As far as we know, there is no general theory for such methods developed yet. An approach could be to adapt error handling methods in telecommunications such as the cyclic redundancy checks [48] that use checksums to detect errors. Adapting such approaches for general mathematical functions could be helpful.

The most desirable improvement for the translator is the support for generic LaTeX. Since the semantic LaTeX macros are newly invented and generic LaTeX is still the de facto standard for mathematical expressions, it is definitely worthwhile to support generic LaTeX later on. However, we already explained the difficulties of this task. One approach we want to realize is the **multiple-scan approach**. This approach is inspired by the approach of humans. When a human reader reads mathematical expressions, he concludes the correct semantics from the structure and the context of the expression. His conclusions highly depend on his own knowledge. If he is not familiar with a mathematical symbol, he has difficulties to understand the whole expression.

The multiple-scan approach tries to adapt these techniques with the following three objectives:

(1) narrow down possible meanings only from the expression itself, without referring to the context of the expression;

(2) refine the process with conclusions from the nearby context of the expressions and

(3) improve the previous process by analyzing not only the nearby context but the overall topic of the whole scientific paper or book, its references and other publications by the authors.

Starting with the lexicon files from the POM-tagger, we want to narrow down the possible meanings of an expression. It is planned to extend the POM-tagger from [63] to achieve objective (1). Furthermore, a large-scale corpus study has shown that around 70 percent of the symbolic elements in scientific papers are denoted in the surrounded text [62]. With this awareness we can achieve objective (2) by just searching the close context [47, 53, 54]. If the correct semantic information is still unsure, objective (3) is the last way to find a solution. Online compendia, such as arXiv, can be used to discover the overall topic of a scientific paper, the references and the area of research of the authors. The MCAT search engine developed by Kristianto, Topic and Aizawa [36, 45] is able to extract and score information from the document at this '*document*

granularity level'. Ideally, there is only one possible interpretation left after step (3). Otherwise, even a human reader might have difficulties to understand the expression.

Figure 6.1 visualizes the process of conclusions for the Jacobi polynomial example from table 1.1.

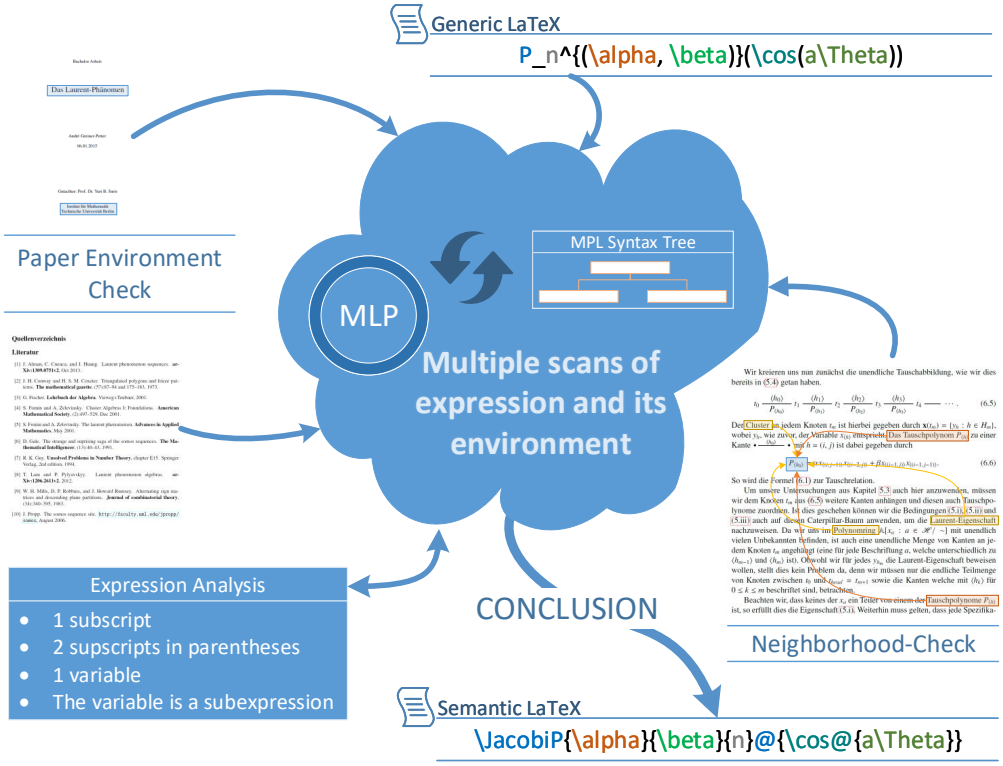


Figure 6.1: Visualization of the multiple-scan approach. Exemplified using the Jacobi polynomial example.

