Semantic Tagging of Mathematical Expressions

Pao-Yu Chien and Pu-Jen Cheng
Department of Computer Science and Information Engineering
National Taiwan University, Taiwan
b97901186@gmail.com, pjcheng@csie.ntu.edu.tw

ABSTRACT

Semantic tagging of mathematical expressions (STME) gives semantic meanings to tokens in mathematical expressions. In this work, we propose a novel STME approach that relies on neither text along with expressions, nor labelled training data. Instead, our method only requires a mathematical grammar set. We point out that, besides the grammar of mathematics, the special property of variables and user habits of writing expressions help us understand the implicit intents of the user. We build a system that considers both restrictions from the grammar and variable properties, and then apply an unsupervised method to our probabilistic model to learn the user habits. To evaluate our system, we build large-scale training and test datasets automatically from a public math forum. The results demonstrate the significant improvement of our method, compared to the maximum-frequency baseline. We also create statistics to reveal the properties of mathematics language.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

Keywords

Mathematics language processing; Semantic enrichment; Semantic tagging

1. INTRODUCTION

As mathematics-related materials on the web surge rapidly, mathematics language processing (MLP) becomes an important issue. Like what nature language processing (NLP) does to text documents and words, MLP should help computers understand the meaning of mathematical expressions and tokens for further retrieval such as mathematical search and mathematics-answering systems.

Besides being displayed as images directly, there are two types of mathematical expression representations: content-based and presentation-based. The former, such as *Content MathML* and *OpenMath*[1], encodes the syntax tree of an expression as well as the semantics of tokens into representation. The latter, such as *Presentation MathML* and LaTeX, only contains the appearance information about an expression. Figure 1 illustrates an example of these representations. Although the semantic information in content-based representation benefits MLP, its tree structure and tedious tags are too complicated for humans. In contrast, presentation-based representation is simpler and more convenient. Consequently, most documents on the web encode expressions with presentation-based representation, including *Wikipedia*[2] and *MathOverflow*[3].

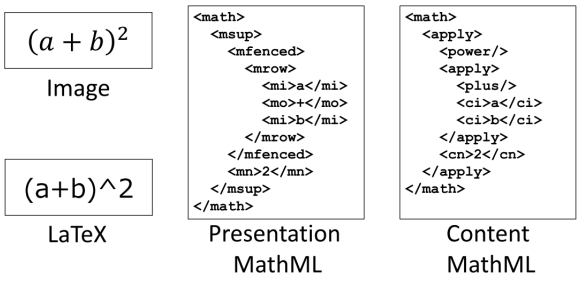

Figure 1: The same expression in different representations. The presentation-based methods use token "^" and "<msup>" to indicate that "2" is a superscript, while the token "<power/>" in the content-based methods further implies that such superscript means an exponent.

The goal of *semantic tagging of mathematical expressions* (STME) is to mark the semantic tags for tokens in (presentation-based) expressions, especially variables [4] like "f" and "p". The semantic tag of a variable should show whether it denotes a number, a function, a matrix, or even more complicated data types. STME plays an important role in all MLP tasks dealing with presentation-based expressions.

However, the state-of-the-art STME algorithms are not good enough. For instance, if a user types "$M^{-1} = M^T$" in Google[5] search bar, the user may be disappointed with the search results. One reason is that modern search engines

[1]http://www.openmath.org/standard/om20
[2]http://en.wikipedia.org/
[3]http://mathoverflow.net/
[4]The word *variable* has different meanings in elementary and advanced mathematics. In this work, a *variable* is simply a symbol denoting some data, which may or may not be a number.
[5]http://www.google.com/

WWW 2015, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3469-3/15/05.
http://dx.doi.org/10.1145/2736277.2741108.

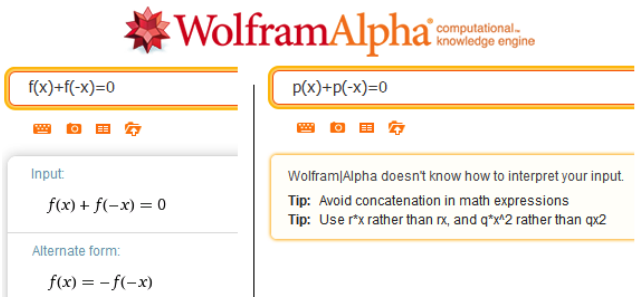

Figure 2: Resulting answers of Wolfram Alpha (on June 4, 2014)

usually ignore punctuation marks in the query, but more significant reason is that search engines cannot figure out the intent of the user: The alphabet "`M`" denotes a matrix, and "`T`" denotes the transposition operation of a matrix.

Take a returning answer of Wolfram Alpha [6] as another example. If one queries an expression "$x^2 = x(1+\sqrt{2})$" to Wolfram Alpha, it provides the plot, alternate forms, and the solutions of the equation. However, if one queries the same expression but replaces "`x`" with "`f`", Wolfram Alpha only returns an alternate form: "$f^2 - f(1+\sqrt{2}) = 0$" without further deduction. Still another example is that Wolfram Alpha returns several results to the query "$f(x) + f(-x) = 0$", but if one queries "$p(x) + p(-x) = 0$" instead, Wolfram Alpha will argue that it does not know how to interpret the input, as shown in Figure 2.

