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

Anonymous Authors

Abstract

Purpose: Modern mathematicians and scientists of math-related disciplines use Word Processor (WP) to write and Computer Algebra System (CAS) to calculate mathematical expressions. Usually, they translate the expressions manually between WP and CAS. This process is time-consuming and error-prone. Our goal is to automate this translation. This paper uses Maple and Mathematica as the CAS, and LaTeX as our WP.
Approach: The National Institute of Standards and Technology (NIST) developed a collection of special LaTeX macros that create links from mathematical symbols to their definitions in the NIST Digital Library of Mathematical Functions (DLMF). We are using these macros to perform rule-based translations between the formulae in the DLMF and CAS. Moreover, we develop software to ease the creation of new rules and to discover inconsistencies.
Findings: We created 396 mappings and translated 58.8% of the DLMF (2,405 expressions) successfully between Maple and DLMF. For a significant percentage, the definitions in Maple and the DLMF were different. Consequently, an atomic symbol in one system maps to a composite expression in the other system. The translator was also successfully used for automatic verification of mathematical online compendia and CAS. Our evaluation techniques discovered two errors in the DLMF and one defect in Maple.
Originality: This paper introduces the first translation tool for special functions between LaTeX and CAS. The approach improves error-prone manual translations and can be used to verify mathematical online compendia and CAS.

Keywords: LaTeX, Computer Algebra System (CAS), Translation, Presentation to Computation (P2C), Special Functions

1 Introduction

A typical workflow of a scientist who writes a scientific publication is to use a Word Processor (WP) to write the paper and one or more Computer Algebra System (CAS) for verification, analysis and visualization. Especially in the Science, Technology, Engineering and Mathematics (STEM) literature, LaTeX[1] has become the de facto standard for writing scientific publications

[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 de facto standard for writing articles in STEM,

over the past 30 years (Knuth, 1997; Knuth, 1998, p. 559; Alex, 2007). LaTeX enables printing of mathematical formulae in a structure similar to handwritten style. For example, consider the specific Jacobi polynomial (*DLMF*, Table 18.3.1)

$$P_n^{(\alpha,\beta)}(\cos(a\Theta)), \tag{1}$$

where n is a nonnegative integer, $\alpha, \beta > -1$, and $a, \Theta \in \mathbf{R}$. This mathematical expression is written in LaTeX as

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

While LaTeX focuses on displaying mathematics, a CAS concentrates on computations and user friendly syntax. Especially important for a CAS is to embed unambiguous semantic information within the input. Each system uses different representations and syntax in consequence. Hence, a writer needs to constantly translate mathematical expressions from one representation to another and back again. Table 1 shows four different representations for (1).

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]]]`

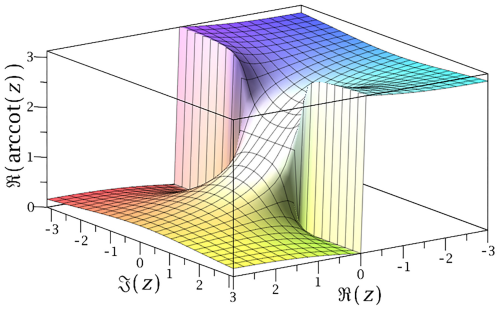
Table 1: Different representations for (1). Generic LaTeX is the default LaTeX expression; semantic LaTeX uses special semantic macros to embed semantic information; and CAS representations are unique to themselves.

Translations from generic LaTeX to CAS are difficult to realize since the full semantic information is not easily constructed from the input. Bruce Miller at the National Institute of Standards and Technology (NIST) has created a set of semantic LaTeX macros (Miller and Youssef, 2003). Each macro ties specific character sequences to a well-defined mathematical object and is linked with the corresponding definition in the Digital Library of Mathematical Functions (DLMF). The Digital Repository of Mathematical Formulae (DRMF) is an outgrowth of the DLMF with the goal to facilitate interaction among a community of mathematicians and scientists (Cohl, McClain, et al., 2014; Cohl, Schubotz, McClain, et al., 2015). The DRMF extends the set of semantic macros. These macros embed necessary semantic information into LaTeX expressions. One example of such a macro is given Table 1 for the semantic LaTeX representation for the Jacobi polynomial. The macros provide isolated access to important parts of the mathematical function, such as the arguments.

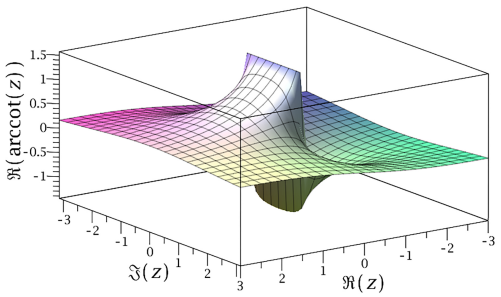
Even with embedded semantic information, a translation between systems can be difficult. A typical example of complex problems occur for multivalued functions (Davenport, 2010). A CAS usually defines *branch cuts* to compute principal values of multivalued functions (England et al., 2014), which makes the implementation of a theoretically continued function to a

and we only focus on mathematical writing in this paper, we have categorized LaTeX as a WP as well.

discontinuous presentation of it. In general, positioning branch cuts follows conventions, but can be positioned arbitrarily in many cases. Communicating and explaining the decision of defined branch cuts is a critical issue for CAS and can vary between various systems (Corless et al., 2000). Figure 1 illustrates two examples of different branch cut positioning for the inverse trigonometric arccotangent function. While Maple[2] defines the branch cut at $[-i\infty, -i]$, $[i, i\infty]$ (Figure 1a), Mathematica defines the branch cut at $[-i, i]$ (Figure 1b).



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



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

Figure 1: Two plots for the real part of the arccotangent function with a branch cut at $[-i\infty, -i]$, $[i, i\infty]$ 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 (England et al., 2014). 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 (Youssef, 2017).