With such improvements the translator possibly becomes multifunctional: from automatic translations between word processors and computer algebra systems over to a verification tool for mathematical compendia, then over to a tool that can automatically enrich mathematical expressions with semantic information. A combination of the automatic semantically enrichment process and LaTeX extraction tools [7] would be even hypothetically able to semantically enrich already published articles in PDF format afterwards.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] Mark J. Ablowitz and Athanassios S. Fokas. *Complex Variables*. Cambridge University Press, Apr. 28, 2003. 660 pp. ISBN: 978-0521534291. URL: http://www.ebook.de/de/product/3261876/mark_j_ablowitz_athanassios_s_fokas_complex_variables.html.

[2] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. ninth Dover printing, tenth GPO printing. New York: Dover, 1964.

[3] Gaudeul Alex. "Do Open Source Developers Respond to Competition? The (La)TeX Case Study". In: *Review of Network Economics* 6.2 (June 2007), pp. 1–25. ISSN: 1446-9022. DOI: 10.2202/1446-9022.1119. URL: https://ideas.repec.org/a/bpj/rneart/v6y2007i2n9.html.

[4] George E. Andrews, Richard Askey, and Ranjan Roy. *Special Functions*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1999. ISBN: 9781107325937. DOI: 10.1017/CBO9781107325937.

[5] Ron Ausbrooks, Stephen Buswell, David Carlisle, et al. *Mathematical Markup Language (MathML) 3*. Tech. rep. Version 3.0 2nd Edition. International Organization for Standardization (ISO), 2014. URL: https://www.w3.org/TR/MathML3/.

[6] Franky Backeljauw and Annie Cuyt. "Algorithm 895: A Continued Fractions Package for Special Functions". In: *ACM Trans. Math. Softw.* 36.3 (July 2009), 15:1–15:20. ISSN: 0098-3500. DOI: 10.1145/1527286.1527289. URL: http://doi.acm.org/10.1145/1527286.1527289.

[7] Josef B. Baker, Alan P. Sexton, and Volker Sorge. "MaxTract: Converting PDF to LaTeX, MathML and Text". In: *Intelligent Computer Mathematics - 11th International Conference, AISC 2012, 19th Symposium, Calculemus 2012, 5th International Workshop, DML 2012, 11th International Conference, MKM 2012, Systems and Projects, Held as Part of CICM 2012, Bremen, Germany, July 8-13, 2012. Proceedings*. Ed. by Johan Jeuring, John A. Campbell, Jacques Carette, Gabriel Dos Reis, Petr Sojka, Makarius Wenzel, and Volker Sorge. Vol. 7362. Lecture Notes in Computer Science. Springer, 2012, pp. 422–426. ISBN: 978-3-642-31373-8. DOI: 10.1007/978-3-642-31374-5_29. URL: https://doi.org/10.1007/978-3-642-31374-5_29.

[8] L. Bernardin, P. Chin, P. DeMarco, et al. *Maple 2016 Programming Guide*. Maplesoft, a division of Waterloo Maple Inc., 2016. ISBN: 978-1-926902-46-3.

[9] Bruno Buchberger, George E. Collins, Rüdiger Loos, and R. Albrecht. "Computer algebra symbolic and algebraic computation". In: *ACM SIGSAM Bulletin* 16.4 (1982), p. 5. DOI: 10.1145/1089310.1089312. URL: http://doi.acm.org/10.1145/1089310.1089312.

[10] Florian Cajori. *A History of Mathematical Notations*. Dover Publications Inc., Mar. 1, 1994. 848 pp. ISBN: 0486677664. URL: http://www.ebook.de/de/product/1683538/florian_cajori_a_history_of_mathematical_notations.html.

[11] Martin Campbell-Kelly, Mary Croarken, Raymond Flood, and Eleanor Robson, eds. *The History of Mathematical Tables: From Sumer to Spreadsheets*. OXFORD UNIV PR, Oct. 11, 2003. 376 pp. ISBN: 978-0-19-850841-0. DOI: 10.1093/acprof:oso/9780198508410.001.0001. URL: http://www.ebook.de/de/product/2768294/the_history_of_mathematical_tables_from_sumer_to_spreadsheets.html.