In this work, we propose a STME approach to LaTeX representations. Our solution is inspired by how people understand expressions (without explanatory text). First, the restrictions of the grammar of mathematics imply most of token semantics. The variable "`p`" in expression "$p(x+1)$" can represent either a function or a number, but not a set. In our approach, we build a context-free grammar (CFG) of mathematics that satisfies such restrictions. Parsing an expression with CFG will output all parse trees of the expression. Our self-built CFG consists of hundreds of LaTeX tokens and rules that cover most of the fundamentals of the grammar of mathematics.

Second, it is believed that a variable usually denotes the same data all the time when it appears in an expression, or even in a document. So if there is the expression "$p = 3$" directly after "$p(x+1)$", we can clearly state that both "`p`'s" represent a number, rather than a function. This property, named the *consistency property* in our work, is the main difference between STME and other tagging problems. It is also why variables are special tokens in mathematical expressions. Due to the assumption that the consistency property holds in the whole document, our approach further reduces the number of possible parse trees of expressions.

After such deduction, if there are still multiple candidates for token semantics, then the user habits are introduced to determine the most probable one. These user behaviors include not only preferred letters for specific types of variables, but also how likely a grammar rule of mathematics is applied. For instance, the token "`P`" frequently represents a polynomial in elementary mathematics, but it will be reserved for the probability function in probability theorem. The expression "$(0,1)$" is more likely to be an open interval in some fields, yet denotes a point on the plane in others. As one can see, the user habits depend on the education levels, field of study, and writing purposes of both readers and writers.

The Wolfram Alpha examples mentioned above reveal that Wolfram Alpha overfits the preferred letters such that it refuses to admit that the token "`p`" can denote the same type of data as "`f`" in the expression. In our solution, we propose a probabilistic model, which is related to *probabilistic context-free grammar* (PCFG), to learn the user habits in the targeted corpus. The probabilistic model considers preferred letters as well as the probability of applying a grammar rule. Since most of the expressions on the web are presentation-encoded, and it is also expensive and time-consuming to manually build the labelled dataset for every targeted corpus, the probabilistic model of STME should be trained by an unsupervised method.

The proposed STME approach has the following different features from existing STME-aware systems. First, we show the importance of consistency property, while other systems usually treat variables in an expression or a document independently. Second, we solve the STME problem by only using the expressions themselves. This makes our solution practical for the documents that contain only expressions like web queries or equation-based questions. Our solution demonstrates that expressions themselves contain sufficient information about their meanings. Third, we notice that user habits, which vary according to different corpora, should be considered when mining the semantics of tokens. Finally, we adopt unsupervised learning in our approach, making our approach flexible to apply on different corpora without labelled data.

We build our training and test datasets from the math forum *AOPS* [7]. Our training dataset is composed of millions of expressions written by forum users. The test dataset is generated automatically from the same corpus by mining clues from the text part of a document. We conduct an experiment to evaluate the performance of our approach and compare it with a frequency-based baseline. We show that our approach increases the accuracy of correct tagging from 87% to 94%. Error analysis of both the baseline method and ours are provided as well. Furthermore, we show several interesting statistics of the mathematics language such as the *Zipf's law* for consistent tokens. These observations will be helpful for designing MLP algorithms.

Section 2 reviews the related works of STME. Section 3 presents the STME problem and the overall framework of our approach, whose details, including preprocessing and model learning are described in Sections 4 and 5, respectively. The experimental results and in-depth analyses are given in Sections 6 and 7. Section 8 concludes the work.

2. RELATED WORK

2.1 Mathematical Search

Given a mathematical expression as a query, *mathematical search* is to retrieve documents that contain relevant expressions to the query. Mathematical search methods can be classified as presentation-based and content-based.

[6]Wolfram Alpha, an question-answering engine developed by Wolfram Research that can deal with mathematical inputs. https://www.wolframalpha.com/

[7]http://www.artofproblemsolving.com/

Presentation-based methods only consider the variables, numbers and operators appearing in an expression, with no idea about the semantic meanings behind the tokens [10, 19, 20, 7]. For example, one can treat the expression as an ordinary sentence in NLP after proper preprocessing. The technique of keyword search can then be applied.

Content-based methods, on the other hand, consider both the semantic meaning of a token, and the syntax tree of an expression [8, 1, 18, 14]. Content-based methods usually estimate the similarity of two expressions based on their syntax trees in various ways. Nghiem et al. [12] compared the performance of presentation-based and content-based methods. They concluded that the semantic information is helpful to mathematical search, especially when the query has a specific meaning.

2.2 Semantic Enrichment

Content-based mathematical search methods usually need an extra transformation from presentation representation to content representation. Such transformation, like turning Presentation MathML into Content MathML, is called the *semantic enrichment* of mathematical expressions. As one can see, the objective of semantic enrichment is similar to STME. In fact, our system can determine the most probable syntax tree of an expression.

Although semantic enrichment is important to MLP, there are few studies on semantic enrichment. SnuggleTeX[8] is a open-source toolkit that converts math mode LaTeX into Presentation MathML, and further semantically enriches it to Content MathML. Many works [14, 9] apply SnuggleTeX for semantic enrichment directly. However, SnuggleTeX is based on manually-encoded rules, instead of learning from datasets.