This article presents a new approach for automatic P2C and vice versa conversions. Translations from presentational to computational (computational to presentational) systems are called forward (backward) translations. A forward translation is denoted with an arrow with the target system language above the arrow. For example,

$$t \overset{\mathfrak{M}_{aple}}{\mapsto} c,$$

where t is an expression in the LaTeX language and c is an element of the Maple language $\mathfrak{M}_{aple}$. As we will see later in this article, we need to compare mathematical concepts between systems. This is impossible from a mathematical point of view. Consider the irrational mathematical constant e, known as Euler's number. The theoretical construct for this symbol cannot be mathematically equivalent to the value `exp(0)` in Maple, caused by computational and implementational limitations. Instead of using the term *equivalent*, we introduce a *appropriate* and *inappropriate*

[2]The mention of specific products, trademarks, or brand names is for purposes of identification only. Such mention is not to be interpreted in any way as an endorsement or certification of such products or brands by the National Institute of Standards and Technology, nor does it imply that the products so identified are necessarily the best available for the purpose. All trademarks mentioned herein belong to their respective owners.

translations. We call a translation such as

$$\texttt{\textbackslash cos@\{z\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{cos(z)} \tag{2}$$

as *appropriate*, while a translation such as

$$\texttt{\textbackslash cos@\{z\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{sin(z)} \tag{3}$$

as *inappropriate*. Note that it is not always as easy as in this example to decide if a translation is appropriate or not. Therefore, this article also presents several validation techniques to automatically verify if a translation is appropriate or inappropriate. In addition to this terminology, we introduce *direct translations*. Later in the paper, we will explain that a translation from one specific mathematical object to its *appropriate* counterpart in the other system is not always possible. We call a translation to the *appropriate* counterpart *direct*. For example, the translation (2) is *direct*, while a translation to the definition of the cosine function

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

is not a *direct* translation. Note that partial results of this paper has been published in (Cohl, Schubotz, Youssef, et al., 2017).

2 Related Work

Since LaTeX became the de facto standard for writing papers in mathematics, most of the CAS provide simple functions to import and export mathematical LaTeX expressions[3]. Those tools have two essential problems. 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. However, for more complex expressions, e.g., the Jacobi polynomial in Table 1, the import functions fail. 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 sufficient semantic information. In consequence of these two problems, an exported expression cannot be imported to the same system again in most cases (except for simple expressions such as those described above). Our tool should solve these problems and provide round-trip translations between LaTeX and CAS.

The semantics must be well known before an expression can be translated. There are two main approaches to solve that problem: (1) someone could specify the semantic information during

[3]The selected CAS Maple, Mathematica, Matlab, and SageMath provide import and/or export functions for LaTeX: Maple, `http://www.maplesoft.com/support/help/Maple/view.aspx?path=latex` seen 06/2017; Mathematica, `https://reference.wolfram.com/language/tutorial/GeneratingAndImportingTeX.html` seen 06/2017; Matlab, `https://www.mathworks.com/help/symbolic/latex.html` seen 06/2017; SageMath, `http://doc.sagemath.org/html/en/tutorial/latex.html` seen 06/2017.

the writing process (pre-defined semantics); and (2) the translator can find out the right semantic information in general mathematical expressions before it translates the expression. So-called *interactive documents*[4], such as the Computable Document Format (CDF)[5] by Wolfram Research, or *worksheets* by Maple, try to solve this problem with the approach (2) and allow one to embed semantic information into the input. 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 (Giceva, Lange, and Rabe, 2009) is able to create web-based interactive documents and uses Open Mathematical Documents (OMDoc) (Kohlhase, 2006) to carry semantics. The documents can be viewed and edited in the browser. Those JOBAD-documents also allow one to perform computations via CAS. This gives one the opportunity to calculate, compute and change mathematical expressions directly in the document. The translation performs in the background, invisible to the user. Similar to the JOBAD architecture, other interactive web documents exist, such as *MathDox* (Cuypers et al., 2008) and *The Planetary System* (Kohlhase et al., 2011). All of these demonstrate the potential of the educational system.

Another approach tries to avoid translation problems by allowing computations directly via the LaTeX compiler, e.g., *LaTeXCalc* (Churchill and Boyd, 2010). Those packages are limited to the abilities of the compiler and therefore are not as powerful as CAS. A workaround for this case is *sagetex* (Drake, 2009), which is a LaTeX package interface for the open source CAS *sage*[6]. This package allows *sage* commands in TeX-files and uses *sage* in the background to compute the commands. In this scenario, a writer still needs to manually translate expressions to the syntax of *sage*.

There exist two approaches for marking up mathematical TeX/LaTeX documents semantically with TeX macros. Namely, sTeX (Kohlhase, 2008) developed by Kohlhase and the DLMF/DRMF LaTeX macros developed by Miller (Miller and Youssef, 2003). This paper shows that it is possible to develop a context-free translation tool using the semantic macros introduced by these two projects. The goal of sTeX was to markup the functional structure of mathematical documents so that they can be exported to the OMDoc format. The macro set developed by Miller introduces new macros for special functions, orthogonal polynomials, and mathematical constants. Each of these macros 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 and the DRMF. We gave the DLMF/DRMF LaTeX macro set the favor for developing the translation engine because it provides DLMF definitions for a comprehensive amount of functions. In contrast, sTeX does not focus on the semantics of functions, is too complex to use, and defines diverse macros for symbols and concepts that CAS usually does not support.

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

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

[6]An abbreviation for *SageMath*.

3 Translation Problems

There are several potential problems for performing translations between systems that embed semantic information in the input. These problems vary from simple cases, e.g., a function is not defined in the system, to complex cases, e.g., different positioning of branch cuts for multivalued functions. This section will discuss the problems and our workarounds.

If a function is defined in one system but not in the other, we can translate the definition of the mathematical function. For example, the *Gudermannian* (*DLMF*, (4.23.10)) $\mathrm{gd}(x)$ function is defined by

$$\mathrm{gd}(x) := \arctan(\sinh x), \quad x \in \mathbb{R}, \tag{4}$$

and linked to the semantic macro `\Gudermannian` in the DLMF but does not exist in Maple. We can perform a translation for the definition (4) instead of macro itself

$$\texttt{\textbackslash Gudermannian\{x\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{arctan(sinh(x))}. \tag{5}$$

Since translations such as these are nonintuitive, describing explanations become necessary for the translation process. A special logging function takes care of each translation and provide details after a successful translation process. Section 5 explains this task further.

Providing detailed information also solves the problem for multiple alternative translations. In some cases, a semantic macro has two alternative representations in the CAS or vice versa. In such cases, the translator picks one of the alternatives and informs the user about the decision.

In case of differences between defined branch cuts we can also use alternative translations to solve the problems. Consider the mentioned case of the arccotangent function (Corless et al., 2000) that has different positioned branch cuts in Maple compared to the DLMF or Mathematica definitions. Alternative but mathematically equivalent translations are

$$\texttt{\textbackslash acot@\{z\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{arccot(z)}, \tag{6}$$

$$\overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{arctan(1/z)}, \tag{7}$$

$$\overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{I/2*ln((z-I)/(z+I))}. \tag{8}$$

The position of the branch cut of the arccotangent function differs after translation (6), which may lead to incorrect calculations later on. The alternative translations (7) and (8) use other functions instead of the arccotangent function. The arctangent function (7) and the natural logarithm (8) have the same positioned branch cuts as in the DLMF and in Maple. In consequence, translation (7) solves the issue, as long as the user do not evaulate the function at $z = 0$, while translation (8) solves the issue except at $z = -i$.

