JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 36, 1021-1034 (2020)
DOI: 10.6688/JISE.202009_36(5).0006

Rule Based Conversion of LaTeX Math Equations into Content MathML (CMML)

SHARAF HUSSAIN, SAMITA BAI AND SHAKEEL KHOJA
Faculty of Computer Science
Institute of Business Administration
Karachi, 75270 Pakistan
E-mail: {shussain; sbai; skhoja}@iba.edu.pk

This paper discusses the formation of math grammar rules for LaTeX math equations. These rules are used to generate Abstract Syntax Tree (AST) which extracts structural information from mathematical expressions given in LaTeX format. Later AST is used to generate XML structure of mathematical expressions that make mathematical expressions machine-readable in heterogeneous environments. A rule-based algorithm is also proposed that converts LaTeX math expressions into Content MathML (CMML), which produces semantic enrichment in web documents. The rules for writing LaTeX math equations are formulated and implemented as LaTeX Math Grammar (LMG), which are used for generating AST. Further, AST is converted into XML structure which is used to generate CMML encoding. Initially, the conversion algorithm is tested on 20 equations used in an NTCIR-12 math competition, then the algorithm is tested on NTCIR-12 Wikipedia-MathIR and ArXiv data sets. The results show that our algorithm is capable of converting LaTeX complex equations into CMML extensively as compared to the existing ones as well as its time efficiency is better than contemporary systems.

Keywords: LaTeX AST, XML tree, math grammar, CMML conversion, semantic analyzer

1. INTRODUCTION

The Internet has become the largest source of information. It facilitates different types of data in varying formats including text, images, videos, voice and math equations. This phenomenon has created a need for efficient searching capabilities for all types of data formats. However, the existing techniques to query data available on the Internet are limited and insufficient, as the expectations of users' search experience have increased over the time. The typical text-based (or keyword-based) search for scientific material is not just what the users want but they also require proper mechanisms to perform a meaningful and contextual search. W3C has developed different markup languages to make web content meaningful and machine-understandable. Math Markup Language (MathML) is one such effort to make math equations machine-readable.

Received August 7, 2019; revised September 28, 2019; accepted November 12, 2019.
Communicated by Filbert Hilman Juwono.

In this paper, we are focusing on semantic enrichment of math equations in the web documents by converting them into MathML format. W3C has developed MathML which is an extension of Extensible Markup Language (XML) for publishing math equations. Moreover, MathML is an ISO standard (ISO/IEC DIS 40314) since 2015 [1] which contains two lexis for writing math equations *i.e*. Presentation MathML (PMML) and Content MathML (CMML). PMML emphases on the layout of the math equations, while CMML deals with the semantics or meaning of the math equations. It also includes a parallel markup language that provides a way of writing math equations in two or more markup trees for the same math equation. It means a math equation can be written in PMML and CMML simultaneously [2].

The modern math retrieval systems formerly called Math Information Retrieval (MIR) systems use MathML for storing and searching math equations on the Internet. Writing math equations in MathML has enormous advantages, such as its CMML encoding clarifies the context of math equations for machine processing and its PMML encoding render math equations in a clear and smooth format. However, most of the scientific documents available on the Internet are in TEX/LATEX format since the authors of scientific research papers prefer to use this script for writing math notations for dissemination purposes. This creates a gap between the existing MIR systems and the available scientific documents containing math equations and notations in TEX/LATEX. To fill the existing gap, there is a need for conversion/transformation tools for converting old documents into the format which contains MathML support.

In this paper, we are introducing an algorithm for converting LATEX math equations into CMML format by using LATEX Math Grammar (LMG). The LMG is developed according to the standard rules of writing math equations [3]. The LMG is written in ANTLR4[1].

For experimentation and evaluations, we have selected 20 equations from NTCIR-12 Wiki formula queries and later we have performed detailed experiments on NTCIR-12 Wikipedia[2] and ArXiv[3] data sets. Wikipedia corpus contained more than 547 million equations of 27835 web pages, while the ArXiv corpus contained 721814 equations of 2653 TEX documents.

The rest of the paper is organized as follows: Section 2 discusses the literature review in the field of writing and transforming math equations into different formats. Section 3, proposes algorithms for developing LATEX AST and CMML encoding, Section 4, explains the experimental results, and Section 5 contains the conclusion of the paper.

2. LITERATURE REVIEW

LATEX is a professional-quality typesetting scripting language. It is the most preferred language for publishing technical and scientific documentation. The LATEX language is a mixture of content and presentation. It is extensible through macro definitions [4], therefore, it is very difficult to extract significant information of math formulae from the LATEX documents. TEX/LATEX math formulae are not suitable for automated machine

[1]https://www.antlr.org/

[2]http://ntcir-math.nii.ac.jp/

[3]https://archive.org/details/arXiv_src_0310_001

processing and sharing math information among heterogeneous systems as they are for typesetting only, and are required to be converted into machine-readable formats such as MathML (*e.g.* CMML) or OpenMath [5].