Grigore et al. [6] and Quoc et al. [15] both proposed approaches to mine the semantic meaning of an expression based on the surrounding text. Mining the meaning of an expression from text is impractical on expression-only documents. It does not work if the writer does not rigorously define variables in text.

Nghiem et al. [13, 11] proposed a series of semantic enrichment approaches. They trained the translation rules (that translate sub-tree of Presentation MathML to sub-tree of Content MathML) and segmentation rules automatically. However, it requires MathML parallel markup corpora, i.e., expressions encoded in both content MathML and presentation MathML, for supervised learning.

2.3 Part-of-Speech Tagging

One may find the STME problem similar to part-of-speech tagging (POST), a well-explored topic in NLP.

POST can be solved by either supervised or unsupervised methods. The state-of-the-art supervised methods reach a high accuracy while the unsupervised methods are relatively hard. Goldwater and Griffiths [5] showed that a fully Bayesian approach improves the accuracy of the Expectation Maximization (EM) algorithm with a 3-gram model. Goldberg et al. [4] showed EM can learn a good POS Tagger when provided with good initial conditions. Ravi et al. [16] further improved the accuracy to 91.6% by integer programming, which searches for the smallest model, and EM training.

[8]http://www2.ph.ed.ac.uk/snuggletex/

Table 1: A tiny example of the token set, tag set and CFG. The tag `comp` stands for complement in set theory.

W	{A,B,-,1,2}	$\bar{v}$	#Init
T	{var, set, comp, minus, number}	V	{#Init, #Exp, #SExp }
W_t	$W_{\text{var}} = \{\text{A,B}\}$ $W_{\text{set}} = \{\text{A,B}\}$ $W_{\text{comp}} = \{\text{-}\}$ $W_{\text{minus}} = \{\text{-}\}$ $W_{\text{number}} = \{\text{1,2}\}$	R	#Exp → #Exp minus var #Exp → number var #Exp → var #SExp → #SExp comp set #SExp → set #Init → #Exp #Init → #SExp
C	{A,B}		

The main difference between STME and POST is that the latter does not obey the consistency property, or at least not as intuitively [17]. Two identical words in a sentence can have distinct part of speeches, e.g. the word "saw" in the sentence "I saw a man with a saw". Moreover, the sequential model performs well in POST, but the syntax tree structure is more important to STME.

3. PROBLEM & SYSTEM FRAMEWORK

3.1 Mathematics Grammar

Due to the lack of available mathematics grammar, we build a simple context-free grammar (CFG) manually for our system. A more general and advanced grammar should be constructed for MLP in the future, while the main ideas of our approach rarely depend on the type of used grammar and can easily adapt to a more advanced one.

A CFG is denoted as $G = (V, T, R, \bar{v})$, where V is the set of **non-terminal symbols**, T is the set of **terminal symbols** (or **terminals**), R is the set of **rules**, and $\bar{v} \in V$ is the start symbol. A rule $r = v \to S$ denotes that we can substitute the sequence of symbols S for the non-terminal symbol v. Length of r is defined as the length of S. In our CFG, we ensure that r has non-zero length.

A **derivation** $\mathcal{T}$ is a series of substitutions based on R that starts at $\bar{v}$ and eventually generates a string of terminals S. In this case, we say that $\mathcal{T}$ *yields* S. We can depict a derivation as a **parse tree**. Two derivations are equivalent if they have identical parse trees.

A string of terminals S may have multiple derivations. Define $\mathcal{T}_S$ as the set of derivations that yield S. All the parse trees in $\mathcal{T}_S$ will tangle and turn into a **parse forest**. *Probabilistic context-free grammar* (PCFG) is to give a probability distribution over $\mathcal{T}_S$, so that the most probable derivation of S can be determined. Table 1 shows a tiny example of CFG. More details of our practical grammar will be given in Section 6.1.

3.2 Problem Formalization

The STME problem is defined here. Let W be the set of tokens and T be the set of tags. Each tag $t \in T$ has a **legal token set** $W_t \subset W$. In addition, there is a subset of tokens $C \subset W$ called **consistent tokens**, which should obey consistency property described later. $G = (V, T, R, \bar{v})$ is a CFG, whose terminal set is the tag set T. We assume that W, W_t, C, T and G are all known as background knowledge.

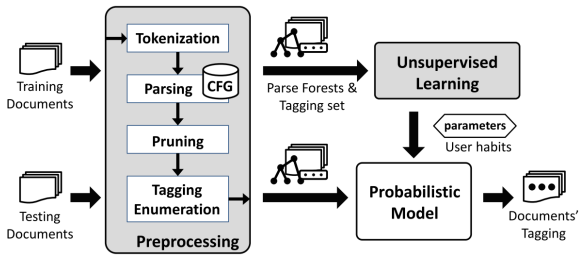

Figure 3: System Diagram of Our STEM Solution

An **expression** is a sequence of tokens. A derivation $\mathcal{T}$ is able to **produce** an expression $E = w_1 w_2 \cdots w_k$ if $\mathcal{T}$ yields tag sequence $S = t_1 t_2 \cdots t_k$, and w_i is a legal token of t_i for all i's. We say that the tag of w_i is t_i in this case.