[12] Timothy Y. Chow. "What is a Closed-Form Number?" In: *The American Mathematical Monthly* 106.5 (May 1999), pp. 440–448. DOI: 10.2307/2589148. URL: http://www.jstor.org/stable/2589148.

[13] Berkeley Churchill and Steven Boyd. *LaTeXCalc*. https://sourceforge.net/projects/latexcalc/. Seen 06/2017. 2010.

[14] Howard S. Cohl, Marjorie A. McClain, Bonita V. Saunders, Moritz Schubotz, and Janelle C. Williams. "Digital Repository of Mathematical Formulae". In: *Intelligent Computer Mathematics - International Conference, CICM 2014, Coimbra, Portugal, July 7-11, 2014. Proceedings*. Ed. by Stephen M. Watt, James H. Davenport, Alan P. Sexton, Petr Sojka, and Josef Urban. Vol. 8543. Lecture Notes in Computer Science. Springer, 2014, pp. 419–422. ISBN: 978-3-319-08433-6. DOI: 10.1007/978-3-319-08434-3_30. URL: https://doi.org/10.1007/978-3-319-08434-3_30.

[15] Howard S. Cohl, Moritz Schubotz, Marjorie A. McClain, Bonita V. Saunders, Cherry Y. Zou, Azeem S. Mohammed, and Alex A. Danoff. "Growing the Digital Repository of Mathematical Formulae with Generic LaTeX Sources". In: *Intelligent Computer Mathematics - International Conference, CICM 2015, Washington, DC, USA, July 13-17, 2015, Proceedings*. Ed. by Manfred Kerber, Jacques Carette, Cezary Kaliszyk, Florian Rabe, and Volker Sorge. Vol. 9150. Lecture Notes in Computer Science. Springer, 2015, pp. 280–287. ISBN: 978-3-319-20614-1. DOI: 10.1007/978-3-319-20615-8_18. URL: https://doi.org/10.1007/978-3-319-20615-8_18.

[16] Howard S. Cohl, Moritz Schubotz, Abdou Youssef, André Greiner-Petter, Jürgen Gerhard, Bonita V. Saunders, Marjorie A. McClain, Joon Bang, and Kevin Chen. "Semantic Preserving Bijective Mappings of Mathematical Formulae Between Document Preparation Systems and Computer Algebra Systems". In: *Intelligent Computer Mathematics - 10th International Conference, CICM 2017, Edinburgh, UK, July 17-21, 2017, Proceedings*. Ed. by Herman Geuvers, Matthew England, Osman Hasan, Florian Rabe, and Olaf Teschke. Vol. 10383. Lecture Notes in Computer Science. Springer, 2017, pp. 115–131. ISBN: 978-3-319-62074-9. DOI: 10.1007/978-3-319-62075-6_9. URL: https://doi.org/10.1007/978-3-319-62075-6_9.

[17] Robert M. Corless, David J. Jeffrey, Stephen M. Watt, and James H. Davenport. ""According to Abramowitz and Stegun" or arccoth needn't be uncouth". In: *ACM SIGSAM*

Bulletin 34.2 (2000), pp. 58–65. DOI: 10.1145/362001.362023. URL: http://doi.acm.org/10.1145/362001.362023.

[18] Hans Cuypers, Arjeh M. Cohen, Jan Willem Knopper, Rikko Verrijzer, and Mark Spanbroek. "MathDox, a system for interactive Mathematics". In: *Proceedings of EdMedia: World Conference on Educational Media and Technology 2008*. Ed. by Joseph Luca and Edgar R. Weippl. Vienna, Austria: Association for the Advancement of Computing in Education (AACE), June 2008, pp. 5177–5182. URL: https://www.learntechlib.org/p/29092.

[19] Annie A. M. Cuyt, Vigidis Brevik Petersen, Brigitte Verdonk, Haakon Waadeland, and William B. Jones. *Handbook of Continued Fractions for Special Functions*. Springer, 2008. ISBN: 978-1-4020-6948-2. URL: http://www.springer.com/de/book/9781402069482#aboutBook.

[20] James H. Davenport. "The Challenges of Multivalued "Functions"". In: *Intelligent Computer Mathematics, 10th International Conference, AISC 2010, 17th Symposium, Calculemus 2010, and 9th International Conference, MKM 2010, Paris, France, July 5-10, 2010. Proceedings*. Ed. by Serge Autexier, Jacques Calmet, David Delahaye, Patrick D. F. Ion, Laurence Rideau, Renaud Rioboo, and Alan P. Sexton. Vol. 6167. Lecture Notes in Computer Science. Springer, 2010, pp. 1–12. ISBN: 978-3-642-14127-0. DOI: 10.1007/978-3-642-14128-7_1. URL: https://doi.org/10.1007/978-3-642-14128-7_1.

