Semantic Preserving Bijective Mappings of Mathematical Formulae Between Document Preparation Systems and Computer Algebra Systems

Howard S. Cohl[1](✉), Moritz Schubotz[2], Abdou Youssef[3], André Greiner-Petter[4], Jürgen Gerhard[5], Bonita V. Saunders[1], Marjorie A. McClain[1], Joon Bang[6], and Kevin Chen[6]

[1] Applied and Computational Mathematics Division, NIST, Gaithersburg, MD, USA
{howard.cohl,bonita.saunders,marjorie.mcclain}@nist.gov

[2] Department of Computer and Information Science, University of Konstanz, Konstanz, Germany
moritz.schubotz@uni-konstanz.de

[3] Department of Computer Science, GWU, Washington DC, USA
ayoussef@gwu.edu

[4] DSIMG, Technische Universität, Berlin, Germany
andre.greiner-petter@t-online.de

[5] Maplesoft, Waterloo, ON, Canada
jgerhard@maplesoft.com

[6] Poolesville High School, Poolesville, MD, USA
joonb3@gmail.com, kchen1250@gmail.com

Abstract. Document preparation systems like LaTeX offer the ability to render mathematical expressions as one would write these on paper. Using LaTeX, LaTeXML, and tools generated for use in the National Institute of Standards (NIST) Digital Library of Mathematical Functions, semantically enhanced mathematical LaTeX markup (semantic LaTeX) is achieved by using a semantic macro set. Computer algebra systems (CAS) such as Maple and Mathematica use alternative markup to represent mathematical expressions. By taking advantage of Youssef's Part-of-Math tagger and CAS internal representations, we develop algorithms to translate mathematical expressions represented in semantic LaTeX to corresponding CAS representations and vice versa. We have also developed tools for translating the entire Wolfram Encoding Continued Fraction Knowledge and University of Antwerp Continued Fractions for Special Functions datasets, for use in the NIST Digital Repository of Mathematical Formulae. The overall goal of these efforts is to provide semantically enriched standard conforming MathML representations to the public for formulae in digital mathematics libraries. These representations include presentation MathML, content MathML, generic LaTeX, semantic LaTeX, and now CAS representations as well.

© Springer International Publishing AG 2017
H. Geuvers et al. (Eds.): CICM 2017, LNAI 10383, pp. 115–131, 2017.
DOI: 10.1007/978-3-319-62075-6_9

1 Problem and Current State

Scientists often use document preparation systems (DPS) to write scientific papers. The well-known DPS LaTeX has become a de-facto standard for writing mathematics papers. On the other hand, scientists working with formulae which occur in their research often need to evaluate special or numerical values, create figures, diagrams and tables. One often uses computer algebra systems (CAS), programs which provide tools for symbolic and numerical computation of mathematical expressions. DPS such as LaTeX, try to render mathematical expressions as accurately as possible and give the opportunity for customization of the layout of mathematical expressions. Alternatively, CAS represent expressions for use in symbolic computation with secondary focus on the layout of the expressions. This difference in format is a common obstacle for scientific workflows.

For example, consider the Euler-Mascheroni (Euler) constant represented by γ. Since generic LaTeX [3] does not provide any semantic information, the LaTeX representation of this mathematical constant is just the command for the Greek letter `\gamma`. Maple[1] and Mathematica, well-known CAS, represent the Euler constant γ with `gamma` and `EulerGamma` respectively. Scientists writing scientific papers, who use CAS often need to be aware of representations in both DPS and CAS. Often different CAS have different capabilities, which implies that scientists might need to know several CAS representations for mathematical symbols, functions, operators, etc. One also needs to be aware when CAS do not support direct translation. We refer to CAS translation as either the forward or backward translation respectively as DPS source to CAS source or vise-versa. For instance, the CAS representation of the number $e \approx 2.71828$ (the base of the natural logarithm) in Mathematica is `E`, whereas in Maple there is no directly translated symbol. In Maple, one needs to evaluate the exponential function at one via `exp(1)` to reproduce its value.

For a scientist, γ and e might represent something altogether different from these constants, such as a variable, function, distribution, vector, etc. In these cases, it would need to be translated in a different way. In order to avoid these kinds of semantic ambiguities (as well as for other reasons), Bruce Miller at NIST, developing for the Digital Library of Mathematical Functions (DLMF) (special functions and orthogonal polynomials of classical analysis) project, has created a set of semantic LaTeX macros [11,13]. Extensions and 'simplifications' have been provided by the Digital Repository of Mathematical Formulae (DRMF) project. We refer to this extended set of semantic LaTeX macros as the DLMF/DRMF macro set, and the mathematical LaTeX which uses this semantic macro set as semantic LaTeX.

[1] 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.

Existing tools which attempt to achieve CAS translations include import/export for LaTeX expressions (such as [10,15]), as well as for MathML. CAS functions such as these, mostly provide only presentation translation in LaTeX and do not provide semantic solutions or workarounds to hidden problems such as subtle differences in CAS function definitions. These differences may also include differences in domains or complex branch cuts of multivalued functions. To fill this lack of knowledge in the CAS translation process, one needs to provide additional information in the DPS source itself and to create interactive documents with references to definitions, theorems and other representations of mathematical expressions. Our approach in this paper, is to develop independent tools for translation between different CAS and semantic LaTeX representations for mathematical expressions. We provide detailed information about CAS translation and warn about the existence of known differences in definitions, domains and branch cuts. For the DRMF, we have decided to focus on CAS translation between the semantic LaTeX representations of classical analysis and internal CAS representations for Maple and Mathematica.

1.1 A CAS, Generic and Semantic LaTeX Representation Example

An example of a mathematical expression is $P_n^{(\alpha,\beta)}(\cos(a\Theta))$ where $P_n^{(\alpha,\beta)}$ is the Jacobi polynomial [5, (18.5.7)]. Table 1 illustrates several DPS and CAS representations for this mathematical expression. Translating the generic LaTeX representation is difficult (see [3]) since the semantic context of the P is obscured. If it represents a special function, one needs to ascertain which function it represents, because there are many examples of standard functions in classical analysis which are given by a P. The semantic LaTeX representation of this mathematical expression encapsulates the mostly-unambiguous semantic meaning of the mathematical expression. This facilitates translation between it and CAS representations. We use the first scan of the Part-of-Math (POM) tagger [19] to facilitate translation between semantic LaTeX and CAS representations.