A **document** D consists of a list of expressions $(E_1, \cdots, E_n)$. $C_D \subset C$ is denoted as the set of consistent tokens appearing in D. A list of derivations $L = (\mathcal{T}_1, \cdots, \mathcal{T}_n)$ (based on grammar G) is a **stamp** of D if it satisfies following conditions:

1. $\mathcal{T}_i$ can produce E_i.
2. **Consistency property**: each consistent token $w \in C_D$ must has its own tag, which should be the same every time it appears in D. In other words, there is a mapping $K(w) : C_D \to T$ such that the tag of every consistent token w in D must be $K(w)$. Such mapping $K(w)$ is called a **tagging** of D or the **tagging** of L. Two stamps of D may share same tagging.

The goal of SMTE is to determine the most probable stamp, or the most probable tagging, of the given document D. Table 1 shows a tiny example, where there are two derivations producing the expression "$A - B$". One yields "`var comp var`" while the other yields "`set comp set`".

3.3 System Framework

Figure 3 introduces the framework of our STME system. When receiving document D, we perform four preprocessing steps on it. First, we **tokenize** each expression in D from text into tokens. Then we **parse** expressions into parse forests based on the mathematics grammar G. Next, we **prune** the parse forests based consistency property. And finally, we **enumerate taggings** of D in order to narrow down the search space of learning.

The result of preprocessing is the parse forest of each expression, as well as a tagging pool consisting of all possible taggings. The whole preprocessing is deterministic and does not involve any machine learning.

After preprocessing, we will train a probabilistic model to score all stamps of D. The top-ranked one will be outputted as the stamp of D, and its tagging will be regarded as the most probable tagging of D. The parameters of the model are trained by unsupervised learning according to an unlabelled training dataset.

4. PREPROCESSING

4.1 Tokenization

We first tokenize every expression from a stream of text into a series of tokens. In our dataset, LaTeX is used to represent expressions. Here we state some details when tokenizing LaTeX representation. We remove tokens that are used for display and do not contain mathematical meanings, e.g., "`\left`" for changing the size of delimiter, "`\,`" for spacing, and "`\color`" for changing the color of characters. We also remove "`\text{}`" that enables user to insert text into an expression since we do not handle text contents.

Besides, we find that lots of users type the names of standard mathematics functions directly; however, these functions usually have specific tokens in LaTeX. For example, the sine function should be written as the token "`\sin`" instead of "`sin`", since the latter means multiplication of variables s, i and n from the viewpoint of LaTeX. As we will discuss in Section 7.4.3, this kind of typo violates the assumption of the consistency property. We, therefore, replace the name of the standard function with its corresponding token directly.

4.2 Parsing

Our parsing algorithm is analogous to a well-known parsing algorithm of CFG [3] with a slight modification. Given an expression $E = w_1 \cdots w_k$, the algorithm uses a boolean $\mathcal{N}(i, j, v)$ to denote if there is a derivation that starts at symbol v and yields $w_i \cdots w_j$. The base case $\mathcal{N}(i, i, v)$ is true if v is a terminal and w_i is a legal token of v. Other values $\mathcal{N}(i, j, v)$ can then be calculated by the dynamic programming technique. More specifically, we try every rule $r = v \to v_1 v_2 \cdots v_k$ to substitute for v. To yield $w_i \cdots w_j$ (after such substitution), each v_x should yield a sub-interval of $w_i \cdots w_j$ in order. Hence, $\mathcal{N}(i, j, v)$ is true if there exists a rule $r = v \to v_1 v_2 \cdots v_k$ and indexes $i = i_1 < i_2 < \cdots < i_{k+1} = j + 1$ such that all elements in the tuple

$$(\mathcal{N}(i_1, i_2 - 1, v_1), \mathcal{N}(i_2, i_3 - 1, v_2), \cdots, \mathcal{N}(i_k, i_{k+1} - 1, v_k))$$

are true. When we link $\mathcal{N}(i, j, v)$ to all tuples satisfying such condition, these $\mathcal{N}$ and links become the nodes and edges of the parse forest of E. The root of the parse tree is $\mathcal{N}(1, k, \bar{v})$.

The length-1 rules in mathematics CFG are usually used for embedding operation priority or synonym into grammar. Two derivations $v_1 \to v_2 \to v_3$ and $v_1 \to v_3$ are regarded distinct in ordinary CFG, but they usually have no difference from the viewpoint of mathematics. Some effort should be made to eliminate such false ambiguities. One possible solution is splitting links and dividing each node $\mathcal{N}(i, j, v)$ into two levels: $\mathcal{N}_1(i, j, v)$ and $\mathcal{N}_2(i, j, v)$. The node $\mathcal{N}_1(i, j, v)$ can only link to $\mathcal{N}_2(i, j, u)$ if u can be substituted for v with a series of substitutions based on length-1 rules. On the other hand, $\mathcal{N}_2(i, j, v)$ links to $\mathcal{N}_1(i_x, i_{x+1} - 1, v_x)$ just like what original $\mathcal{N}(i, j, v)$ does, except that length-1 rules are not considered here.

In practice, we also apply the technique of the *CYK algorithm* [3] to parse more efficiently. In other words, we split the rule $v \to v_1 v_2 \cdots v_k$ into $k - 1$ rules $v \to v_1 v_1^*, v_1^* \to v_2 v_2^*, \cdots$, and $v_{k-1}^* \to v_{k-1} v_k$ if $k > 2$, where v_i^* are pseudo-symbols and will not be reused in anywhere else.