Other problematic cases for translations are the DLMF/DRMF LaTeX macros themselves. In some cases, they do not provide sufficient semantic information to perform translations. One example is the *Wronskian* determinant. For two differentiable functions, the *Wronskian* is defined as (*DLMF*, (1.13.4))

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

In semantic LaTeX it is currently implemented using

```
\Wronskian@{w_1(z), w_2(z)}.
```
(9)

A translation become unfeasible because the macro does not explicitely define the variable of differentiation for the functions w_1 and w_2. For a correct translation, the CAS needs to be aware of the used variable z. We solved this issue by creating a new macro, e.g.,

```
\Wron{z}@{w_1(z)}{w_2(z)}.
```
(10)

This example demonstrates that the DLMF/DRMF LaTeX macros are still a work in progress and are getting constantly updated.

A similar problem is multiplications since they are rarely explicitly marked in LaTeX expressions, e.g., scientists using whitespaces to indicate multiplications rather than using `\cdot` or similar symbols. For such problems, we introduced a new macro `\idot` for an invisible multiplication symbol (this macro will not be rendered). Since this macro is newly introduced by contributers of the DRMF team, and automatic conversion of existing equations is difficult, none of the equations in the DLMF use this macro. As a consequence, the translator has some simple rules to perform translations without explicitly marked translations with `\idot`.

The DLMF/DRMF LaTeX macros do not garantee entirely disambiguated expressions. In Table 2 there are four examples of potentially ambiguous expressions. These expressions are unambiguous for the LaTeX compiler since it only considers the very next token for superscripts and subscripts. Our translator follows the same rules to solve these issues.

Potentially 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 2: Potentially ambiguous LaTeX expressions and how LaTeX displays them.

Another more questionable translation decision are alphanumerical expressions. As explained in Table 6, the Part-of-Math (PoM)-tagger 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 PoM-Parsed Tree (PPT). The translator interprets alphanumerical expressions as multiplications for two reasons: (1) we would assume that the inputs '4b' and 'b4' are mathematically equivalent; and (2) it is more common in mathematics to use single letter names for variables (Cajori, 1994). Therefore we define the following definitions

$$\texttt{4b} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{4*b},$$
$$\texttt{b4} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{b*4},$$
$$\texttt{energy} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{e*n*e*r*g*y}.$$

In general, the translator is designed to find a work-around for disambiguating expressions. Only in case there is no way to solve the ambiguity with the defined rules, the translation process stops.

4 The Translator

All translations are defined by a library (Comma-Separated Values (CSV) and JavaScript Object Notation (JSON) files) that defines translation patterns for each function and symbol. The pattern uses `$i` as placeholders to define the positions of the arguments. For example, the translation patterns for the Jacobi polynomial are illustrated in Table 3.

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: Forward and backward translation patterns for the Jacobi polynomial example (1) in this manuscript. The pattern for the backward translation is the same for Maple and Mathematica.

These placeholders causes trouble when the CAS uses the symbol `$` for other reasons, e.g., the differentiation in Maple is defined by

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

where f is an algebraic expression or an equation, x is the name of the differentiation variable, and n is an integer representing the n-th order differentiation[7]. A translation for $\frac{d^2x^2}{dx^2}$ should look like this

$$\texttt{\deriv[2]\{x\^2\}\{x\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{diff(x\^2, [x\$2])}$$

but would end up as

$$\texttt{\deriv[2]\{x\^2\}\{x\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{diff(x\^2, [xx])}.$$

We can solve this issue by using parentheses in such cases, e.g., `diff($1, [$2$($0)])`.

[7] `https://www.maplesoft.com/support/help/maple/view.aspx?path=diff`, seen 07/2018

The DLMF/DRMF LaTeX macros also allow one to specify optional arguments to distinguish between standard and another version of these functions. The Legendre and associated Legendre function of the first kind are examples of such cases. The library that defines translations for each macro uses the macro name as the primary key to identify the translations. The Legendre and associated Legendre function of the first kind both use the same macro `\LegendreP`. To distinguish such cases, we use a special syntax for such cases, shown in Table 4.

Semantic Macro Entry	Maple Entry
`\LegendreP{\nu}@{x}`	`LegendreP($0, $1)`
`X1:\LegendrePX\LegendreP[\mu]{\nu}@{x}`	`LegendreP($1, $0, $2)`

Table 4: Example entries of the Legendre and associated Legendre function in the translation library. The prefix notation `X<d>:<name>X` defines the translation for `<name>` with `<d>`-number of optional arguments.

The translator uses the PoM-Tagger (Youssef, 2017)[8] to parse LaTeX expressions into a parsed tree. The PoM-Tagger is an LL-Parser defined by a context-free grammar in Backus-Naur Form (BNF). Each token will be tagged by meta information defined in lexicon files. We extend the lexicon files to provide also the information that is necessary for the translation process. An example of an entry of the lexicon file is given in Table 5.

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
Mathematica: `Sin[$0]`
Mathematica-Link: reference.wolfram.com/ language/ref/Sin.html

Table 5: The entry of the trigonometric sine function in the lexicon file.

[8] Named according to the Part-of-Speech-Taggers in Natural Language Processing (NLP).

5 Forward Translations

An abstract translator class analyzes each node of the parsed tree and delegates them to specialized subtranslators. In this process, an object called Translated Expression Object (TEO) is built. This TEO is needed to rebuild a string representation of the mathematical expression after all nodes of the parsed tree were translated.

The parsed tree generated by the PoM-tagger is not a mathematical expression tree. The PoM project aims to disambiguate mathematical LaTeX expressions and generates an expression tree. However, in the current state, many expressions cannot be disambiguated yet. In consequence, the PoM-tagger generates a raw parsed tree where each token in the LaTeX expression is a node in the tree. We call this parsed tree, the PPT.

The overall forward translation process is explained in Figure 2. All translation patterns and related information are stored in the DLMF/DRMF tables. These tables are converted by the `lexicon-creator` to the `DLMF-macros-lexicon` lexicon file. Together with the `global-lexicon` file, the PPT will be created by the PoM-tagger. The `latex-converter` takes a string representation of a semantic LaTeX expression and uses the PoM engine as well as our `Translator` to create an appropriate string representation for a specified CAS.