Table 1. DPS and CAS representations for Jacobi polynomial expression

Different Systems	Different 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]]]`

2 The Part-of-Math Tagger

There are different approaches for interpreting LaTeX. There exist several parsers for LaTeX, for instance `texvcjs`, which is a part of Mathoid [18]. There is also

LaTeXML [12,13] which processes LaTeX. There is also an alternative grammar developed by Ginev [7]. A new approach has been developed [19] which is not a fully fledged grammar but only extracts POM from math LaTeX. The purpose of the POM is to extract semantic information from mathematics in LaTeX. The tagger works in several stages (termed *scans*) and interacts with several machine learning (ML) based algorithms.

Given an input LaTeX math document, the first scan of the tagger examines terms and groups them into sub-expressions when indicated. For instance `\frac{1}{2}` is a sub-expression of numerator and denominator. A term is, in the sense of Backus-Naur form, a pre-defined non-terminal expression and can represent LaTeX macros, environments, reserved symbols (such as the LaTeX line break command \\) or numerical or alphanumerical expressions. Sub-expressions and terms get tagged due the first scan of the tagger, with two separate tag categories: (1) definite tags (such as *operation*, *function*, *exponent*, etc.) that the tagger is certain of; and tags which consist of alternative and tentative features which include alternative roles and meanings. These second category of tags are drawn from a specific knowledge base which has been collected for the tagger. Tagged terms are called math terms. Math terms are rarely distinct at this stage and often have multiple features.

Scans 2 and 3 are expected to be completed in the next 2 years. These involve some natural language processing (NLP) algorithms as well as ML-based algorithms [14,17]. Those scans will: (1) select the right features from among the alternative features identified in the first scan; (2) disambiguate the terms; and (3) group subsequences of terms into unambiguous sub-expressions and tag them, thus deriving definite mostly-unambiguous semantics of math terms and expressions. The NLP/ML algorithms include math topic modeling, math context modeling, math document classification (into various standard areas of math), and definition-harvesting algorithms.

Specifically, to narrow down the role/meaning of a math term, it helps to know which area of mathematics the input document is in. This calls for a *math-document classifier*. Furthermore, knowing the topic, which is more specific than the area of the document, will shed even more light on the math terms. Even more targeted is the notion of *context* which, if properly formulated, will take the POM tagger a long way in narrowing down the tag choices.

In [19], Youssef defines a new notion of a math-term's context, which involves several components, such as (1) the area and topic of the term's document; (2) the document-provided definitions; (3) the topic model and theme class of the term's *neighborhood* in the document; (4) the actual mathematical expression containing the term; as well as (5) a small number of natural language sentences surrounding the mathematical expression. Parts of this context are the textual definitions and explanations of terms and notations which can be present or absent from the input document. These can also be near the target terms or far and distributed from them. The NLP/ML-based algorithms for the 2$^{\text{nd}}$ and 3$^{\text{rd}}$ scans of the tagger will model and track the term's contexts, and will harvest definitions and explanations and associate them with the target terms.

3 Semantic LaTeX to CAS Translation

We have used a mathematical language parser (MLP) as an interface for the above-described first scan of the POM tagger to build syntax trees of mathematical expressions in LaTeX and provide CAS translations from semantic LaTeX to CAS representations. The MLP provides all functionality to interact with the results of the POM tagger. We extended the general information of each term to its CAS representation, links to definitions on the DLMF/DRMF websites, as well as the corresponding CAS websites. We also add information about domains, position of branch cuts and further explanations if necessary. Since the multiple scans of the POM tagger are still a work in progress, our CAS translation is based on the first scan (see Sect. 2). Figure 1 shows the syntax tree corresponding to the LaTeX expression `\sqrt[3]{x^3} + \frac{y}{2}`; note that '`x`' and '`3`' in '`x^3`' are not treated (in Fig. 1) as siblings (i.e., children of '`^`') because the first scan of the tagger does not recognize this hierarchy (but it will be rectified in POM Scans 2 and 3). The general CAS translation process translates each node without changing the hierarchy of the tree recursively. With this approach, we are able to translate nested function calls.

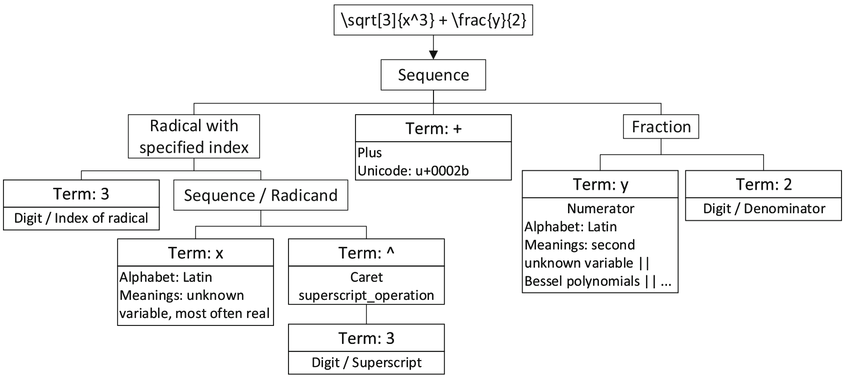


Fig. 1. Syntax tree of $\sqrt[3]{x^3} + \frac{y}{2}$ produced by the first scan of the POM tagger.

The syntax tree obtained by the first POM scan depends on the known terms of the tagger. Although the tagger's first scan tags macros if those macros' definition are provided to it, it is currently agnostic of the DLMF/DRMF macros. Therefore, as it currently stands, the first scan of the tagger extracts, but does not recognize/tag DLMF/DRMF macros as hierarchical structures, but rather treats those macros as sequences of terms. The syntax tree in Fig. 2 was created by the tagger for our Jacobi polynomial example in Sect. 1.1. The tagger extracts expressions enclosed between open and closed curly braces {...} which we refer to as *delimited balanced expressions*. The given argument is a sub-expression and produces another hierarchical tree structure.