The Content Math Markup Language (CMML) is developed by W3C to provide an explicit encoding of the underlying mathematical structure of expression, rather than any particular rendering for the expression [6]. Several MIR systems are using CMML for indexing and retrieving math equations, some of them are discussed below.

MCAT is a math-aware full-text search engine which indexes math formulae in PMML/CMML for retrieval of math formula contents [7]. It uses the order of occurrence of PMML/CMML elements for storing equations into the index.

MIaS is a MIR system that indexes math equations, both in PMML and CMML [8], It also performs canonicalization (*e.g.* the standard form of math equation) and unification on PMML/CMML encoding before indexing them. The MathWebSearch is a Math formula search engine that stores math equations into Substitution Tree(ST) [9], using CMML encoding for storing math formulae into the ST index [10]. Tangent math is another tree-based math search engine which stores CMML encoding of math formulae into Symbol Layout Tree(SLT), where symbols are mapped with their layout to the occurrence of constants, operators, variable and relationship [11]. The skeletal component of a MIR system is its Math Language Processing (MLP) unit, which is required for the computers to understand mathematical expressions. It is a very difficult task for computers to comprehend how humans perceive mathematical expressions. This difficulty can be removed by writing proper grammar for mathematical expressions. Therefore, we need a set of well-defined rules for writing math expressions. This grammar may also

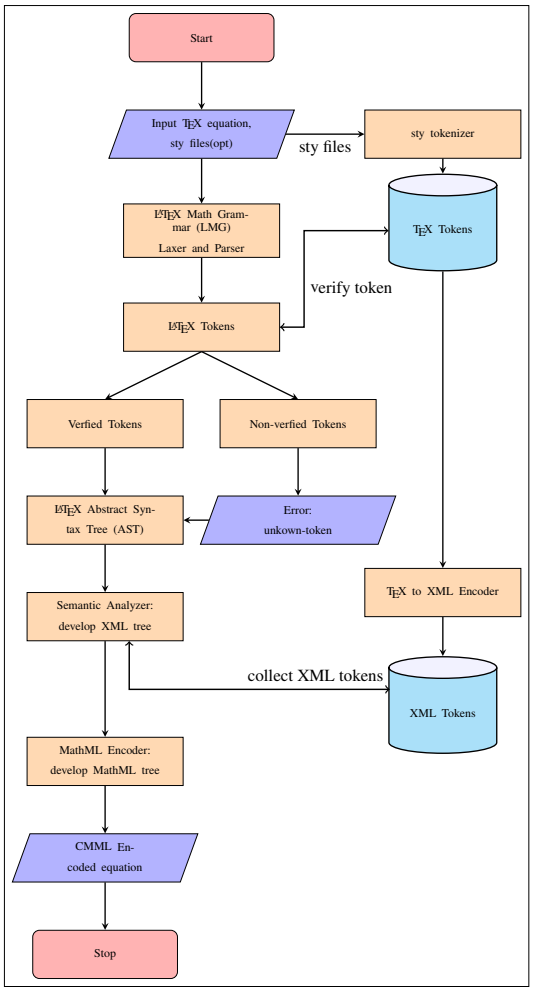

Fig. 1. Process flow of LATEX to MathML conversion.

provide help to encode math expressions into markup language (*e.g.* CMML). The MLP is also crucial for math knowledge management, math knowledge discovery, and machine learning processing [12, 13].