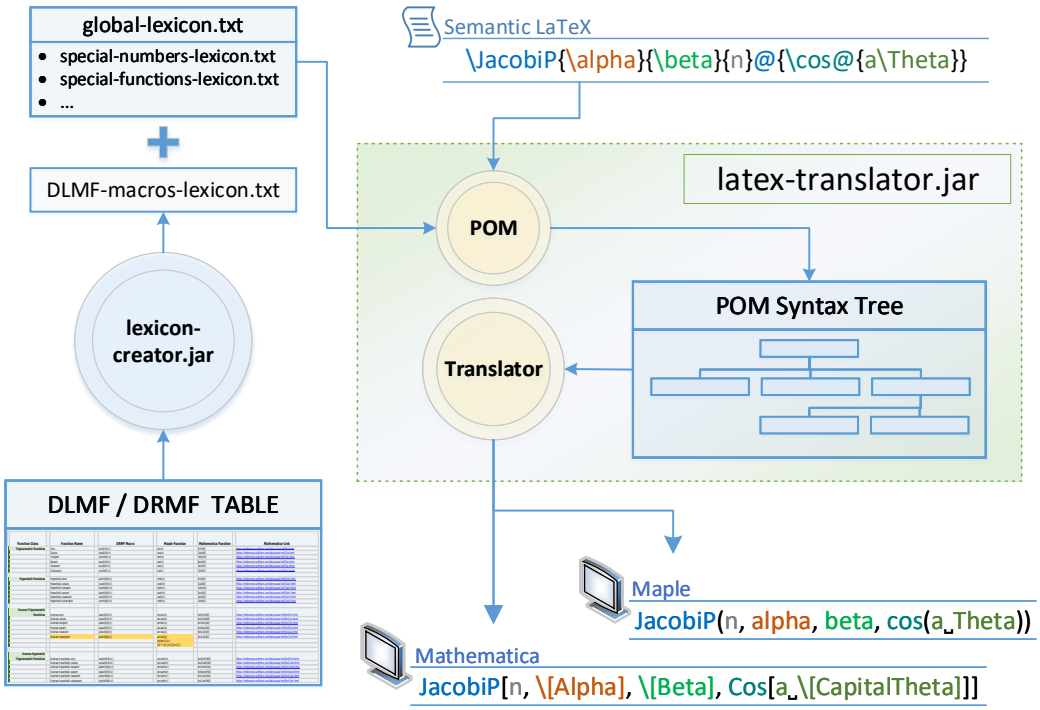


Figure 2: Process diagram of a forward translation process. The PoM-tagger generates the PPT based on lexicon and JSON files. The PPT will be translated to different CAS.

5.1 *Analyzing the PoM-Parsed Tree*

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 PPT.

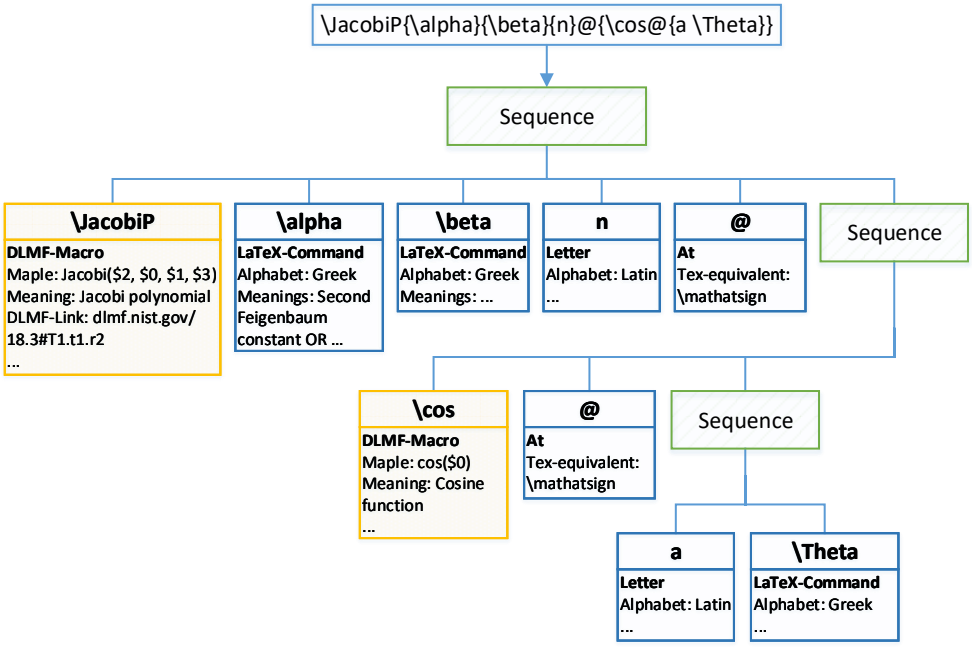


Figure 3: The PPT for the Jacobi polynomial example (1) using the DLMF/DRMF LaTeX macro. Each leaf contains information from the lexicon files.

Figure 3 visualizes the PPT of the Jacobi polynomial example from Table 1. Because of the differences to expression trees, a backward conversion of the PPT to a string representation can be difficult, especially for finding necessary or unnecessary parentheses. Therefore we create the TEO.

With these tools, we can translate a LaTeX expression by translating the PPT node by node and perform group or reordering operations for some special cases. The algorithm is realized in a simple recursive structure. The Algorithm 1 explains the translation process. Whenever the algorithm finds a leaf, it can translate this single term. If the node is not a leaf, it

Algorithm 1 Simple translation algorithm for the POM-Parse Tree

Input: Root r of the parse tree T

```
procedure TRANSLATE(r)
    if r is leaf then
        TRANSLATE_LEAF(r);
    else
        for all children v_n of r do
            TRANSLATE(v_n);
        end for
    end if
end procedure
```