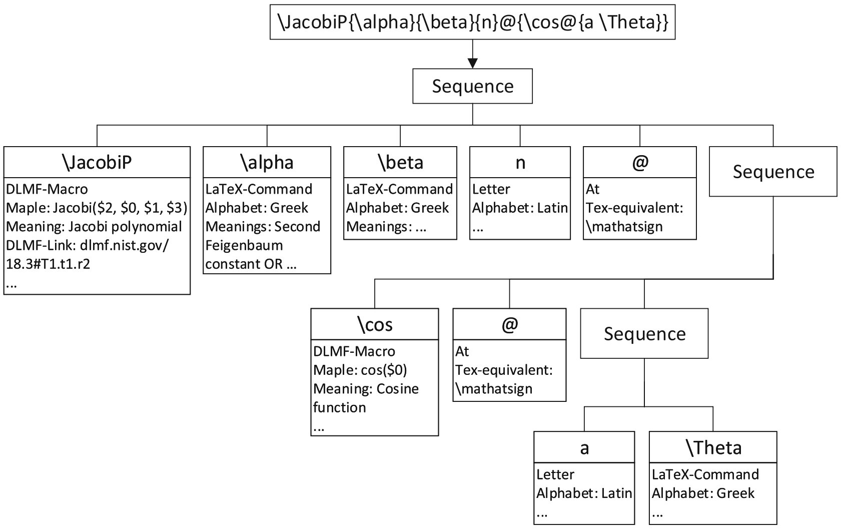


Fig. 2. Syntax tree for Jacobi polynomial expression generated by the first POM scan.

3.1 Implementation

CAS translations for DLMF/DRMF macros are stored in `CSV` files, to make them easy to edit. Besides that, CAS translations for Greek letters and mathematical constants are stored separately in JSON files. In addition to the DLMF/DRMF macro set, generic LaTeX also provides built-in commands for mathematical functions, such as `\frac` or `\sqrt`. CAS translations for these macros are defined in another JSON file.

Table 2. A lexicon entry.

DLMF	`\sin@@z`
DLMF-Link	dlmf.nist.gov/4.14#E1
Maple	`sin($0)`
Mathematica	`Sin[$0]`

Since the POM tagger assumes the existence of special formatted lexicon files to extract information for unknown commands, the `CSV` files containing CAS translation information has to be converted into lexicon files. Table 2 shows a part of the lexicon entry for the DLMF/DRMF macro `\sin@@{z}`[2]. Translations to CAS are realized by patterns with placeholders. The symbol `$i` indicates the i-th variable or parameter of the macro.

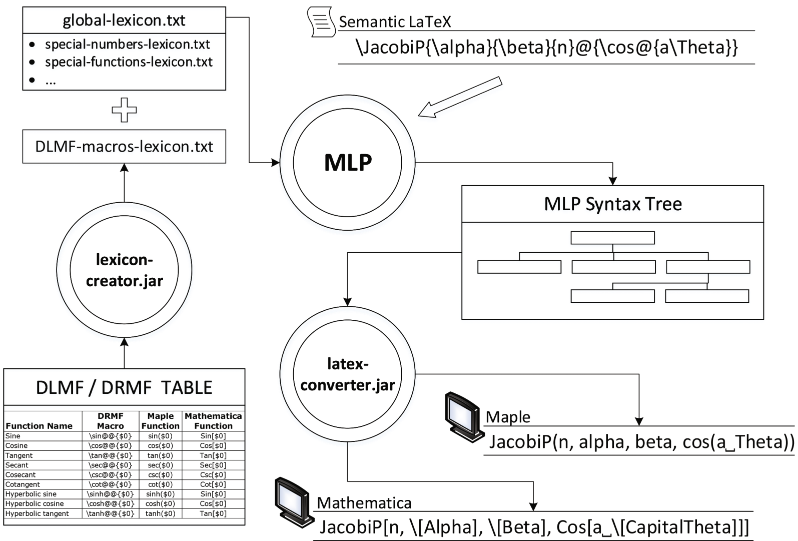
Our CAS translation process is structured recursively. A CAS translation of a node will be delegated to a specialized class for certain kinds of nodes. Even though our CAS translation process assumes semantic LaTeX with DLMF/DRMF macros, we sometimes allow for extra information obtained from

[2] The usage of multiple @ symbols in Miller's LaTeX macro set provides capability for alternative presentations, such as $\sin(z)$ and $\sin\ z$ for one and two @ symbols respectively.

generic LaTeX expressions. For instance, we distinguish between the following cases: (1) a Latin letter is used for an elementary constant; (2) a generic LaTeX command (such as the LaTeX command for a Greek letter) is used for an elementary constant. In both cases, the program checks if there are known DLMF/DRMF macros to represent the constant in semantic LaTeX. If so, we inform the user of the DLMF/DRMF macro for the constant, but the Latin letter or LaTeX command is not translated.

There are currently only three known Latin letters where this occurs, the imaginary unit i, Euler's number e, and Catalan's constant C. If one wants to translate the Latin letter to the constant, then one needs to use the designated macro. In these three cases they are `\iunit`, `\expe` and `\CatalansConstant`. Examples of LaTeX commands which may represent elementary constants are π and α which are often used to represent the ratio of a circle's circumference to its diameter, and the fine-structure constant respectively which are `\cpi` and `\finestructure`. Hence, Latin and Greek letters will be always translated as Latin and Greek letters respectively.