Algorithm 1: LaTeX AST

```
   Purpose  : Generate a Abstract Syntax Tree (AST)
              of a LaTeX Equation
   Input    : A LaTeX Equation (LE)
   Output   : Abstract Syntax Tree (AST)
1  ParseTree ← empty
2  getRelation (LE)
3  return ParseTree
4  Function getRelation (le) :
5      if le.isExpr()==True then
6          getExpr (le.expr());
7          else
8              relation =le.getRelName();
9              ParseTree.addNode(relation);
10             getRelation (le.getLHS);
11             getRelation (le.getRHS);
12         end
13     end
14 Function getExpr (expr) :
15     if expr.isAtom()==True then
16         ParseTree.addNode(expr);
17         else if (expr.isScript()==True) then
18             script = expr.getScriptName();
19             ParseTree.addNode((script));
20             getRelation (expr.getBase());
21             getRelation (expr.getScript());
22         end
23         else if (expr.isArthmaticOp()==True) then
24             OpName = expr.getOpName();
25             ParseTree.addNode(OpName);
26             getRelation (expr.getLHS());
27             getRelation (expr.getRHS());
28         end
29         else if (expr.isRow()==True) then
30             FnRowAdd (expr)
31         end
32         else if (expr.isFunction()==True) then
33             FnName = expr.getFnName();
34             ParseTree.addNode(FnName);
35             if FnName= bmatrix | pmatrix | vmatrix |
                 cases then
36                 FnMatrix (expr) ;
37                 else
38                     FnEquation (expr) ;
39                 end
40             end
41         end
42         else
43             Print(Equation Error);
44         end
45     end
46 Function FnEquation (expr) :
47     FnArgsCount = FnName.getArgSize();
48     if FnArgsCount = 0 then
49         if FnName.symbole()==True then
50             symbole=FnName.convert();
51             getExpr (symbol);
52         end
53         else
54             for i=1 to FnArgsCount do
55                 getRelation (expr.getFnArg(i));
56             end
57         end
58     end
59 Function FnMatrix (expr) :
60     RowCount = expr.getRowSize(expr);
61     for i=1 to RowCount do
62         ParseTree.addNode(getRelation
             (expr.getRow(i)));
63     end
64 Function FnRowAdd (expr) :
65     RowElements = expr.rowElementsCount();
66     for i=1 to RowElements do
67         ParseTree.addNode(getRelation
             (expr.getRowElement(i)));
68     end
```

Several transformation tools have been developed for converting TeX/LaTeX documents into MathML as described in the W3C literature[4]. However, some tools are made to convert LaTeX equations into PMML only. Math2Web[5], mathematical[6],

[4]https://www.w3.org/wiki/Math_Tools

[5]http://www.mathtoweb.com/cgi-bin/mathtoweb_home.pl

[6]https://github.com/gjtorikian/mathematical

latex2mathml[7], mathconverter[8], TeXZilla[9], and Mathoid[10].The following tools convert LaTeX equations into CMML, however with certain limitations as described below:

`latexml` system consists of two Perl scripts `latexml` and `latexmlpost`. `latexml` converts TeX documents into XML format, and `latexmlpost` is a post-processor script that converts XML documents into variety of other formats including HTML and XHTML. `latexml` system can convert LaTeX math equations into PMML, CMML, and parallel markup. `latexml` system uses `latexml` bindings for the semantic macros in LaTeX packages. The processing speed of this system is very slow as a new process is required to reload all bindings and Perl modules for each document [14].

TeX4HT is a tool that converts TeX documents into (X)HTML format [15]. It is a dynamically configurable TeX-based publishing system specially designed to produce HTML documents and converts math equations into PMML only.

TRALICS is a tool that converts LaTeX documents into XML format [16]. It converts LaTeX math equations into PMML only making it appropriate for publishing HTML and PDF documents, but cannot be used for exchanging the information of math equations among heterogeneous environments.

Ttm[11] and Hermes[12] are some other tools but are used to generate XML and PMML formats. However, they have been deprecated over time due to their limited use.

Hence, it is clear that a tool to convert LaTeX into CMML format is essentially required to use mathematical knowledge for expert systems. Therefore, we propose a rule-based system that converts LaTeX equations into CMML format which produces unambiguous output in a small duration as compared to existing systems. Since the system is rule-based, therefore malformed or deformed expressions will result in an error. At the moment our system does not have any mechanism to auto-correct such expressions.

3. LaTeX TO CMML CONVERSION PROCESS

As shown in the flowchart of Fig. 1, the conversion process starts by giving a LaTeX equation and an optional **.sty** file to the processor. The processor first checks the optional .sty file that contains user-defined commands/macros which are tokenized and stored into a repository called 'TEX Tokens'. This process is discussed in detail in subsection 3.1.1. Then the LaTeX equation is parsed through a built-in LaTeX Math Grammar (LMG) parser to generate LaTeX math tokens, which are then verified against an existing token repository. Further, Abstract Syntax Tree(AST) is generated from the verified tokens, which helps in generating XML structure and MathML tree of the equation. This process is explained in subsections 3.1.2 and 3.1.3 and the complete detail of this process is explained in Algorithms 1 and 2. CMML encoding is then extracted from the MathML tree. The CMML extraction process is discussed in subsection 3.1.4.