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

Input: Root r of a POM-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);
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
```

starts to translate all children of the node recursively.

This approach is not completely feasible since the algorithm needs to look ahead and check the following siblings in some cases, e.g., in the case of a semantic macro with arguments (see Figure 3). The `TRANSLATE_LEAF` function needs to know the following siblings of the current node. Algorithm 2 explains a more detailed but abstract version of the final algorithm of the presented translation tool.

If the root r is a leaf, it still can be translated as a leaf. Eventually, some of the following siblings are needed to translate r. The list of *following_siblings* in Line 3 might be reduced to avoid multiple translations for one node. 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 next node. Line 8 checks if there are following siblings left and calls the `ABSTRACT_TRANSLATOR` recursively in such cases. Translated expressions are stored by the TEO object. Algorithm 2 is a simplified version of the translator process. The Lines 3 and 6 process the translations for each node. Table 6 gives an overview of all the different node types the root r can be. A more detailed explanation of the types can be found in (Youssef, 2017).

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 delegate the translation to specialized subtranslators. This delegation process is implemented in Lines 3 and 6 of Algorithm 2. Subsection 5.2 discusses these classes in more detail.

This approach seems to work for most semantic LaTeX expressions. However, there is one problem with this straightforward approach which was mentioned shortly mentioned in Section 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 6: A table of all kinds of nodes in a PoM syntax tree. Note that this table groups some kinds for a better overview. For a complete list and a more detailed version see (Youssef, 2017).

There are many different types of notations used to represent formulae. Figure 4 illustrates the simple expression $(a+b)x$ in different notations. The Normal Polish Notation (NPN)[9] (hereafter called prefix notation) places the operator to the left of/before its operands. The Reverse Polish Notation (RPN)[10] (hereafter called postfix notation) does the opposite and places the operator

[9] Also known as *Warsaw Notation* or *prefix notation*

[10] Also known as *postfix notation*

to the right of/after its operands. The infix notation is commonly used in arithmetic and places the operator between their operands. This only makes sense as long as the operator is a binary operator.

In mathematical expressions, notations are mostly mixed, depending on the case and number of operands. For example, infix notation is common for binary operators (+, −, ·, mod , etc.), while functional notations are conveniently used for any kind of functions (sin, cos, etc.). Sometimes the same symbol is used in different notations to distinguish different meanings. For example, the '−' as a 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 commonly used in infix notation. An example for the postfix notation is factorial, such as '2!'.

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

Figure 4: The mathematical expression '$(a+b)\cdot x$' in infix, prefix, postfix and functional notation.

Most programming languages (and CAS as well) internally use prefix or postfix notation and do not mix the notations in one expression, since it is more convenient to parse expressions in uniform notations. However, the common practice in science is to use mixed notations in expressions. Since the PoM has rarely implemented mathematical grammatical rules yet, it takes the input as it is and does not build an expression tree. Therefore, it parses all four examples from Figure 4 to four different PPTs rather than to one unique expression tree. In general, this is not a problem for our translation process since most CAS are familiar with most common notations. Therefore, the translator does not need to know that 'a' and 'b' are the operands of the binary operator '+' in '$a + b$.' The translator could simply 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 CAS which use non-standard notations.

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 translator would fail in its current state. Supporting CAS with another notation would be a part of future work.

Nonetheless, adopting different notations, in some situations, could also solve potential ambiguities. Consider the two potentially ambiguous examples in Table 7. 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 sequences 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 7: Potentially 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 7 to clarify the different interpretations in Example 2. Note that the use of parentheses will not always resolve 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. We define the translations as follows:

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

Algorithm 2 does not allow this translation right now. It has no access to previously translated nodes in its current state. This problem is solved by the TEO that stores and groups translated objects like lists. This allows one to access the latest translated expression and use it as the argument for the factorial function. Table 8 shows three example for the TEO list that groups some tokens.

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

Table 8: How the TEO-list groups subexpressions.

5.2 Subtranslators

A `SequenceTranslator` translates the *sequence* and *balanced expressions* in the PPT. If a node n is a leaf and the represented symbol is 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 balancing 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 half-opened, half-closed intervals. Whether an expression is an interval or another expression is difficult to decide and can depend on the context. Also, the parentheses checker could simply be deactivated to allow mismatched parentheses in an expression. Another option is to use of interval macros. e.g., `\intcc@{a}{b} = [a,b]`.

The `SequenceTranslator` also handles positions of multiplication symbols. There are a couple of obvious choices to translate multiplication. The most common symbol for multiplications is still the white space (or no space between the tokens), as explained previously. Consider the simple expression '$2n\pi$'. The PPT generates a sequence node with three children, namely 2, n and π. This sequence should be interpreted as a multiplication of the three elements. The `SequenceTranslator` checks the types of the current and next nodes in the tree to decide if it should add 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)` to distinguish between f as a function call and f as a symbol.