4.3 Parse Forest Pruning

Now we are going to remove the derivations that violate the consistency property. Recall that for an expression $E = w_1 \cdots w_k$, $\mathcal{N}$ and their links form the parse forest of E. If a node $\mathcal{N}(i, i, t)$ is not in the parse forest, i.e., the node and the root node $\mathcal{N}(1, k, \bar{v})$ are disconnected, it implies that t cannot be the tag of token w_i.

For a document D with parse forests, we repeatedly check each consistent token w. w may appear in D multiple times. If t can be the tag for some of them but not for the others (due to the deduction from parse forest mentioned above), by consistency property we know that t must not be the tag of w. That is to say, there is no tagging K of D that satisfies $K(w) = t$. Thus we remove all nodes in parse forests representing that w's tag is t, as well as all links connected to them. The process of removing nodes goes recursively. Removing a node may cause other nodes in parse forests to have no links or to be disconnected from the parse forest. It may also eliminate candidate tags of another consistent token due to the consistency property. This process continues until all parse forests in D become stable.

4.4 Tagging Enumeration

The last step of preprocessing is to enumerate all possible taggings of the document D. This step is for reducing search space of our learning algorithm described in Section 5.3. We want to find the restriction between consistent tokens based on the consistency property again. Consider a simple document that contains one expression "$a + b$". The token "`a`" can have, says, five kinds of tags: "`num`", "`function`", "`set`", "`vector`", and "`matrix`". All these tags obey the consistency property since "`a`" only appears once in D, and so does the token "`b`". At first glance, D might have 25 taggings in total. But of course we know that only 5 of them are possible, since "`a`" and "`b`" must be compatible to be summed up.

We enumerate all tag combinations of consistent tokens as a tagging before launching the learning algorithm. By tracing the parse forests of D, we carefully verify whether there is a stamp with such a tagging. If so, this tagging will be put into a set called **tagging pool**. Every tagging in the tagging pool must belong to a stamp of D.

The time complexity of this step is exponential, and one can use a more efficient search algorithm instead of enumerating all tag combinations to expedite the process. In practice, we find that there are not too many tag combinations for a document, so the computational cost is acceptable.

5. PROBABILISTIC MODEL

After preprocessing, we obtain parse forests that embed all stamps of the document D, as well as a tagging pool consisting of all possible taggings of D. This section introduces our probabilistic model and learning algorithm.

5.1 Two Steps of Writing Expressions

If we look deeper at how people write down an expression, we can separate such process into two steps. Imagine that a person would like to define a set containing all square numbers. She may first come up with the most common format to define a set with specific condition:

$$\text{set} = \{\text{number}|\text{condition}\}$$

the condition in this case is that the number is square of another number, so the format becomes:

$$\text{set} = \big\{\text{number}|\text{number} = \text{number2}^{\wedge}\text{constant}\big\} \quad (1)$$

After the format is decided, she then chooses tokens to present each noun in Eq. (1). For example, lots of people are used to denoting two related real numbers as "`x`" and "`y`". Furthermore, people tend to use an uppercase letter to represent a set, especially "`S`" because it is the first letter of "set". Therefore, the resulting expression becomes

$$S = \big\{x|x = y^2\big\} \quad (2)$$

(We omit the condition that y should be an integer to keep the example clear).

The example shows that writing down an expression can be separated into the decisions of (1) *which format (derivation) to be used*, i.e., Eq. (1), and (2) *which token to be used*, i.e., Eq. (2). (The example is still simplified. In the real model, she has to first come up with the tag "`equal`" in Eq. (1), then realize there is only one token "`=`" that can be used to represent "`equal`". So do the other tokens). These two steps will correspond to two types of parameters involved in our probabilistic model.

5.2 Model Definition

For a document $D = (E_1, \cdots, E_n)$, we would like to find the most probable stamp L^* of D that maximizes the probability $P(L|D)$:

$$L^* = \underset{L}{\text{argmax}}\ P(L|D) \quad (3)$$

It is reasonable to regard the tagging of L^* as the most probable tagging of D. Let $L = (\mathcal{T}_1, \cdots, \mathcal{T}_n)$ be a stamp of D. According to Bayes' theorem, we have

$$P(L|D) = \frac{P(D|L)P(L)}{P(D)} \propto P(D|L)P(L) \quad (4)$$

since D is fixed. By assuming that the probabilities of expressions in D are independent, Eq. (4) becomes

$$P(L|D) \propto \prod_{i=1}^{n} P(E_i|\mathcal{T}_i)P(\mathcal{T}_i) \quad (5)$$

Based on the explanation in Section 5.1, $P(\mathcal{T}_i)$ in Eq. (5) relates to *which format to be used* and $P(E_i|\mathcal{T}_i)$ relates to *which token to be used*. Thus our probabilistic model matches the discussion in Section 5.1.