[7]https://github.com/roniemartinez/latex2mathml
[8]https://github.com/roniemartinez/latex2mathml
[9]http://fred-wang.github.io/TeXZilla/
[10]https://www.mediawiki.org/wiki/Manual:Mathoid
[11]http://hutchinson.belmont.ma.us/tth/mml/
[12]https://www.openmath.org/meetings/bremen2003/hermes.htm

3.1 Process Flow of the Conversion Algorithm

The LaTeX to CMML conversion algorithm takes two arguments: first argument takes equation in LaTeX standard style [17], enclosed in `$...$`, `\(...\)`, `\[...\]`, or `begin{equation}...end{equation}` and the second argument which is optional, takes **.sty** files separated with commas. As we know that all LaTeX commands start with '\' followed by arguments which are enclosed in curly braces (*i.e.* { }), further the properties of these commands are provided in square braces (*i.e.* [ ]), so we need to extract LaTeX commands with their all constituents. To start the process, *LaTeX tokenizer* divides LaTeX equation into different TeX components. The TeX components are further categorized as commands, operators, characters, and numbers. Only the TeX commands (*i.e. TeX Tokens*) are stored in the *TeX Tokens* repository along with their arguments and properties specifications. All TeX tokens are verified from the *TeX repository*. If the token is not present in *TeX repository* then it is tagged with *non-verified* flag and if the token is available then it is tagged with *verified* flag. The second argument is send to *sty tokenizer*, if it is provided in the command.

3.1.1 sty tokenizer

The second argument of the algorithm which is optional takes **.sty** files in an array string separated with a comma. If the second argument is present in the command then the system will search for **.sty** files in the specified locations and send it to *sty tokenizer*, where TeX commands and macros are collected from the **.sty** files along with their parametric information and sent to *TeX Tokens* repository. The *TeX Tokenizer* checks the availability of newly coming token if the upcoming token is not present in the TeX Token repository then it will be stored in the repository otherwise it will be discarded.

3.1.2 Rules for developing LaTeX Abstract Syntax Tree (AST)

A mathematical expression is a phrase that groups together numbers, letters, operators, and functions. In mathematical expressions, numbers are considered as constant values, such as '1' is a constant value in Fig. 2. The variables in an expression are represented by letters or symbols such as '*P*,*x*,*N*,*n*' are the variables as shown in the Fig. 2. A function in mathematical expressions returns some value for given inputs, $\sqrt{25}$ is a function which returns square root of input value 25, similarly $\sum$,$\hat{}$,$\int$ are also examples of different functions. In mathematical expressions, different types of operators are used, such as arithmetic (*e.g.* $+,-,\times,/,\pm,\mp$), logical (*e.g.* $=,<,>,\leq,\geq$), boolean (*e.g.* $\wedge,\vee,\neg$), and miscellaneous (*e.g.* $\cap,\cup,\oplus,\otimes,\sqcup$) operators. As AST is required to extract the hierarchy and relations of the above mentioned entities. We use the following predefined rules for constructing AST of an equation:

1. Initially, LaTeX equation (LE) is broken by a relational (*e.g.* logical, boolean, and miscellaneous) operators into Left-Hand-Side (L.H.S) and Right-Hand-Side (R.H.S) expressions. The relational operator will be added to the parse tree as a node. If the equation does not contain any relational operator then it is considered as an expression and the equation is directly send to `getExp(expr)` function.

2. Each L.H.S and R.H.S expression of an equation are further analyzed for the relational operator. If the L.H.S/R.H.S expression does not contain any relational

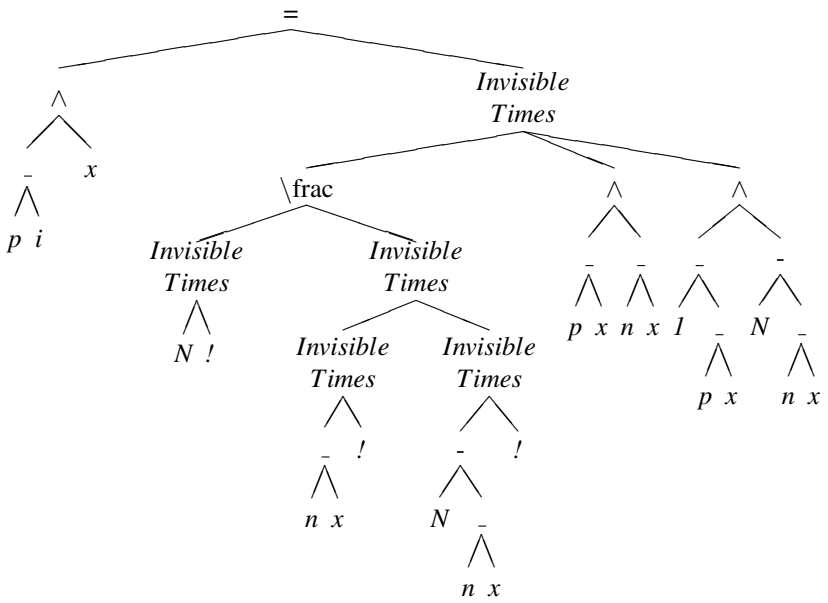

Fig. 2. AST of $\left(P_i^x = \frac{N!}{n_x!(N-n_x)!} P_x^{n_x}(1-p_x)^{N-n_x}\right)$.

operator then it is considered as an expression. Therefore, L.H.S/R.H.S expression is passed to `getExp(expr)` function.