Algorithm 3 explains the `MacroTranslator` without error handling. It has extracted necessary information from the PPT, such as how many arguments this function has. In consequence, it also processes the following siblings to translate the arguments. There are some special cases about the next node right after the macro occurs. This node can be

- an exponent, such as for '`^2`';
- an optional parameter in square brackets;
- a parameter in curly brackets (a *sequence* node in the PPT) if none of the above;
- '@' symbols if none of the above; or
- 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 one usually writes the exponent after the arguments of a function in CAS. Hence, 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 starts a balanced sequence of 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) );        ▷ number is given in info.
15:     SKIP_AT_SIGNS( args, info );                            ▷ number is given in info.
16:     argList.add( TRANSLATE_VARIABLES(args, info) );         ▷ number is given in 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 9 shows an example for the trigonometric cosine function with multiple exponents.

As you can see, the input expression in semantic LaTeX is interpreted as

$$\left(\cos(x)^2\right)^2. \tag{11}$$

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

Table 9: A trigonometric cosine function example with exponents before and after the argument.

6 Maple to Semantic LaTeX Translator

Instead of writing a custom Maple syntax parser, we using Maple's internal data structure to get a syntax tree of the input[11]. Maple allows several different input styles. The `1D` input is mainly used for programming purposes and therefore also used to perform our translations. Internally, Maple uses a Directed Acyclic Graph (DAG) for syntax trees.

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 5 illustrates the internal DAG representation with headers and arguments.

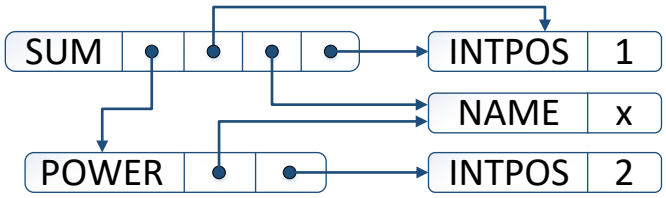


Figure 5: The internal Maple DAG representation of x^2+x.

One can access the internal data structure of expressions via the `ToInert` command, which returns the `InertForm`. The `InertForm` format is a nested list[12] of the internal DAG for the given expression. Some of the important types for the nodes are specified in Table 10. The translator uses the `OpenMaple` (Bernardin et al., 2016, §14.3) Application Programming Interface (API) for interacting with Maple's kernel implementation.

6.1 Automatic Changes of Inputs in Maple

Maple evaluates inputs automatically and changes the input into an internal representation. This internal representation might look a bit different to the input. One example has already been given with Figure 5, where each summand of a sum is stored with a factor. Here is a list of all internal changes that occur for inputs.

- Maple evaluates input expressions immediately.
- There is no data type to represent 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 $x^{\frac{1}{2}}$.

[11] In consequence, a license of Maple is mandatory to perform backward translations. The translator is using the version Maple 2016.

[12] The nested list is a tree representation of a DAG that splits nodes with multiple parents into multiple nodes so that each node has only one parent node.

Type	Explanation
SUM	Sums. Internally stored with factors for each summand, i.e., 'x+y' would be stored as '$x \cdot 1 + y \cdot 1$'.
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 10: A subset of important internal Maple data types. See (Bernardin et al., 2016) for a complete list.

- 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 using 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 an integer numerator and a positive integer 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 11 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 11: Example of unevaluation quotes for `1D` Maple input expressions.

Since we want to keep a translated expression similar to the input expression, we implemented some cosmetic rules for the backward translations that solve or reduce the effects from the list of changes above.

- We use unevaluation quotes to suppress evaluations of the input.
- We perform a reordering of factors and summands so that negative factors appear in front

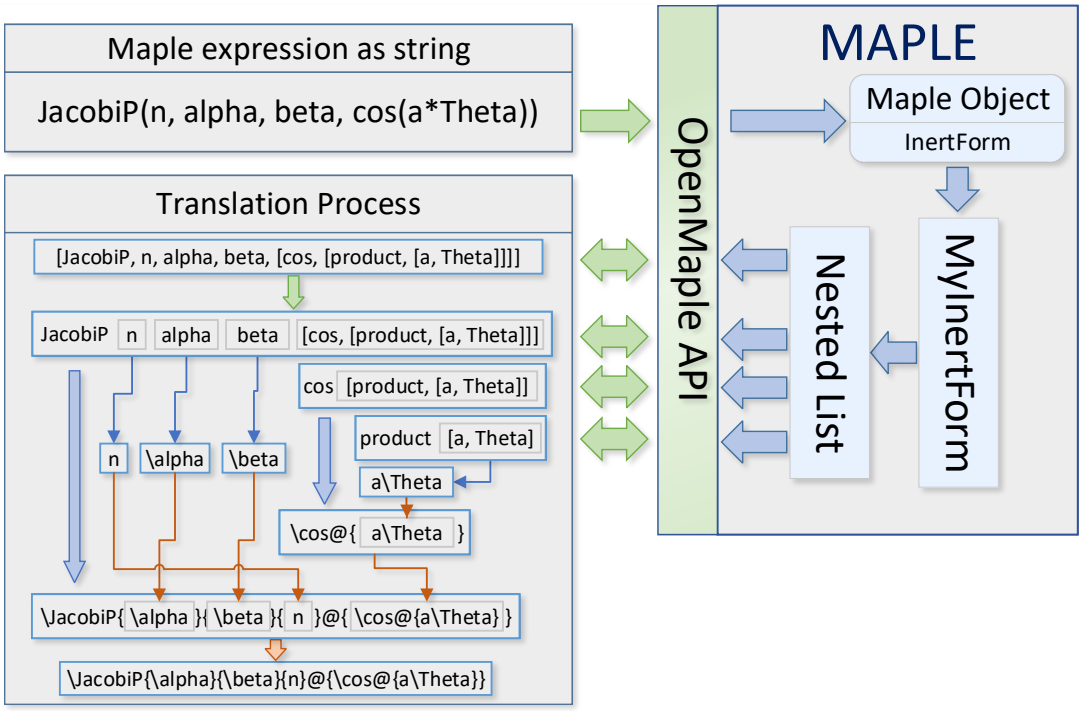


Figure 6: A scheme of the backward translation process from Maple for the Jacobi polynomial expression $P_n^{(\alpha,\beta)}(\cos(a\Theta))$. The input string is converted by the Maple kernel into the nested list representation. This list is translated by subtranslators (blue and red arrows). A function translation (bold blue arrows) is again realized using translation patterns to define the position of the arguments (red arrows).

of the summand. This gives us the opportunity to translate $x-y$ to $x-y$ instead of $x+y\cdot(-1)$.

- We introduced new internal data types `MYFLOAT` and `DIVIDE` to translate floats and fractions in more convenient notations.

The translation process then follows the same principle as for the forward translations. Since the syntax tree of Maple is an expression tree, we do not need to implement special reordering or grouping algorithms to perform backward translations. Translations for functions are also realized via patterns and placeholders. Figure 6 illustrates the backward translation process for the Jacobi polynomial example from Table 1.

7 Evaluation

We implemented three approaches to evaluate whether a translation was *appropriate* or *inappropriate*.

1. **Round Trip Tests**: translates expressions back and forth and analyzing the changes.
2. **Function Relation Tests (Symbolical)**: translates mathematically proven equivalent expressions from one system to a CAS and evaluate whether the relation remains valid via symbolical equivalence checks.
3. **Numerical Tests**: take the same equations from Approach 2 but evaluates them on specific numerical values to test equivalence.

7.1 Round Trip Tests

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**. 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.

Table 12 illustrates an example of a round trip test which reaches a fixed point. The test formula is

$$\frac{\cos\left(a\Theta\right)}{2}. \tag{12}$$

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 12: A round trip test reaching a fixed point.

Step 4 is identical to step 2, and since the translator is a deterministic algorithm, it can be easily shown that step 2 and step 3 are fixed-point representations for semantic LaTeX and Maple.

There is currently only one exception known where a round trip test does not reach a fixed point representation: Legendre's incomplete elliptic integrals (*DLMF*, (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

$$\texttt{\textbackslash EllIntF@\{\textbackslash phi\}\{k\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{EllipticF(sin(phi),k)}, \tag{13}$$

$$\texttt{\textbackslash EllIntF@\{\textbackslash asin@\{\textbackslash phi\}\}\{k\}} \overset{\mathfrak{M}_{aple}}{\mapsfrom} \texttt{EllipticF(phi,k)}. \tag{14}$$

The round-trip translations produce infinite chains of sine and inverse sine calls because there are no evaluations involved.