Let's focus on the conditional probability $P(E|\mathcal{T})$ first, i.e., the probability that a derivation produces the expression $E = w_1 w_2 \cdots w_k$. Denote $S = t_1 t_2 \cdots t_k$ as the tag sequence yielded by $\mathcal{T}$. We assume that the probability only depends on S and that the tokens are independent from each other. Then $P(E|\mathcal{T})$ becomes

$$P(E|\mathcal{T}) = P(E|S) = \prod_{i=1}^{k} P(w_i|t_i) \quad (6)$$

$P(w_i|t_i)$ is the first type of parameters in our probabilistic model that should be learned. It refers to the user habits that which token people prefer to use to represent such tag. For some tags, the token is unique, like the tag "`equal`" has only one token "`=`". Other tags may have multiple choices. Some of them are used more frequently than the others, like tag "`set`" and token "`S`" in the example given in Section 5.1.

Now we turn to dealing with $P(\mathcal{T})$, i.e., the probability that a derivation $\mathcal{T}$ is generated under the given grammar. Consider all substitutions in $\mathcal{T}$ and denote them as a set $R_{\mathcal{T}}$. We estimate the probability of $\mathcal{T}$ as

$$P(\mathcal{T}) = \prod_{r \in R_{\mathcal{T}}} P(r|h(r)) \quad (7)$$

where $h(r)$ is the prior knowledge for conditional probability. $P(r|h(r))$ is the second type of parameters that should be learned. A regular PCFG sets $h(r) = h_1(r) = v$ for the rule $r = v \to S$. In this case, $P(r|h(r))$ measures how frequently the rule r is used. It does not consider any dependencies between the rules. We then call $h_1(r)$ a **unigram model**.

Integrating more information into $h(r)$ results in a more advanced probabilistic model. Here we modify the idea of the bigram model to fit the tree-based structure. That is, $h(r) = h_2(r)$ is a tuple of a rule and a number (r_f, c), denoting that the substitution r is applied to the c-th outputted symbol of substitution r_f. We call $h_2(r)$ a **bigram model**. Take the expression

$$A \cup B = \{x|2|x\} \tag{8}$$

as instance. One derivation $\mathcal{T}$ for Eq. (8) is shown in the left of Figure 4. If we use the bigram model, i.e., $h(r) = h_2(r)$, the probability of $\mathcal{T}$ will be:

$$\begin{aligned} P(\mathcal{T}) =& P\left(r_{\text{equal}}|\emptyset\right) \times P\left(r_{\text{union}}|(r_{\text{equal}}, 1)\right) \times \\ & P\left(r_{\text{condSet}}|(r_{\text{equal}}, 3)\right) \times P\left(r_{\text{divisible}}|(r_{\text{condSet}}, 4)\right) \end{aligned} \tag{9}$$

In practice, the length-1 rules are frequently used for handling the priority of operations or synonyms. Such rules rarely contain any mathematical meanings. The right part of Figure 4 shows the derivation $\mathcal{T}$ for Eq. (8) by considering the length-1 rules. These rules make the bigram model meaningless. Consequently, we always ignore length-1 rules when calculating the probability of a derivation.

5.3 Obtaining the Most Probable Stamp

The next problem is how to compute Eq. (3), i.e., obtaining the most probable stamp L^* among all possible stamps of document D.

We enumerate the taggings from the tagging pool generated in Section 4.4. Once tagging K is fixed, expressions E_i in D become independent. Hence we turn to finding the most probable derivation $\mathcal{T}$ for every expression $E = w_1 \cdots w_k$ in D. Here we use dynamic programming to solve the problem. One may associate our algorithm with the *inside-outside algorithm* [2] of PCFG.

Define $\mathcal{Q}(i, j, v, r)$ as the maximum possibility among all sub-trees whose root is v and first substitution is r. And the sub-trees should be able to yield $w_i \cdots w_j$. The base case $\mathcal{Q}(i, i, v, r)$ has non-zero value $P(w_i|v)$ iff $r = \emptyset$ and either $K(w_i) = v$ for consistent token w_i or $w_i \in W_v$ for inconsistent token w_i. For $i < j$, we can calculate its value with dynamic programming. There are length-1 and length-2 rules in practice. Let $r = u \to v_l v_r$. The value of $\mathcal{Q}(i, j, v, r)$ is zero if $v \neq u$ and there is no derivation generating u from v; otherwise, $\mathcal{Q}(i, j, v, r)$ can be calculated as follows:

$$\begin{aligned} \mathcal{Q}(i, j, v, r) = \max_{k, r_l, r_r} & \left(P\left(r_l|(r, 1)\right) Q(i, k, v_l, r_l) \times \right. \\ & \left. P\left(r_r|(r, 2)\right) Q(k + 1, j, v_r, r_r)\right) \end{aligned}$$

The maximum possibility of derivations with stamp K becomes:

$$\max_{L\text{'s tagging is}K} P(L|D) = \max_{r} \mathcal{Q}(1, k, \bar{v}, r) \tag{10}$$

With dynamic programming, the most probable derivation of E can also be obtained. Combining the most probable derivations of expressions in D, it becomes the most probable stamp among all stamps having the tagging K.

5.4 Learning the Probabilistic Model

Due to the scarcity of labelled training data, unsupervised learning is more appropriate for training the probabilistic model. We implement the EM algorithm to maximize the overall probability among documents in a training dataset.