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

[22] Dan Drake. *sagetex*. https://ctan.org/tex-archive/macros/latex/contrib/sagetex/. Seen 06/2017. Comprehensive, June 2009.

[23] Antonio J. Durán, Mario Pérez, and Juan Luis Varona. "Misfortunes of a mathematicians' trio using Computer Algebra Systems: Can we trust?" In: *CoRR* abs/1312.3270 (2013). URL: http://arxiv.org/abs/1312.3270.

[24] Matthew England, Edgardo S. Cheb-Terrab, Russell J. Bradford, James H. Davenport, and David J. Wilson. "Branch cuts in Maple 17". In: *ACM Comm. Computer Algebra* 48.1/2 (2014), pp. 24–27. DOI: 10.1145/2644288.2644293. URL: http://doi.acm.org/10.1145/2644288.2644293.

[25] *Formal Language - Encyclopaedia of Mathematics, Supplement III*. Springer Netherlands, Nov. 23, 2007. ISBN: 978-0-306-48373-8. URL: https://www.encyclopediaofmath.org/index.php/Formal_language.

[26] Wikimedia Foundation. *Wikipedia*. https://en.wikipedia.org. Jan. 2001.

[27] Herman Geuvers, Matthew England, Osman Hasan, Florian Rabe, and Olaf Teschke, eds. *Intelligent Computer Mathematics - 10th International Conference, CICM 2017, Edinburgh, UK, July 17-21, 2017, Proceedings*. Vol. 10383. Lecture Notes in Computer Science. Springer, 2017. ISBN: 978-3-319-62074-9. DOI: 10.1007/978-3-319-62075-6. URL: https://doi.org/10.1007/978-3-319-62075-6.

[28] Jana Giceva, Christoph Lange, and Florian Rabe. "Integrating Web Services into Active Mathematical Documents". In: *Intelligent Computer Mathematics, 16th Symposium, Calculemus 2009, 8th International Conference, MKM 2009, Held as Part of CICM*

2009, Grand Bend, Canada, July 6-12, 2009. Proceedings. Ed. by Jacques Carette, Lucas Dixon, Claudio Sacerdoti Coen, and Stephen M. Watt. Vol. 5625. Lecture Notes in Computer Science. Springer, 2009, pp. 279–293. ISBN: 978-3-642-02613-3. DOI: 10.1007/978-3-642-02614-0_24. URL: https://doi.org/10.1007/978-3-642-02614-0_24.

[29] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979. ISBN: 0-201-02988-X.

[30] William L. Hosch. *Special Function*. Encyclopædia Britannica. Encyclopædia Britannica. Sept. 2006. URL: https://www.britannica.com/topic/special-function.

[31] Donald E. Knuth. *Digital Typography*. Reissue. Lecture Notes (Book 78). Center for the Study of Language and Information (CSLI), June 11, 1998. 685 pp. ISBN: 978-1575860107.

[32] Donald Ervin Knuth. *The Art of Computer Programming, Volume I: Fundamental Algorithms, 3rd Edition*. Addison-Wesley, 1997. ISBN: 0201896834. URL: http://www.worldcat.org/oclc/312910844.

[33] Michael Kohlhase. *OMDoc - An Open Markup Format for Mathematical Documents [version 1.2]*. Lecture Notes in Computer Science. Springer, 2006. ISBN: 3-540-37897-9. DOI: 10.1007/11826095. URL: https://doi.org/10.1007/11826095.

[34] Michael Kohlhase. "Using LaTeX as a Semantic Markup Format". In: *Mathematics in Computer Science* 2.2 (2008), pp. 279–304. DOI: 10.1007/s11786-008-0055-5. URL: https://doi.org/10.1007/s11786-008-0055-5.

[35] Michael Kohlhase, Joseph Corneli, Catalin David, et al. "The Planetary System: Web 3.0 & Active Documents for STEM". In: *Proceedings of the International Conference on Computational Science, ICCS 2011, Nanyang Technological University, Singapore, 1-3 June, 2011*. Ed. by Mitsuhisa Sato, Satoshi Matsuoka, Peter M. A. Sloot, G. Dick van Albada, and Jack Dongarra. Vol. 4. Procedia Computer Science. Elsevier, 2011, pp. 598–607. DOI: 10.1016/j.procs.2011.04.063. URL: https://doi.org/10.1016/j.procs.2011.04.063.