3. The `getExp(expr)` function analyzes input expression for its type.

   3.1. If the input expression is an atom that does not contain any relational/arithmetic operator, script (*e.g.* super and sub), and LaTeX function. Then the value of the atom will be added to the parse tree as a leaf node.

   3.2. Else if the input expression contains a script (*e.g.* ˆ, _) symbol then it is added to parse tree as a node. Furthermore, the base and script components are separated and passed to `getRelation()` function for further analysis.

   3.3. Else if the input expression contains an arithmetic operator (*e.g.* $+, -, \times, /$, $\pm, \mp$ and \invisibletimes) then it is added to parse tree as a node and L.H.S and R.H.S expressions are passed to `getRelation()` function for further analysis.

   3.4. Else if the input expression is a LaTeX function (*e.g.* $\sum, \int, \surd$) then function arguments are checked. If function contains zero argument and it is a LaTeX symbol then it passes to `getExpr()` function for further analysis. Otherwise, each function argument will be separated out and again sent to `getRelation()` function for further analysis.

   3.5. Else if the input expression is a LaTeX matrix(*e.g.* bmatrix, pmatrix, vmatrix) or case statement, it passes to `FnMatrix` function where rows are separated and added to parse tree and each element of a row is sent to

`getRelation()` function for further analysis and is added to parse tree as a child of a row.

4. If the above conditions are not satisfied then an error message is generated in the log file.

As an example, we take the equation $P_i^x = \frac{N!}{n_x!(N-n_x)!} P_x^{n_x}(1-p_x)^{N-n_x}$ for elaborating the concept of LaTeX AST. Initially the equation is divided into two components because of a logical operator (*i.e.* '=') has been used. The L.H.S (*e.g.* P_i^x) does not contain any logical operator so it is sent to `getExpr()` function which constructs the L.H.S AST of '=' operator. The R.H.S (*e.g.* $\frac{N!}{n_x!(N-n_x)!} P_x^{n_x}(1-p_x)^{N-n_x}$) of '=' operator contains LaTeX function (*i.e.* \frac), arithmetic and script operators so the R.H.S AST of '=' operator is constructed according to rules mentioned earlier. The details of AST construction is described in Algorithm 1 and the constructed AST is shown in Fig. 2.

3.1.3 Semantic analyzer, XML tree

The *Semantic Analyzer* collects XML tokens from the *XML Tokens* repository which are equivalent to LaTeX tokens. In case of XML equivalent token is not present in the repository then an error token is generated for unavailable one. The XML semantic tree is generated with *XML Tokens* as defined in XML namespace as shown in Fig. 3. A Depth-First-Search (DFS) algorithm is used to replace LaTeX elements with XML elements in the LaTeX AST. The complete procedure of replacement of LaTeX tokens with XML tokens is shown in Algorithm 2.

Algorithm 2: XMLT (LT)

Purpose : Generate a XML Tree from a LaTeX AST
Input : A LaTeX Tree $LT =< V, E >$
Output : XML Tree

```
1  mark each vertex in V with 0 as mark of being "unvisited";
2  count ← 0;
3  for each vertex v in V do
4  |   if v is marked with 0 then
5  |   |   Replace (v);
6  |   end
7  end
8  Function Replace (v) :
   |   Purpose  : Visits recursively all the unvisited vertices connected to vertex v by some path and replace
   |              each vertex with XML equivalent symbol
   |   Input    : a vertex v
   |   Output   : Replace LaTeX elements with XML Element
9  |   count ← count+1;
10 |   if vertex v equivalent XML element exists then
11 |   |   replace vertex v with XML equivalent element;
12 |   end
13 |   else
14 |   |   replace vertex v with < Error > tag;
15 |   end
16 |   mark v with count;
17 |   for each vertex w in V adjuscent to v do
18 |   |   Replace (v)
19 |   end
```

3.1.4 CMML encoding

The XML encoded tree is passed to CMML encoder where necessary elements are added into it and generates a standard CMML encoded equation. For example, the `<apply>` element is used to develop an expression tree that represents a function or an operator to its arguments. The resulting tree corresponds to complete mathematical expression. Generally, this means a piece of mathematics that could be surrounded by parentheses or logical brackets without changing its meaning. The opening and closing tags of `<apply>...<\apply>` specify exactly the scope of any operator or function. Similarly, other elements of CMML are used to generate CMML encoding.