The program consists of two executable JAR files. One organizes the transformation from `CSV` files to lexicon files, while the other translates the generated syntax tree to a CAS representation. Figure 3 describes the CAS translation process. The program currently supports forward CAS translations for Maple and Mathematica.


global-lexicon.txt
special-numbers-lexicon.txt
special-functions-lexicon.txt
...
DLMF-macros-lexicon.txt
lexicon-creator.jar
Semantic LaTeX
\JacobiP{\alpha}{\beta}{n}@{\cos@{a\Theta}}
MLP
MLP Syntax Tree
latex-converter.jar
Maple
JacobiP(n, alpha, beta, cos(a_Theta))
Mathematica
JacobiP[n, \[Alpha], \[Beta], Cos[a_\[CapitalTheta]]]
DLMF / DRMF TABLE

Function Name	DRMF Macro	Maple Function	Mathematica Function
Sine	\sin@@{$0}	sin($0)	Sin[$0]
Cosine	\cos@@{$0}	cos($0)	Cos[$0]
Tangent	\tan@@{$0}	tan($0)	Tan[$0]
Secant	\sec@@{$0}	sec($0)	Sec[$0]
Cosecant	\csc@@{$0}	csc($0)	Csc[$0]
Cotangent	\cot@@{$0}	cot($0)	Cot[$0]
Hyperbolic sine	\sinh@@{$0}	sinh($0)	Sin[$0]
Hyperbolic cosine	\cosh@@{$0}	cosh($0)	Cos[$0]
Hyperbolic tangent	\tanh@@{$0}	tanh($0)	Tan[$0]

Fig. 3. Flow diagram for translation between semantic LaTeX and a CAS representations. The MLP is the only interface to the POM tagger and provides all functionality for interaction with the results of the POM tagger (such as analyzing the syntax tree and extracting information from the lexicon.)

4 Maple to Semantic LaTeX Translation

Maple has its own syntax and programming language, and users interact with Maple by entering commands and expressions in Maple syntax. For example, the mathematical expression $\int_0^\infty (\pi + \sin(2x))/x^2 dx$, would be entered in Maple as

```
int((Pi+sin(2*x))/x^2, x=0..infinity).        (1)
```

In the sequel, we will refer to Maple syntax such as the syntactically correct format (1) as (i) the 1D Maple representation. Maple also provides a (ii) 2D representation (whose internal format is similar to MathML), and its display is similar to the LaTeX rendering of the mathematical expression. In addition, Maple uses two internal representations (iii) Maple_DAG, and (iv) Inert_Form representation. Note that, even though DAG commonly refers to the general graph theoretic/generic data structure, *directed acyclic graph*, in Maple it has become synonymous with "Maple internal data structure," whether it actually represents a DAG or not.

In our translation from Maple to semantic LaTeX, only the Maple 1D and Inert_Fo- rm representations are used. Programmatic access to the Maple kernel (its internal data structures/commands) from other programming languages such as Java or C is possible through a published application programming interface (API) called OpenMaple [8, Sect. 14.3]. The OpenMaple Java API is used in this project. Some of the functionality used includes (1) parsing a string in 1D representation and converting it to its Maple_DAG and Inert_Form representations (see below); (2) accessing elements of Maple's internal data structures; (3) performing manipulations on Maple data structures in the Maple kernel.

Mathematical expressions in Maple are internally represented as Maple_DAG representations. Figure 4 illustrates the Maple_DAG representation of the 1D Maple expression (1). The variable x is stored only once in memory, and all three occurrences of it refer to the same Maple object. This type of common subexpression reuse is the reason why Maple data structures are organized as DAGs and not as trees. In addition to mathematical expressions, Maple also has a variety of other data structures (e.g., sets, lists, arrays, vectors, matrices, tables, procedures, modules). The structure of a Maple_DAG is in the form $\boxed{\textit{Header}}\boxed{\textit{Data}_1 \cdots \textit{Data}_n}$. *Header* encodes both the type and

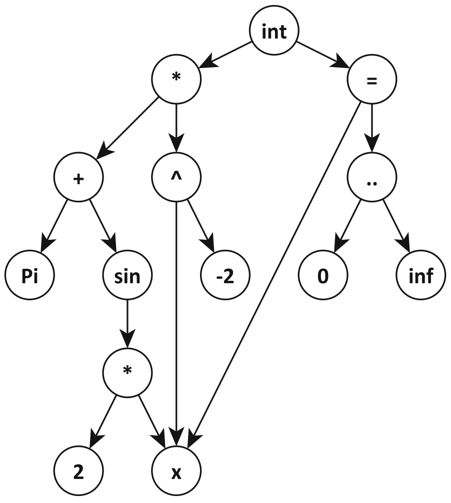


Fig. 4. Example Maple_DAG for (1).

the length n of the `Maple_DAG` and $Data_1, \ldots, Data_n$ are `Maple_DAG`s (see [8, Appendix A.3]).

For this project, another tree-like representation that closely mirrors the internal `Maple_DAG` representation (and can be accessed more easily through the OpenMaple Java API) was chosen, the `Inert_Form`. The `Inert_Form` is given by nested function calls of the form `_Inert_XXX` ($Data_1$, ..., $Data_n$), where `XXX` is a type tag (see [8, Appendix A.3]), and $Data_1$, ..., $Data_n$ can themselves be `Inert_Form`s. In Maple, the `Inert_Form` representation can be obtained via the command `ToInert`. For example, the `Inert_Form` representation of the Maple expression (1) is

```
_Inert_FUNCTION( _Inert_NAME("Int"), _Inert_EXPSEQ(_Inert_PROD( _Inert_SUM( _Inert_NAME("Pi"),
_Inert_FUNCTION( _Inert_NAME("sin"), _Inert_EXPSEQ( _Inert_PROD(_Inert_NAME("x"),
_Inert_INTPOS(2))))), _Inert_POWER( _Inert_NAME("x"), _Inert_INTNEG(2)))
_Inert_EQUATION( _Inert_NAME("x"), _Inert_RANGE( _Inert_INTPOS(0),_Inert_NAME("infinity"))))).
```

In order to facilitate access to the `Inert_Form` from the OpenMaple Java API, the `Inert_Form` is converted to a `nested list` representation (Fig. 5), where the first element of each (sub)-list is an `_Inert_XXX` tag. For example, the Maple equation `x=0..infinity` which contains the integration bounds (which is a sub-`Maple_DAG` of Maple expression (1)), is as follows in the `nested list` representation of the `Inert_Form`:

```
[_Inert_EQUATION, [_Inert_NAME, "x"], [_Inert_RANGE, [_Inert_INTPOS, 0], [_Inert_NAME, "infinity"]]].
```

4.1 Implementation

Our CAS translation engine enters the `1D Maple` representation via the OpenMaple API for Java [9] and converts the previously described `Inert_Form` to a `nested list` representation. For Maple expressions, the `nested list` has a tree structure. We have organized the backward translation in a similar fashion to the forward translation (see Sect. 3).