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. Consider we mistakenly defined an *inappropriate* forward and backward translation for the sine function

$$\texttt{\textbackslash sin@\{\textbackslash phi\}} \overset{\mathfrak{M}_{aple}}{\leftrightarrow} \texttt{cos(phi)}, \tag{15}$$

$$\texttt{\textbackslash cos@\{\textbackslash phi\}} \overset{\mathfrak{M}_{aple}}{\leftrightarrow} \texttt{sin(phi)}. \tag{16}$$

In that case the round trip test would not detect any errors but reaches a fixed point representation.

7.2 Function Relation Tests

The DLMF is a compendium for special functions and orthogonal polynomials and lists several relations between the functions and polynomials. The idea of this evaluation approach is to translate an entire relation and test whether the relation remains valid after performing the translations.

With this technique we can detect translation errors such as in (15) and (16). Consider the DLMF equation for the sine and cosine function (*DLMF*, (4.21.2))

$$\sin(u + v) = \sin u \cos v + \cos u \sin v. \tag{17}$$

Assume the translator would forward translate the expression based on (15, 16). Than

$$\texttt{\textbackslash sin@\{u + v\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{cos(u + v)}, \tag{18}$$

$$\texttt{\textbackslash sin@@\{u\}\textbackslash cos@@\{v\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{cos(u)*sin(v)}, \tag{19}$$

$$\texttt{\textbackslash cos@@\{u\}\textbackslash sin@@\{v\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{sin(u)*cos(v)}. \tag{20}$$

This produces the equation in Maple

$$\cos(u + v) = \cos u \sin v + \sin u \cos v, \tag{21}$$

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 often have difficulties testing simple equations symbolically. For example, equation (*DLMF*, (4.35.34))

$$\sinh(x + i \cdot y) = \sinh x \cos y + i \cosh x \sin y,$$

as a difference of the left- and right-hand side cannot be simplified to zero by default. 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. Because of implementation details, there are some techniques that helps Maple to find possible simplifications. For example we can force Maple to convert the formula

$$\sinh x + \sin x \tag{22}$$

to an equivalent representation using their exponential representations, namely

$$\frac{1}{2}e^{x} - \frac{1}{2}e^{-x} - \frac{1}{2}i\left(e^{ix} - e^{-ix}\right). \tag{23}$$

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 complex example (*DLMF*, (12.7.10))

$$U(0,z) = \sqrt{\frac{z}{2\pi}}K_{\frac{1}{4}}\left(\frac{1}{4}z^{2}\right), \tag{24}$$

where $U(0,z)$ is the parabolic cylinder function and $K_\nu(z)$ is the modified Bessel function of the second kind. Both functions are well-defined in both systems and we can define a *direct* translation for (24). The modified Bessel function of the second kind has its branch cut in Maple and in the DLMF at $z < 0$. However, the argument of K contains a z^2. If $|\mathrm{ph}(z)| \in \left(\frac{\pi}{2}, \pi\right)$ the value of the right-hand side of (24) would be no longer on the principal branch. However, Maple will still compute the principal values independently of the value of z. Hence, a translation

$$\texttt{\textbackslash BesselK\{\textbackslash frac\{1\}\{4\}\}@\{\textbackslash frac\{1\}\{4\}z\^{}2\}} \overset{\mathfrak{M}_{aple}}{\mapsto} \texttt{BesselK(1/4,(1/4)*z\^{}2)} \tag{25}$$

is incorrect if $|\mathrm{ph}(z)| \in \left(\frac{\pi}{2}, \pi\right)$ and one should instead use the analytic continuation for the right-hand side of (24). To evaluate such complex cases, the equivalence checks of CAS are insufficient. Therefore we implement numerical tests as an additional step.

7.3 Numerical Tests

Consider the differences of the left- and right-hand side of equation (24)

$$D(z) := U(0,z) - \sqrt{\frac{z}{2\pi}}K_{\frac{1}{4}}\left(\frac{1}{4}z^{2}\right). \tag{26}$$

Table 13 presents four computations for $D(z)$, one value for each quadrant in the complex plane.

Considering machine accuracy and the default precision to 10 significant digits, we can regard the first and last values as zero differences. While this evaluation is very powerful, it has a

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

Table 13: Four computations of $D(z)$ in Maple.

significant problem. Even when all tested values return zero, it does not prove the equivalence of (24). However, when the values are different from zero, it does indicate that there might be an error satisfying one of the four cases (Cohl, Greiner-Petter, and Schubotz, 2018):

1. the numerical engine tests invalid combinations of values;
2. the translation is incorrect;
3. there may be an error in the DLMF source; or
4. there may be an error in Maple.

7.4 Results

There are 685 DLMF/DRMF LaTeX macros[13] in total, and 665 of them were implemented in the translator engine. We defined forward translations to Maple for 201 of the macros and backward translations from Maple for 195 functions.

The DLMF provides a dataset of LaTeX expressions with semantic macros. We extracted 4087 equations from the DLMF and apply our round-trip and relation tests on them. The translator was able to translate 2405[14] (58.8%) of the extracted equations without errors. 660 (27.4%) of the successfully translated expressions were verified by the simplification techniques of Maple. We apply additional numerical tests for the remaining 1745 equations. For 418 (24%) cases the numerical tests were valid. More detailed results for numerical and symbolical tests were presented in (Cohl, Greiner-Petter, and Schubotz, 2018).

The evaluation techniques has proven to be very powerful for evaluating CAS and online mathematical compendia such as the DLMF. During the evaluations, we were able to detect several errors in the translation and evaluation engine, and also discovered two errors in the DLMF and one error in Maple's `simplify` function.

The numerical test engine was able to discover a sign error in equation (*DLMF*, (14.5.14))[15]

$$\mathsf{Q}_{\nu}^{-1/2}(\cos\theta)=-\left(\frac{\pi}{2\sin\theta}\right)^{1/2}\frac{\cos\left(\left(\nu+\frac{1}{2}\right)\theta\right)}{\nu+\frac{1}{2}}. \tag{27}$$

[13] The DLMF/DRMF semantic macros are still a work in progress, and the total number is constantly changing.
[14] All percentages are approximately calculated.
[15] The equation had originally been stated as shown in (27). The error was reported on 10th April 2017.

The error can be found on (Olver et al., 2010, p. 359) and has been fixed in the DLMF with version 1.0.16. The same engine also identified a missing comma in the constraint of (*DLMF*, (10.16.7)). The original constraint was given by $2\nu \neq -1, -2 - 3, \ldots$, with a missing comma after the -2.

We have also noticed that our testing procedure is able to identify errors in CAS procedures, namely the Maple `simplify` procedure. The left-hand side of (*DLMF*, (7.18.4)) is given by

$$\frac{\mathrm{d}^n}{\mathrm{d}z^n}\left(e^{z^2}\operatorname{erfc} z\right),$$

where $n \in \mathbb{N}_0$, e is the base of the natural logarithm, and erfc is the complementary error function. Our translation correctly produces