4. EXPERIMENTS AND RESULTS

To perform an experimental investigation of the conversion process from LaTeX to CMML format, the NTCIR-12 MathIR competition query set is selected. The query set contains 20 formulae from the diverse background of mathematical fields. Each math formula from the query set (*e.g.* in LaTeX format) is provided to `SnuggleTex`[13], `latexml`[14], and to our proposed system called `LMG` as an input for the CMML conversion. The `SnuggleTex` converted only 4 formulae from the query set. However, `latexml` converted 15 formulae successfully. The `LMG` converted all the formulae without producing any error or warning. According to experimental investigation, `SnuggleTex` can only perform simple and basic LaTeX formulae into CMML while `latexml` can perform a bit complex LaTeX math formulae into CMML. Moreover, `latexml` cannot perform CMML conversion if **.sty** files are used as they are not are not part of `latexml` LaTeX.pool package. The proposed rule-based algorithm (*e.g.* `LMG`) performed all conversions successfully by recognizing the inherent structure of the mathematical expressions. The LaTeX math equations used in the initial experiment and their conversion results are reported in [18].

The extensive experiment of CMML conversion is performed on NTCIR-12 Wikipedia data set, which contains 547727786 LaTeX equations in 27835 files and Arxiv[15] data set, which contains 721814 LaTeX equations in 2653 files. The conversion is performed on `latexml(version 0.8.3)` and the proposed system `LMG`, the conversion results are evaluated based on warnings and errors generated during the conversion process.

The `latexml` facilitates the error handling and categorization with the help of a comprehensive framework based on the effect produced during the conversion process. The level of conversion problems can be broken into three categories: *Warnings* that provide the information about the produced outcome that is abortive in nature, *Errors* that mostly signify more complicated problems that lead to malformed results, and *Fatals* that classify a class of errors that are quite severe and can cause processing to stop. The errors, warnings, and fatals are further categorized into problem definitions, which indicate the problem type. During the conversion process, two types of problems may arise,

[13] https://www2.ph.ed.ac.uk/snuggletex/documentation/overview-and-features.html
[14] https://dlmf.nist.gov/LaTeXML/
[15] https://archive.org/details/arXiv_src_0310_001

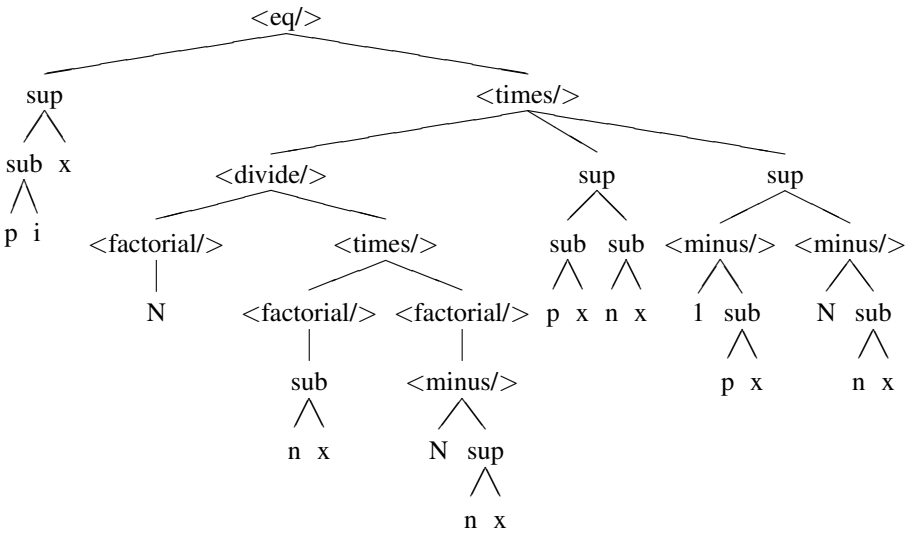

Fig. 3. XML encoded tree of $\left(P_i^x = \frac{N!}{n_x!(N-n_x)!} P_x^{n_x}(1-p_x)^{N-n_x}\right)$.

Table 1. Performance comparison on Wikipedia LaTeX equations.

	latexml			**LMG**		
	Errors	Warnings	Fatals	Errors	Warnings	Fatals
undefined	8896	0	0	1063	0	0
expected	47290	31180	1035	4015	33721	37
unexpected	125783	59643	12240	14763	4532	976
not_parsed	0	130326	0	0	0	0
missing_file	0	0	0	0	0	0
too_many_errors	0	0	2	0	0	0
latex	0	0	0	0	0	0
uninitialized	0	2133	0	0	0	0
misdefined	35578	0	0	0	0	0
malformed	141396	0	0	0	0	0

1. On account of malformed TeX input, incomplete `latexml` bindings, or bindings that are not well defined according to the TeX input, or the user-defined macros, are utilized in TeX input. These issues are reported by the system as, *undefined, ignore, expected, unexpected, not_parsed, missing_file, too_many_errors, latex, and uninitialized*.