Since Maple automatically tries to simplify input expressions, we implemented some additional changes to prevent such simplifications and changes to the input expression. We would prefer that the representation of a translated expression remain as similar as possible to the input expression. This facilitates user comprehension, as well as the debugging process, of the CAS translation. Maple's internal representation presents obstacles when trying to keep an internal expression in the syntactical form of the input expression. For instance, Maple performs automatic (1) simplification of input expressions; (2) representation of radicals as powers with fractional exponents (e.g., `\sqrt[5]{x^3}` represented as `x^{3/5}`); (3) representation of negative terms as positive terms multiplied by `-1` (since Maple's internal structure has no primitives for negation or subtraction); and (4) representation of division by a term as a multiplication of that term raised to a negative power (since Maple's internal structure has no primitives for division).

To prevent automatic simplifications in Maple, one can enclose input expressions between single quotes '...', also known as `unevaluation quotes`. This does

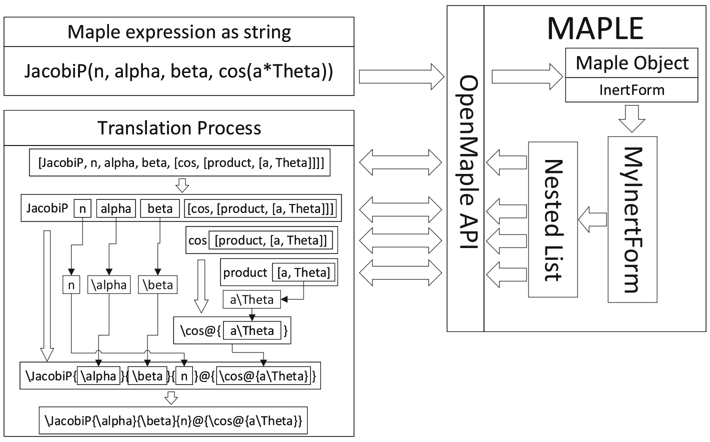

Maple expression as string
JacobiP(n, alpha, beta, cos(a*Theta))
Translation Process
[JacobiP, n, alpha, beta, [cos, [product, [a, Theta]]]]
JacobiP
n
alpha
beta
[cos, [product, [a, Theta]]]
cos
[product, [a, Theta]]
product
[a, Theta]
a\Theta
n
\alpha
\beta
\cos@{ a\Theta }
\JacobiP{ \alpha }{ \beta }{ n }@{ \cos@{a\Theta} }
\JacobiP{\alpha}{\beta}{n}@{\cos@{a\Theta}}
OpenMaple API
MAPLE
Maple Object
InertForm
MyInertForm
Nested List

Fig. 5. The program flow diagram explains the translation from Maple to semantic LaTeX. The input string is parsed into a Maple object and Maple procedures create a new internal form of the object and builds a `nested list` from this new form. The CAS translation process assembles the semantic LaTeX expression by translating each element recursively.

not prevent arithmetic simplifications but does prevent all other simplifications to the input expression. For instance, if we have input `sin(Pi)+2-1`, then the output is `1`; and if we have input `'sin(Pi)+2-1'`, then the output is `sin(Pi)+1`. By using unevaluation quotes, Maple does not convert a radical to a power with fractional exponents, and the internal representation remains an unevaluated `sqrt` (for square roots) or `root` (for higher order radicals). Maple automatically represents a negative term such as `-a` by a product `a*(-1)`. To resolve this we first switch the order of the terms so that constants are in front, e.g., `(-1)*a`, and then check if the leading constant is positive or negative. If it is negative, we remove the multiplication and insert a negative sign in front of the term.

Maple's rendering engine only changes negative powers to fractions if the power is a ratio of integers, otherwise it keeps the exponent representation. We only translate terms with negative integer exponents to fractions, and otherwise retain the internal exponent representation. For this purpose, we perform a preprocessing step (in Maple) that introduces a new `DIVIDE` element in the tree representation. For instance, without the `DIVIDE` element the input `(1/(x+3))^(-I)` produces `\left((3+x)^{-1}\right)^{-\iunit}`, and with the `DIVIDE` element it produces `\left(\frac{1}{3+x}\right)^{-\iunit}`.

Using the above described manipulations, a typical translated expression is very similar to the input expression. As an example, without any of the techniques above, the input expression `cos(Pi*2)/sqrt((3*beta)/4-3*I)` would be automatically simplified and changed internally, and the resulting semantic LaTeX would be `2\idot(3\idot\beta+12\idot\iunit\idot(-1))^{-\frac{1}{2}}`.[3] With unevaluation quotes, the CAS translation produces

```
\cos@{\cpi\idot2}\idot\left(\sqrt{\beta\idot\frac34+\iunit\idot(-3)}\right)^{-1}.
```

Furthermore, with our improvements for subtractions, we translate the radicand to `\frac{3}{4}\idot\beta-3\idot\iunit`, and with the `DIVIDE` element, we translate the base with exponent `-1` as a fraction, and our translated expression is

```
\frac{\cos@{2\idot\cpi}}{\sqrt{\frac{3}{4}\idot\beta-3\idot\iunit}},
```

which is very similar to the input expression.