[36] Giovanni Yoko Kristianto, Goran Topic, and Akiko Aizawa. "MCAT Math Retrieval System for NTCIR-12 MathIR Task". In: *Proceedings of the 12th NTCIR Conference on Evaluation of Information Access Technologies, National Center of Sciences, Tokyo, Japan, June 7-10, 2016*. Ed. by Noriko Kando, Tetsuya Sakai, and Mark Sanderson. National Institute of Informatics (NII), 2016. URL: http://research.nii.ac.jp/ntcir/workshop/OnlineProceedings12/pdf/ntcir/MathIR/04-NTCIR12-MathIR-KristiantoGY.pdf.

[37] Leslie Lamport. *Latex Document Preparation System Users*. Addison Wesley Publishing Co, 1985. ISBN: 0-201-15790-X. URL: https://www.amazon.com/Latex-Document-Preparation-System-Users/dp/020115790X?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=020115790X.

[38] Daniel W. Lozier. "NIST Digital Library of Mathematical Functions". In: *Ann. Math. Artif. Intell.* 38.1-3 (2003), pp. 105–119. DOI: 10.1023/A:1022915830921. URL: https://doi.org/10.1023/A:1022915830921.

[39] Anil Maheshwari and Michiel Smid. *Introduction to Theory of Computation*. Seen 07/2017. School of Computer Science, Carleton University, Ottawa, Canada, Mar. 2017. URL: http://cglab.ca/~michiel/TheoryOfComputation/TheoryOfComputation.pdf.

[40] Christopher D. Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT Press, 2001. ISBN: 978-0-262-13360-9.

[41] Maplesoft. *Produce output suitable for LaTeX 2e*. http://www.maplesoft.com/support/help/Maple/view.aspx?path=latex. Since version Maple 18, Seen 06/2017.

[42] MathWorks. *LaTeX form of symbolic expression in MATLAB*. https://www.mathworks.com/help/symbolic/latex.html. Seen 06/2017.

[43] Bruce R. Miller and Abdou Youssef. "Technical Aspects of the Digital Library of Mathematical Functions". In: *Ann. Math. Artif. Intell.* 38.1-3 (2003), pp. 121–136. DOI: 10.1023/A:1022967814992. URL: https://doi.org/10.1023/A:1022967814992.

[44] Cleve B. Moler. "MATLAB: A Mathematical Visualization Laboratory". In: *COMPCON'88, Digest of Papers, Thirty-Third IEEE Computer Society International Conference, San Francisco, California, USA, February 29 - March 4, 1988*. IEEE Computer Society, 1988, pp. 480–481. ISBN: 0-8186-0828-5. DOI: 10.1109/CMPCON.1988.4915. URL: https://doi.org/10.1109/CMPCON.1988.4915.

[45] Shunsuke Ohashi, Giovanni Yoko Kristianto, Goran Topic, and Akiko Aizawa. "Efficient Algorithm for Math Formula Semantic Search". In: *IEICE Transactions* 99-D.4 (2016), pp. 979–988. URL: http://search.ieice.org/bin/summary.php?id=e99-d_4_979.

[46] Frank W. Olver, Daniel W. Lozier, Ronald F. Boisvert, and Charles W. Clark. *NIST Handbook of Mathematical Functions*. 1st. New York, NY, USA: Cambridge University Press, Apr. 30, 2010. 968 pp. ISBN: 9780521140638. URL: http://www.ebook.de/de/product/10464908/nist_handbook_of_mathematical_functions_paperback_and_cd_rom.html.

[47] Robert Pagel and Moritz Schubotz. "Mathematical Language Processing Project". In: *Joint Proceedings of the MathUI, OpenMath and ThEdu Workshops and Work in Progress track at CICM co-located with Conferences on Intelligent Computer Mathematics (CICM 2014), Coimbra, Portugal, July 7-11, 2014*. Ed. by Matthew England, James H. Davenport, Andrea Kohlhase, et al. Vol. 1186. CEUR Workshop Proceedings. CEUR-WS.org, 2014. URL: http://ceur-ws.org/Vol-1186/paper-23.pdf.

[48] W. W. Peterson and D. T. Brown. "Cyclic Codes for Error Detection". In: *Proceedings of the IRE* 49.1 (Jan. 1961), pp. 228–235. ISSN: 0096-8390. DOI: 10.1109/JRPROC.1961.287814.

[49] H. A. Priestley. *Introduction to Complex Analysis*. Oxford University Press, Aug. 28, 2003. 344 pp. ISBN: 0198525621.