2. On account of programming errors in the `latexml` core, or in the binding files, or the document model. These issues are reported by the system as, *misdefined, deprecated, malformed, I/O, Perl, and internal*

The `latexml` error handling framework is effective and comprehensive, therefore, it has been selected for comparing the performance of the systems `latexml` and our proposed system `LMG`. The `LMG` produced three main errors during the conversion process,

1. **undefined**: These errors are caused by undefined/unavailable TeX tokens in the token repository.

Table 2. Performance comparison on Arxiv LATEX equations.

	latexml			**LMG**		
	Errors	Warnings	Fatals	Errors	Warnings	Fatals
undefined	8550	474	0	6551	387	0
expected	981	3887986	0	763	2706315	0
unexpected	7396	1517	1	5763	1616	0
not_parsed	0	2185	0	0	47	0
missing_file	129	789	0	0	0	0
too_many_errors	0	0	115	0	0	37
latex	62	57	0	0	0	0
uninitialized	0	50673	0	0	0	0
misdefined	2866	0	4	0	0	0
malformed	3187	142	0	0	0	0

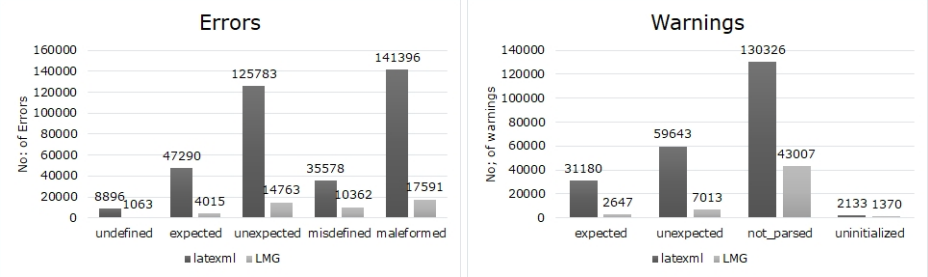

Fig. 4. Errors and warnings comparison between *latexml* & *LMG*.

2. **expected**: These errors are generated due to the missing part of LATEX tokens. The system is expected for their availability but not find in the tokens.

3. **unexpected**: These errors are evolved due to extra information attached with LATEX tokens but the system is not expected their availability in the tokens.

Tables 1 and 2 show the conversion performance of `latexml` and `LMG` systems on NTCIR-12 Wikipedia and Arxiv datasets respectively. This shows that `latexml` produced many errors and warnings during the conversion process. In contrast, `LMG` produced a few errors during the conversion process. A comparison of errors and warnings are also illustrated in Fig. 4.

The conversion time is another evaluation metric for the performance of both the systems. The `latexml` took around 112 hours to convert 547.772 million Wikipedia equations, in contrast, `LMG` took around 33.6 hours to complete this process. For the Arxiv[16] collection, which contains 721814 equations, `latexml` took 5 hours and `LMG` took 1.5 hours to complete the conversion process. Fig. 5 shows the time comparison of the conversion process between `latexml` and `LMG`, this graph also shows that the performance of the conversion process of `LMG` is better than `latexml`.

[16]https://archive.org/details/arXiv_src_0310_001

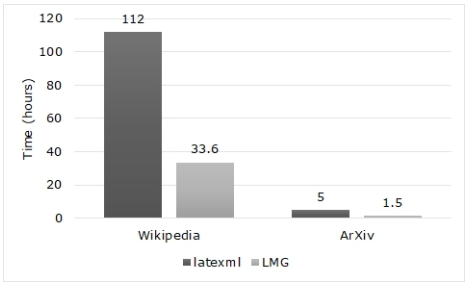

Fig. 5. Conversion time comparison between *latexml* & *LMG*.

5. CONCLUSION & FUTURE WORK

In this paper, we had tried to develop rules for extracting AST of math equations written in LaTeX. ANTLR-4 is used to write a grammar for these rules termed as LMG. This grammar is then used to generate an XML tree for MathML conversion, especially for CMML encoding. Since this conversion is strictly based on pre-defined rules, so the output is unambiguous and precise, making our process efficient and speedy as compared to the existing ones. The experimental results show that the proposed conversion process is quite adequate and time-efficient.

One area of expansion to this research could be to introduce some self-correcting mechanisms to correct malformed LaTeX expression in order to reduce undefined and unexpected errors. The LMG can also be used for Math Language Processing (MLP), which is an emerging field of Machine Learning. In the future, we plan to use LMG for learning math concepts using Machine Learning/Deep Learning algorithms, which could be beneficial for understanding, retrieving, and labeling math equations for expert systems.