5 Translation of the Maple/Mathematica CFSF/eCF datasets

Table 3. Example `CFSF` statement

```
create( 'contfrac',
  label = "EF.exp.sfrac.01",
  booklabel = "11.1.2",
  dlmflabel = "4.9.3",
  front = 1,
  begin = [[2*z, 2-z], [z^2/6, 1]],
  general = [[(1/(4*(2*m-3)*(2*m-1)))*z^2,1]],
  function = exp,
  lhs = exp(z),
  category = "S-fraction"
):
```

We have developed Python tools to convert the Maple *Continued Fractions for Special Functions* (`CFSF`) formulae dataset [2]. This dataset is connected with the book *Handbook of Continued Fractions for Special Functions* (2008) [4] into semantic LaTeX. Using this semantic LaTeX we generate MediaWiki Wikitext for seeding in the DRMF. The Maple source for `CFSF` formulae is distributed in many `.mpl` files which are stored in a hierarchical directory substructure. The Maple source for `CFSF` formulae consists of `create` statements (e.g., Table 3) which contain a sequence of fields describing the details of the formula. The fields are: `type`, `category`, `constraints`, `begin`, `factor`, `front`, `parameters`, `booklabel`, `dlmflabel`, `function`, `lhs`, `even`, `odd`, and `general`.

For the example create statement in Table 3, we generate the semantic LaTeX:

```
\expe^{z}=1+\frac{2z}{2-z}\subplus\frac{\frac{z^{2}}{6}}{1}\subplus
\CFK{m}{3}{\infty}@@{\frac{1}{4\left(2m-3\right)\left(2m-1\right)}z^{2}}{1},
```

which when rendered by `pdflatex` produces the formula

$$e^z = 1 + \frac{2z}{2-z} + \frac{\frac{z^2}{6}}{1} + \mathop{\mathrm{K}}_{m=3}^{\infty}\left(\frac{\frac{1}{4(2m-3)(2m-1)}z^2}{1}.\right) \tag{2}$$

[3] `\idot` is our semantic LaTeX macro which represents multiplication without any corresponding presentation appearance.

The `type` field, which is the first argument of the `create` statement, represents its 0th field. (For the example in Table 3, `type=contfrac`). All fields except `category`, `parameters`, and `function` are used for conversion (although they contain potentially useful semantic information). After each field has been translated to semantic LaTeX, they are assembled together. For instance, `front` is joined with `begin`, which based on `type`, is determined to be a list of fractions joined together by `subplus` symbols. Finally `general` is merged with a `subplus`.

We operate on two external dictionaries. Dict. 1 stores every unique Maple symbol/function occurring in the `CFSF` dataset as well as their semantic LaTeX representations. These are stored as string arrays to denote positions of function arguments. Dict. 2 provides necessary information to convert our substructure of directories which contain the `.mlp` files, as well as their contents to generate sections and subsections in the produced LaTeX file.

Our translation starts with a tokenization process which searches for occurrences of key symbols/characters (e.g., mathematical operators, parentheses, spaces) within a given Maple representation and splits the string on those terms. This produces a list of tokens. We categorize into three types of tokens: (1) `normal` (operands, numbers, variables, etc.); (2) `operator` (addition, subtraction, negative sign, etc.); and (3) `function call` (those functions which are called). After tokenization, each token is parsed and translated to semantic LaTeX, and then re-assembled as follows. When encountering a `normal` token, we check that it is defined as having a translation in Dict. 1. If so, the token is swapped with the corresponding LaTeX macro. When encoutering an `operator` token, it is identified as either unary/binary, or as `enclosure` (i.e., parenthesis, square bracket, or curly brace). If `operator` is unary, the token after `operator` is used to generate the LaTeX representation. Binary `operator` is handled similarly, but instead uses preceding and following terms. If left `enclosure` is found, a flag is set until a matching `enclosure` is caught, and then the contents between matching `enclosures` are rebuilt (following the order of operations). If `function call` is encountered (listed in the Dict. 1), then we search for the corresponding delimiter. Function arguments are extracted by comparing with the Maple function (in Dict. 1), and then translated and used to replace dummy arguments. Finally, the newly built expression replaces the tokens from the original `function call` for the discovered delimiter. We translate 252 `CFSF` formulae from 55 files located in 10 subdirectories. This corresponds to an output semantic LaTeX file with 10 sections and 55 subsections.

We have also developed code to translate the Wolfram *Encoding Continued Fraction Knowledge* (`eCF`) dataset [6,16] to semantic LaTeX. We create Dict. 3, which contains a list of corresponding Mathematica functions/semantic LaTeX. An example statement from the `eCF` dataset is

```
ConditionalExpression[E^z==1+(2*z)/(2-z+z^2/(6*(1+Inactive[ContinuedFractionK]
[z^2/(4*(1+2*k)*(3+2*k)),1,{k,1,Infinity}]))),Element[z,Complexes]].
```

Our code produces the following semantic LaTeX

```
\expe^{z}=1+\frac{2z}{2-z+\frac{z^{2}}{6\left(1+\CFK{k}{1}{\infty}
@@{\frac{z^{2}}{4\left(1+2k\right)\left(3+2k\right)}}{1}\right)}}.
```

This produces the following rendered formula

$$e^z = 1 + \cfrac{2z}{2 - z + \cfrac{z^2}{6\left(1+\underset{k=1}{\overset{\infty}{\mathrm{K}}}\left(\frac{\frac{z^2}{4(1+2k)(3+2k)}}{1}\right)\right)}}.$$

We input the eCF Mathematica dataset from a single Identities.m file and process every Mathematica expression as follows. For each Mathematica expression, (1) we identify all Mathematica function occurences; (2) extract its arguments; (3) and rebuild the corresponding semantic LaTeX expression from Dict. 3. During extraction, the program identifies the location of the function within a formula and searches until it finds a left bracket, indicating the beginning of a Mathematica function. Our splitting process is able to recognize recursive macro calls (matching fence symbols). We translate 1365 Mathematica eCF formulae.

6 Evaluation

Here, we describe our approach for validating the correctness of our mappings, as well as discuss the performance of our system obtained on a hand crafted test set.

One validation approach is to take advantage of numerical evaluation using software tools such as the DLMF Standard Reference Tables (DLMF Tables) [1], CAS, and software libraries[4]. These tools provide numerical evaluation for special functions with their own unique features. One can validate forward CAS translations by comparing numerical values in CAS to ground truth values.