After training, a stamp will be selected and assigned to each document, and the parameters of the probabilistic model will be set as:

$$P(w|t) = \frac{n_{(w,t)} + \alpha}{n_{(t)} + \alpha|W_t|} \tag{11}$$

$$P(r|(r_f, c)) = \frac{n_{(r,r_f,c)} + \beta}{n_{(r_f,c)} + \beta|R|} \tag{12}$$

where $n_{(r,r_f,c)}$ is the number of occurrences that the substitution r is applied to the c-th outputted symbol of substitution r_f in the training data, and $n_{(w,t)}$ is the number of occurrences that the tag of w is t in the training data. α and β are constants for smoothing, $|W_t|$ is the size of legal token set of t, and $|R|$ is the number of rules.

To train the model by EM, we first randomly assign a stamp to each document. The E-step scans through all possible taggings of document D to find the most likely stamp under current parameters. The M-step re-computes the parameters based on Eq. (11), Eq. (12) and current numbers of occurrences, e.g., $n_{(r,r_f,c)}$.

6. EXPERIMENTS

6.1 Setting

Since most of the mathematical expressions are represented in LaTeX form, the token set W and grammar G in this work are constructed based on LaTeX. To the best of our knowledge, there does not exist a publicly available CFG for a mathematics grammar. Therefore, we build the CFG grammar G manually, containing 294 LaTeX tokens and 814 rules. These rules cover most of the fundamental mathematics. 109 out of 294 tokens (including English alphabets, Greek letters, and Hebrew letters) are consistent. These consistent tokens are frequently used as variables in mathematical expressions. All consistent tokens have three legal tags: "`num`", "`function`", and "`set`" in this work. Some consistent tokens have additional legal tags like the tokens "`e`" and "`i`" can be "`constant`" (i.e. Euler's number and imaginary number), and "Σ" can be the "`summation`" symbol.

6.2 Dataset

Our dataset is collected from the *Art of Problem Solving* (AOPS), a forum mainly for discussing the various problems of high school and college mathematics as well as mathematical contests. There are 56 sub-forums in AOPS (unrelated sub-forums like *Chemistry* and *Games* are ignored). We collect all topics posted before May 30, 2013. Each topic consists of a *post*, which usually contains a mathematical problem and several *replies*, which may be discussions or solutions to the problem. Both post and reply in a topic can contain text as well as mathematical expressions. We treat all expressions in a topic, including the post and the first few replies (up to 19), as a document. It leads to 227,948 documents and 3,979,588 expressions in total.

We preprocess expressions whose length is between 7 and 300, and those failing to tokenize or parse will be removed.

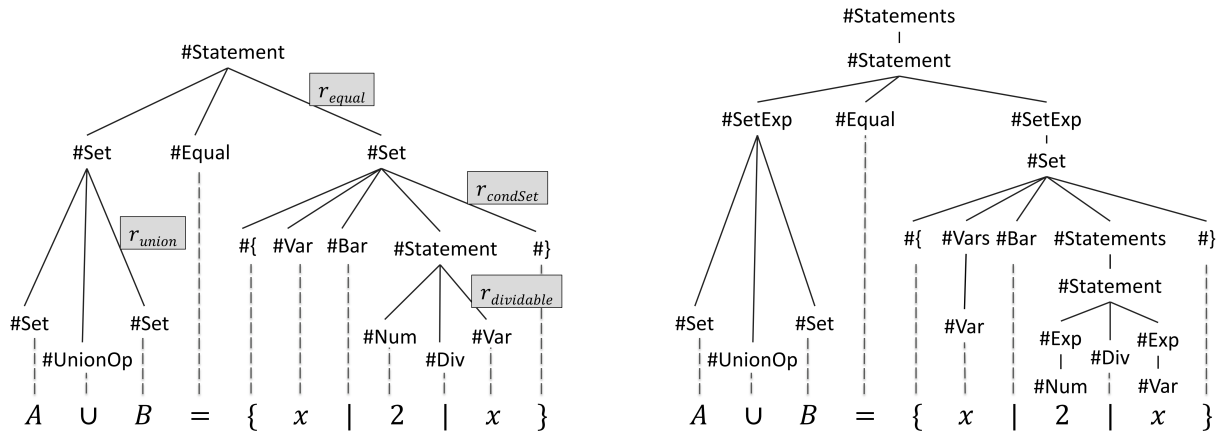

Figure 4: Derivation of Eq. (8). Left: ignore length-1 rules. Right: consider length-1 rules.

Also, documents that have no expressions left or do not have any possible taggings will be removed from the dataset. Eventually, there are 186,364 documents and 2,914,085 expressions in the dataset. The average length of expressions is 12.93. 33% of the expressions have more than one derivation. 53% of the documents have more than one possible tagging.

6.3 Test Dataset and Ground Truth

To make the size of our test dataset larger, we generate the test dataset with ground truth automatically. Let $D = (E_1, \cdots, E_k)$ be a document in the AOPS forum and D_i be the pseudo document containing only one expression E_i. Assume that D_i has multiple taggings, which means that the tags of consistent tokens in E_i are ambiguous. These tags, on the other hand, may be determined if we look back to the original document D since the consistency property makes some restrictions. We add D_i into our test dataset if D_i has multiple taggings yet D has only one tagging. The ground truth of D_i is given from the tagging of D. Note that all documents in the test data contain only one expression.