```
diff((exp((z)^(2))*erfc(z)), [z$(n))].
```

However, the Maple 2016 `simplify` function falsely returns 0 for the translated left-hand side. Maplesoft has confirmed in a private communication that this is indeed a defect in Maple 2016. Furthermore, although the nature of the defect changes, the defect still persists in Maple 2018 as of the publication of this manuscript.

8 Conclusion & Future Work

The translator concept has proven itself by discovering errors in the online DLMF 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, not only can discover errors in mathematical compendia but can also detect errors in simplifications or computations of the CAS.

The tasks for future work are diverse. The main task is to improve the translator by implementing more functions and features. For example, for the current state, only translations to Maple's standard function library were implemented. Maple allows one to load extra packages dynamically and therefore support an enhanced set of functions. This feature would drastically increase the number of possible translations. With such improvements, further work on evaluation techniques become worthwhile to evaluate the DLMF and CAS.

The translator was designed to be easily extendable. This allows one to implement translations for other CAS without much effort. An extensive weakness is the dependency on the special macros from the DLMF. The translator is not able to translate functions without using these macros. Currently, we are working on mathematical information retrieval techniques which will allow for an extension of the translator to generic LaTeX inputs.

References

Alex, G. (June 2007). “Do Open Source Developers Respond to Competition? The (La)TeX Case Study”. In: *Review of Network Economics* 6.2, pp. 1–25.

Bernardin, L., Chin, P., DeMarco, P., Geddes, K. O., Hare, D. E. G., Heal, K. M., Labahn, G., May, J. P., McCarron, J., Monagan, M. B., Ohashi, D., and Vorkoetter, S. M. (2016). *Maple 2016 Programming Guide*. Maplesoft, a division of Waterloo Maple Inc.

Cajori, F. (Mar. 1, 1994). *A History of Mathematical Notations*. Dover Publications Inc. 848 pp.

Churchill, B. and Boyd, S. (2010). *LaTeXCalc*. `https://sourceforge.net/projects/latexcalc/`. Seen 06/2017.

Cohl, H. S., Greiner-Petter, A., and Schubotz, M. (2018). “Automated Symbolic and Numerical Testing of DLMF Formulae using Computer Algebra Systems”. In: *Proceedings of the 11th Conference on Intelligent Computer Mathematics, CICM 2018, RISC, Hagenberg, Austria*. Accepted Full Paper.

Cohl, H. S., McClain, M. A., Saunders, B. V., Schubotz, M., and Williams, J. C. (2014). “Digital Repository of Mathematical Formulae”. In: *Intelligent Computer Mathematics - International Conference, CICM 2014, Coimbra, Portugal, July 7-11, 2014. Proceedings*. Ed. by S. M. Watt, J. H. Davenport, A. P. Sexton, P. Sojka, and J. Urban. Vol. 8543. Lecture Notes in Computer Science. Springer, pp. 419–422.

Cohl, H. S., Schubotz, M., McClain, M. A., Saunders, B. V., Zou, C. Y., Mohammed, A. S., and Danoff, A. A. (2015). “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 M. Kerber, J. Carette, C. Kaliszyk, F. Rabe, and V. Sorge. Vol. 9150. Lecture Notes in Computer Science. Springer, pp. 280–287.

Cohl, H. S., Schubotz, M., Youssef, A., Greiner-Petter, A., Gerhard, J., Saunders, B. V., McClain, M. A., Bang, J., and Chen, K. (2017). “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 H. Geuvers, M. England, O. Hasan, F. Rabe, and O. Teschke. Vol. 10383. Lecture Notes in Computer Science. Springer, pp. 115–131.

Corless, R. M., Jeffrey, D. J., Watt, S. M., and Davenport, J. H. (2000). “"According to Abramowitz and Stegun" or arccoth needn’t be uncouth”. In: *ACM SIGSAM Bulletin* 34.2, pp. 58–65.

Cuypers, H., Cohen, A. M., Knopper, J. W., Verrijzer, R., and Spanbroek, M. (June 2008). “MathDox, a system for interactive Mathematics”. In: *Proceedings of EdMedia: World Conference on Educational Media and Technology 2008*. Ed. by J. Luca and E. R. Weippl. Vienna, Austria: Association for the Advancement of Computing in Education (AACE), pp. 5177–5182.

Davenport, J. H. (2010). “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 S. Autexier, J. Calmet, D. Delahaye, P. D. F. Ion, L. Rideau, R. Rioboo, and A. P. Sexton. Vol. 6167. Lecture Notes in Computer Science. Springer, pp. 1–12.

DLMF. *NIST Digital Library of Mathematical Functions*. Release 1.0.19 of 2018-06-22. 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, editors. URL: http://dlmf.nist.gov/.

Drake, D. (June 2009). *sagetex*. https://ctan.org/tex-archive/macros/latex/contrib/sagetex/. Seen 06/2017. Comprehensive.

England, M., Cheb-Terrab, E. S., Bradford, R. J., Davenport, J. H., and Wilson, D. J. (2014). "Branch cuts in Maple 17". In: *ACM Comm. Computer Algebra* 48.1/2, pp. 24–27.

Geuvers, H., England, M., Hasan, O., Rabe, F., and Teschke, O., eds. (2017). *Intelligent Computer Mathematics - 10th International Conference, CICM 2017, Edinburgh, UK, July 17-21, 2017, Proceedings*. Vol. 10383. Lecture Notes in Computer Science. Springer.

Giceva, J., Lange, C., and Rabe, F. (2009). "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 J. Carette, L. Dixon, C. S. Coen, and S. M. Watt. Vol. 5625. Lecture Notes in Computer Science. Springer, pp. 279–293.

Knuth, D. E. (1997). *The Art of Computer Programming, Volume I: Fundamental Algorithms, 3rd Edition*. Addison-Wesley.

– (June 11, 1998). *Digital Typography*. Reissue. Lecture Notes (Book 78). Center for the Study of Language and Information (CSLI). 685 pp.

Kohlhase, M. (2006). *OMDoc - An Open Markup Format for Mathematical Documents [version 1.2]*. Lecture Notes in Computer Science. Springer.

– (2008). "Using LaTeX as a Semantic Markup Format". In: *Mathematics in Computer Science* 2.2, pp. 279–304.

Kohlhase, M., Corneli, J., David, C., Ginev, D., Jucovschi, C., Kohlhase, A., Lange, C., Matican, B., Mirea, S., and Zholudev, V. (2011). "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 M. Sato, S. Matsuoka, P. M. A. Sloot, G. D. van Albada, and J. Dongarra. Vol. 4. Procedia Computer Science. Elsevier, pp. 598–607.

Miller, B. R. and Youssef, A. (2003). "Technical Aspects of the Digital Library of Mathematical Functions". In: *Ann. Math. Artif. Intell.* 38.1-3, pp. 121–136.

Olver, F. W., Lozier, D. W., Boisvert, R. F., and Clark, C. W. (Apr. 30, 2010). *NIST Handbook of Mathematical Functions*. 1st. New York, NY, USA: Cambridge University Press. 968 pp.

Youssef, A. (2017). "Part-of-Math Tagging and Applications". In: *Intelligent Computer Mathematics - 10th International Conference, CICM 2017, Edinburgh, UK, July 17-21, 2017, Proceedings*. Ed. by H. Geuvers, M. England, O. Hasan, F. Rabe, and O. Teschke. Vol. 10383. Lecture Notes in Computer Science. Springer, pp. 356–374.