Another validation approach is to use mathematical relations between different functions. For instance, if we forward translate two functions separately, one could determine if the relation between the two translated functions remains valid. One example relation is for the Jacobi elliptic functions sn, cn, dn, and the complete elliptic integral K [5, Table 22.4.3], namely $\mathrm{sn}(z + K(k), k) = \mathrm{cn}(z, k)/\mathrm{dn}(z, k)$, where $z \in \mathbb{C}$, and $k \in (0, 1)$. In the limit as $k \to 0$, this relation produces $\sin\left(z + \frac{\pi}{2}\right) = \cos z$, where $z \in \mathbb{C}$. The DLMF provides relations such as these for many special functions. An alternative relation is particularly helpful to validate CAS translations with different positions of branch cuts, namely the relation between the parabolic cylinder function U and the modified Bessel function of the second kind [5, (12.7.10)] $U(0, z) = \sqrt{z/(2\pi)}K_{1/4}(\frac{1}{4}z^2)$, where $z \in \mathbb{C}$. Note that z^2 is no longer on the principal branch of the modified Bessel function of the second kind when $\mathrm{ph}(z) \in (\frac{\pi}{2}, \pi)$, but a CAS would still compute values on the principal branch. Therefore, a CAS translation from `\BesselK{\frac{1}{4}}@{\frac{1}{4}z^2}` to `BesselK(1/4,(1/4)*z^2)` is incorrect if $\mathrm{ph}(z) \in (\frac{\pi}{2}, \pi)$, even though the equation is true in that domain. In order for the CAS to verify the formula in that domain, it must use [5, (10.34.4)]

[4] See for instance: http://dlmf.nist.gov/software.

for the function on the right-hand side. Other validation tests may not be able to identify a problem with this CAS translation.

One obstacle for such relations are the limitations of ever-improving CAS simplification functions. Define the formula difference, as the difference between the left- and right-hand sides of a mathematical formula. CAS simplify for the Jacobi elliptic/trigonometric relation should produce `0`, but might have more difficulties with the parabolic cylinder function relation. However, CAS simplify functions work more effectively on round trip tests.

6.1 Round Trip Tests

One of the main techniques we use to validate CAS translations are round trip tests which take advantage of CAS simplification functions. Since we have developed CAS translations between semantic LaTeX ↔ Maple, round trip tests are evaluated in Maple. Maple's simplification function is called `simplify`. Two expressions are symbolically equivalent, if `simplify` returns zero for the formula difference. On the other hand, it is not possible to disprove the equivalence of the expressions when the function returns something different to zero.

Our round trip tests start either from a valid semantic LaTeX expression or from a valid Maple expression. A CAS translation from the start representation to the other representation and back again is called one cycle. A round trip reaches a fixed point, when the string representation is identical to its previous string representation. The round trip test concludes when it reaches a fixed point in both representations. Additionally, we test if the fixed point representation in Maple is symbolically equivalent to the input representation by simplifying the differences between both of these with the Maple simplify function. Since there is no mathematical equivalence tester for LaTeX expressions (neither generic nor semantic LaTeX), we manually verify LaTeX representations for our test cases by rendering the LaTeX.

As shown in Sect. 4.1, prior to backward translation, in round trip testing, there will be differences between input and output Maple representations. After adapting these changes, and assuming the functions exist in both semantic LaTeX and CAS, the round trip test should reach a fixed point. In fact, we reached a fixed point in semantic LaTeX after one cycle and in Maple after $1\frac{1}{2}$ cycles (see Table 4 for an example) for most of the cases we tried. If the input representation is already identical to Maple's representation, then the fixed point will be reached after at most a half cycle.

Table 4. A round trip test reach a fixed point.

step	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)`

One example exception is for CAS translations which introduce additional function compositions on arguments. For instance, Legendre's incomplete elliptic integrals [5, (19.2.4–7)] are defined with the amplitude ϕ in the first argument, while Maple's implementation takes the trigonometric sine of the amplitude as the first argument. For instance, one has the CAS translations `\EllIntF@{\phi}{k}` $\mapsto$ `EllipticF(sin(phi),k)`, and `\EllIntF@{\asin@{z}}{k}` $\leftarrow\!\shortmid$ `EllipticF(z,k)`. These CAS translations produce an infinite chain of sine and inverse sine function calls. Because round trip tests prevent simplification during the translation process (see Sect. 4.1), Maple is not used to simplify the chain until the round trip test is concluded.

6.2 Summary of Evaluation Techniques

Equivalence tests for special function relations are able to verify relations in CAS as well as identify hidden problems such as differences in branch cuts and CAS limitations. We use the simplify method to test equivalences. For the relations in Sect. 6, CAS simplify for the Jacobi elliptic function example yields `0`. Furthermore, a spectrum of real, complex, and complex conjugate numerical values for z and $k \in (0, 1)$ the formula difference converges to zero for an increasing precision. If simplification returns something other than zero, we can test the equivalence for specific values. For the Bessel function relation, the formula difference for $z = 1+i$ converges to zero for increasing precision, but does not converge to zero if $z = -1 + i$. However, using analytic continuation [5, (10.34.4)], it does converges to zero. Clearly, the numerical evaluation test is also able to locate branch cut issues in the CAS translation. Furthermore, this provides a very powerful debugging method for our translation as well as for CAS functionality. This was demonstrated by discovering an overall sign error in DLMF equation [5, (14.5.14)].

Round trip tests are also useful for identifying syntax errors in the semantic LaTeX since the CAS translation then fails. The simplification procedure is improved for round trip tests, because it only needs to simplify similar expressions with identical function calls. However, this approach is not able to identify hidden problems that a CAS translation might need to resolve in order to be correct, if the round trip test has not reached a fixed point. Other than with the round trip test approach, we have not discovered any automated tests for backward CAS translations. We have evaluated 37 round trip test cases which produce a fixed point, similar to that given in Table 4. These use formulae from the DLMF/DRMF and produce a difference of the left- and right-hand sides equaling `0`.