[50] Wolfram Research. *Computable Document Format (CDF)*. http://www.wolfram.com/cdf/. July 2011.

[51] Wolfram Research. *Generating and Importing TeX*. https://reference.wolfram.com/language/tutorial/GeneratingAndImportingTeX.html.

[52] Moritz Schubotz. "Augmenting Mathematical Formulae for More Effective Querying & Efficient Presentation". PhD thesis. Technical University of Berlin, Germany, 2017. ISBN: 978-3-7450-6208-3. URL: http://d-nb.info/1135201722.

[53] Moritz Schubotz, Alexey Grigorev, Marcus Leich, Howard S. Cohl, Norman Meuschke, Bela Gipp, Abdou S. Youssef, and Volker Markl. "Semantification of Identifiers in Mathematics for Better Math Information Retrieval". In: *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, July 17-21, 2016*. Ed. by Raffaele Perego, Fabrizio Sebastiani, Javed A. Aslam, Ian Ruthven, and Justin Zobel. ACM, 2016, pp. 135–144. ISBN: 978-1-4503-4069-4. DOI: 10.1145/2911451.2911503. URL: http://doi.acm.org/10.1145/2911451.2911503.

[54] Moritz Schubotz, Leonard Krämer, Norman Meuschke, Felix Hamborg, and Bela Gipp. "Evaluating and Improving the Extraction of Mathematical Identifier Definitions". In: *Proc. Conference and Labs of the Evaluation Forum (CLEF)*. 2017.

[55] Moritz Schubotz, Norman Meuschke, Thomas Hepp, Howard S. Cohl, and Bela Gipp. "VMEXT: A Visualization Tool for Mathematical Expression Trees". In: *Intelligent Computer Mathematics - 10th International Conference, CICM 2017, Edinburgh, UK, July 17-21, 2017, Proceedings*. Ed. by Herman Geuvers, Matthew England, Osman Hasan, Florian Rabe, and Olaf Teschke. Vol. 10383. Lecture Notes in Computer Science. Springer, 2017, pp. 340–355. ISBN: 978-3-319-62074-9. DOI: 10.1007/978-3-319-62075-6_24. URL: https://doi.org/10.1007/978-3-319-62075-6_24.

[56] Daniel Shanks. *Solved and Unsolved Problems in Number Theory*. 3rd ed. New York, USA: Chelsea Publishing Co., Inc., 1985. ISBN: 0-8284-1297-9.

[57] William Stein. *Sage Math*. http://doc.sagemath.org/html/en/tutorial/latex.html. Seen 6/2017.

[58] H. Strubbe. "Presentation of the SCHOONSCHIP system". In: *ACM SIGSAM Bulletin* 8.3 (1974), pp. 55–60. DOI: 10.1145/1086837.1086845. URL: http://doi.acm.org/10.1145/1086837.1086845.

[59] Michael Trott and Eric Wolfgang Weisstein. *Computational Knowledge of Continued Fractions*. http://blog.wolframalpha.com/2013/05/16/computational-knowledge-of-continued-fractions. seen 3/2017. May 2013.

[60] Michael Trott, Eric Weisstein, Oleg Marichev, and Todd Rowland. *The eCF-Project – Final Report*. Tech. rep. Wolfram Foundation, Oct. 2013. URL: http://www.wolframfoundation.org/programs/FinalReport.pdf.

[61] Stephen Wolfram. *Mathematica - a system for doing mathematics by computer*. Addison-Wesley, 1988. ISBN: 0201193302. URL: http://www.worldcat.org/oclc/16830839.

[62] Magdalena Wolska and Mihai Grigore. "Symbol Declarations in Mathematical Writing". In: *Towards a Digital Mathematics Library. Paris, France, July 7-8th, 2010*. Brno, Czech Republic: Masaryk University Press, 2010, pp. 119–127. URL: http://eudml.org/doc/221086.

[63] Abdou Youssef. "Part-of-Math Tagging and Applications". In: *Intelligent Computer Mathematics - 10th International Conference, CICM 2017, Edinburgh, UK, July 17-21,*

2017, Proceedings. Ed. by Herman Geuvers, Matthew England, Osman Hasan, Florian Rabe, and Olaf Teschke. Vol. 10383. Lecture Notes in Computer Science. Springer, 2017, pp. 356–374. ISBN: 978-3-319-62074-9. DOI: 10.1007/978-3-319-62075-6_25. URL: https://doi.org/10.1007/978-3-319-62075-6_25.