REFERENCES

1. M. Schubotz, "Augmenting mathematical formulae for more effective querying efficient presentation," Ph.D. dissertation, Technical University of Berlin, Germany, 2017.
2. B. R. Miller, "Strategies for parallel markup," in *Proceedings of International Conference on Intelligent Computer Mathematics*, 2015, pp. 203-210.
3. T. Gowers, J. Barrow-Green, and I. Leader, *The Princeton Companion to Mathematics*, Princeton University Press, NJ, 2008.
4. H. Stamerjohanns, M. Kohlhase, D. Ginev, C. David, and B. Miller, "Transforming large collections of scientific publications to xml," *Mathematics in Computer Science*, Vol. 3, 2010, pp. 299-307.
5. M. Kohlhase, *OMDoc - An Open Markup Format for Mathematical Documents [version 1.2]*, Lecture Notes in Computer Science, Vol. 4180, Springer, Berlin, 2006.
6. R. Ausbrooks, S. Buswell, D. Carlisle, G. Chavchanidze, S. Dalmas, S. Devitt, A. Diaz, S. Dooley, R. Hunter, P. Ion, M. Kohlhase, A. Lazrek, P. Libbrecht, B. Miller,

R. Miner, M. Sargent, B. Smith, N. Soiffer, R. Sutor, and S. Watt, "Mathematical markup language (mathml) version 3.0," *World Wide Web - WWW*, 01 2010.
7. S. Ohashi, G. Y. Kristianto, G. Topic, and A. Aizawa, "Efficient algorithm for math formula semantic search," *IEICE Transactions*, Vol. 99-D, 2016, pp. 979-988.
8. M. Ruzicka, P. Sojka, and M. Liska, "Math indexer and searcher under the hood: Fine-tuning query expansion and unification strategies," in *Proceedings of the 12th NTCIR Conference on Evaluation of Information Access Technologies*, 2016, pp. 331-337.
9. P. Graf, "Substitution tree indexing," in *Proceedings of the 6th International Conference on Rewriting Techniques and Applications*, 1995, pp. 117-131.
10. M. Kohlhase, B. Matican, and C. Prodescu, "Mathwebsearch 0.5: Scaling an open formula search engine," in *Proceedings of the 11th International Conference on Intelligent Computer Mathematics*, 2012, pp. 342-357.
11. N. Pattaniyil and R. Zanibbi, "Combining TF-IDF text retrieval with an inverted index over symbol pairs in math expressions: The tangent math search engine at NTCIR 2014," in *Proceedings of the 11th NTCIR Conference on Evaluation of Information Access Technologies*, 2014, pp. 135-142.
12. A. Youssef, "Part-of-math tagging and applications," in *Proceedings of the 10th International Conference on Intelligent Computer Mathematics*, 2017, pp. 356-374.
13. A. Youssef and B. R. Miller, "Deep learning for math knowledge processing," in *Proceedings of the 11th International Conference on Intelligent Computer Mathematics*, 2018, pp. 271-286.
14. H. Stamerjohanns, D. Ginev, C. David, D. Misev, V. Zamdzhiev, and M. Kohlhase, "Mathml-aware article conversion from latex, acomparison study," in *Proceedings of Workshop Towards Digital Mathematics Library*, 2009, pp. 109-120.
15. G. Cevolani, "Introduzione a tex4ht," in *Proceedings of Italian TUG Meeting*, 2004.
16. J. Grimm, "Producing mathml with tralics," in *Towards a Digital Mathematics Library*, Masaryk University Press, Paris, 2010, pp. 105-117.
17. D. E. Knuth, *The TeXbook*, Addison-Wesley Professional, MA, 1984.
18. S. Hussain, S. Bai, and S. Khoja, "Content Mathml(CMML) conversion using Latex Math Grammar(LMG)," in *Proceedings of the 7th International Conference on Smart Computing and Communications*, 2019, pp. 1-5.

Sharaf Hussain is pursuing his Ph.D. in the field Computer Science from the Institute of Business Administration (IBA), Karachi, Pakistan. His core research area includes information retrieval in general and mathematical information retrieval on web in particular.

Samita Bai is currently pursuing Ph.D. degree in the field of Computer Science from Institute of Business Administration (IBA) Karachi, Pakistan. Her core research area includes semantic web in general and linked data query execution strategies in particular.

Shakeel Khoja is a Professor at Faculty of Computer Science, IBA and a Commonwealth Academic Fellow. He read for his Ph.D. and Postdoc at School of Electronics and Computer Science, University of Southampton, UK. His research work includes the development of E-learning frameworks, digital libraries, content and concept based browsing, and application of multimedia tools over the web.