We have created a test dataset[5] of 4,165 semantic LaTeX formulae, extracted from the DLMF. We translated each test case to a representation in Maple and used Maple's `simplify` function on the formula difference to verify that the translated formulae remain valid. Our forward translation tool (Sect. 3) was able to translate 2,232 (approx. 53.59%) test cases and verify 477 of these. Pre-conversion improved the effectiveness of `simplify` and were used to convert the translated expression to a different form before simplification of the formula difference. We used conversions to exponential and hypergeometric form and expanded the translated expression. Pre-conversion increased the number of formulae verified to 662 and 1,570 test cases were translated but not verified. The remaining 1,933 test cases were not translated, because they contain

[5] We are planning to make the dataset available from http://drmf.wmflabs.org.

DLMF/DRMF macros without a known translation to Maple (987 cases), such as the q-hypergeometric function [5, (17.4.1)] (in 58 cases), or an error appeared during the translation or verification process (639 cases). Furthermore, 316 cases were ignored, because they did not contain enough semantic information to provide a translation or the test case was not a relation. It is interesting to note that we were able to enhance the semantics of 74 Wronskian relations by rewriting the macro so that it included the variable that derivatives are taken with respect to as a parameter. A similar semantic enhancement is possible for another 186 formulae where the potentially ambiguous prime notation '′' is used for derivatives.

Acknowledgements. We are indebted to Wikimedia Labs, the XSEDE project, Springer-Verlag, the California Institute of Technology, and Maplesoft for their contributions and continued support. We would also like to thank Eric Weisstein for supplying the Wolfram eCF dataset, Annie Cuyt, Franky Backeljauw, and Stefan Becuwe for supplying the University of Antwerp CFSF Maple dataset, and Adri Olde Daalhuis for discussions related to complex multivalued functions.

References

1. DLMF Standard Reference Tables. http://dlmftables.uantwerpen.be/. Joint project of NIST ACMD and U. Antwerp's CMA Group, Seen June 2017
2. Backeljauw, F., Cuyt, A.: Algorithm 895: a continued fractions package for special functions. Trans. Math. Softw. **36**(3), Art. 15, 20 (2009). Association for Computing Machinery
3. Cohl, H.S., Schubotz, M., McClain, M.A., Saunders, B.V., Zou, C.Y., Mohammed, A.S., Danoff, A.A.: Growing the digital repository of mathematical formulae with generic LaTeX sources. In: Kerber, M., Carette, J., Kaliszyk, C., Rabe, F., Sorge, V. (eds.) CICM 2015. LNCS, vol. 9150, pp. 280–287. Springer, Cham (2015). doi:10.1007/978-3-319-20615-8_18
4. Cuyt, A., Petersen, V.B., Verdonk, B., Waadeland, H., Jones, W.B.: Handbook of Continued Fractions for Special Functions. Springer, New York (2008). With contributions by Franky Backeljauw and Catherine Bonan-Hamada, Verified numerical output by Stefan Becuwe and Cuyt
5. Olver, F.W.J., Olde Daalhuis, A.B., Lozier, D.W., Schneider, B.I., Boisvert, R.F., Clark, C.W., Miller, B.R., Saunders, B.V. (eds.) NIST Digital Library of Mathematical Functions. http://dlmf.nist.gov/. Release 1.0.15 of 01 June 2017
6. Weisstein, E.: eCF Encoding Continued Fraction Knowledge in Computational Form. http://www.wolfram.com/broadcast/video.php?c=385&v=1342. Seen June 2017
7. Ginev, D.: LaTeXML-Plugin-MathSyntax. https://github.com/dginev/LaTeXML-Plugin-MathSyntax/. Seen June 2017
8. 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., Vorkoetter, S.M.: Maple Programming Guide. Maplesoft, a division of Waterloo Maple Inc. (2016)
9. Maplesoft. OpenMaple API, Seen March 2017. https://www.maplesoft.com/support/help/Maple/view.aspx?path=OpenMaple. Since Maple 9, Seen June 2017
10. Maplesoft. Produce output suitable for LaTeX2e. http://www.maplesoft.com/support/help/Maple/view.aspx?path=latex. Since version Maple 18

11. Miller, B.R.: Drafting DLMF Content Dictionaries. Talk presented at the OpenMath Workshop of the 9th Conference on Intelligent Computer Mathematics, CICM 2016 (2016)
12. Miller, B.R.: `LaTeXML`: A LaTeX to XML converter. http://dlmf.nist.gov/LaTeXML/. Seen June 2017
13. Miller, B.R., Youssef, A.: Technical aspects of the digital library of mathematical functions. Ann. Math. Artif. Intell. **38**(1–3), 121–136 (2003)
14. Pagel, R., Schubotz, M.: Mathematical language processing project. In: CICM Workshops. CEUR Workshop Proceedings, vol. 1186. CEUR-WS.org (2014)
15. Wolfram Research. Generating and Importing TeX. https://reference.wolfram.com/language/tutorial/GeneratingAndImportingTeX.html
16. S. Wolfram. Computational Knowledge of Continued Fractions. http://blog.wolframalpha.com/2013/05/16/computational-knowledge-of-continued-fractions. Seen June 2017
17. Schubotz, M., Grigoriev, A., Cohl, H.S., Meuschke, N., Gipp, B., Youssef, A., Leich, M., Markl, V.: Semantification of identifiers in mathematics for better math information retrieval. In: The 39th Annual ACM Special Interest Group on Information Retrieval, Pisa, Tuscany, Italy (2016)
18. Schubotz, M., Wicke, G.: Mathoid: robust, scalable, fast and accessible math rendering for wikipedia. In: Watt, S.M., Davenport, J.H., Sexton, A.P., Sojka, P., Urban, J. (eds.) CICM 2014. LNCS, vol. 8543, pp. 224–235. Springer, Cham (2014). doi:10.1007/978-3-319-08434-3_17
19. Youssef, A.: Part-of-math tagging and applications. Submitted to the 10th Conference on Intelligent Computer Mathematics (CICM 2017), Edinburgh, Scotland, July 2017