Unfortunately, the resulting test dataset has a serious bias. That is, over 97% of consistent tokens are "`num`" in the test dataset. It is because that those tokens having the tag "`num`" are much more easily determined than those having the tag "`function`" or "`set`". For example, the tag of the token "`x`" in the expression "$x + 1$" or "e^x" is definitely "`num`" (at least in fundamental mathematics). In contrast, "`function`" is the hardest tag to be determined in preprocessing. Unless clearly specified like "$f : \mathbb{R} \to \mathbb{R}$", a function in all other expressions can also be explained as a number without any conflict. As a result, an extra effort must be taken to balance the test dataset.

To do this, we seek for clues in the text part of the retrieved documents. We first list a few specific keywords for each tag. The text part of the topic is divided into *chunks* with punctuations (such as comma and period) and prepositions (such as *to, of* and *with*). If an expression consisting of only one token is included in a chunk with a specific keyword, the tag corresponding to the keyword will be directly assigned to the token. For example, consider the text "*Assume S is the non-empty set of integer that satisfies...*". Both "*set*" and "*integer*" are keywords, and their corresponding tags are "`set`" and "`num`". But the keyword "*integer*" is not in the same chunk with expression "`S`" due to the preposition "*of*". Therefore, we assign the tag of token "`S`" as "`set`" instead of "`num`".

These assignments will be used in the parse-forest-pruning step when preprocessing document D. It makes documents containing "`set`" and "`function`" more likely to be unambiguous and be added into the test dataset. Note that there are other works that semantically enrich expressions by text [6, 15], but here we only need a naive solution that is workable in our dataset. Finally, there are 20,297 documents in total in the test dataset. Among 53,493 consistent tokens in the test dataset, the percentages of "`num`", "`function`", and "`set`" are about 64%, 23%, and 12%, respectively.

In brief, we build the tagging of test data as the ground truth based on the consistency property. Other kinds of ground truth, such as the tag of non-consistent tokens or the stamp of the document, cannot be generated in a similar way, because the consistency property does not hold. Consequently, even though our method returns the most probable stamp of a document, only the performance of outputting the correct tagging (or equivalently, outputting the correct tag of consistent tokens) can be evaluated in this work.

6.4 Competitive Methods

We compare the performance of the following methods:

1. `PR`: After preprocessing, the outputted tagging is randomly chosen from all possible taggings. No model training is involved.

2. `S`: To each consistent token w, this method always outputs the most frequent tag t that w has in the training data. The method requires labelled training data. However, our dataset does not have any labels, and using labelled data from other sources will allow `S` to learn the user habits different from those of our dataset. Therefore, we allow `S` to be trained based on 10% of our (automatically labelled) test data. Learning from the test dataset makes `S` overfit the test dataset. The reported performance of `S`, as a baseline, will be higher than it should actually be.

Table 2: Accuracy of different methods that 1) output correct tagging to a document, 2) output correct tag to a (consistent) token and 3) output correct tag to each type of token

Method	Document	Token	Type of token		
			"num"	"func"	"set"
PR	0.458	0.686	0.701	0.505	0.914
S	0.876	0.880	0.899	0.853	0.780
PW	0.890	0.931	0.930	0.920	0.961
PW1G	0.935	0.962	0.989	0.901	0.944
PW2G	0.945	0.968	0.986	0.912	0.929

3. PW: This method basically follows our approach, but it only calculates $P(E_i|S_i)$ instead of $P(E_i|S_i)P(S_i)$ in Eq. (5). In other words, PW considers neither derivations nor the grammar of mathematics. It degenerates from a tree-based model into a sequential model.

4. PW1G: This method implements our approach completely, where the unigram model, i.e. $h(r) = h_1(r)$, is used in Eq. (7).

5. PW2G: Same as PW1G, this method implements our approach completely. But the bigram model, i.e. $h(r) = h_2(r)$, is used in Eq. (7).

7. EXPERIMENTAL RESULTS

7.1 Overall Performance

We calculate the accuracy of each method that 1) outputs the correct tagging of a document and 2) outputs the correct tag of a consistent token. Results are reported in Table 2.

The random method PR obtains 45.8% accuracy, showing that most documents in the test dataset have only two to three possible taggings (after preprocessing). The baseline method S, which always assigns the maximum-frequency tag to a consistent token, obtains 87% accuracy. It reveals that S is a simple solution of STME with acceptable performance. It might be the reason why available systems such as Snuggletex adopt rule-based methods as their STEM solutions. However, there are still 12% of tokens that S cannot correctly handle. Note that the performance of S for token type "set" is worse than that of the random method, meaning that the data type "set" can often be determined in preprocessing.

We test the performance of our solution with different probabilistic models. Even the weakest one (PW) beats the baseline S in both document and token accuracy. PW learns the user habit of preferred letters. By further taking the rule likelihood (i.e., the unigram and bigram models) into consideration, the accuracy increases from 89% to 94.5%. The result shows that preferred letter habits and rules likelihood are helpful to STME.

Although we cannot evaluate the accuracy of our method that outputs the correct stamp, we show that the resulting tagging of STME can help reduce the derivation ambiguity of an expression. In our dataset, there are 973,270 (33%) expressions that have more than one derivation. If the tagging of documents is decided (by our method), the proportion of ambiguous expressions decreases to 10%. The proportion of documents containing ambiguous expressions also decreases from 72% to 39%. That is, nearly two thirds of derivation ambiguities can be eliminated.

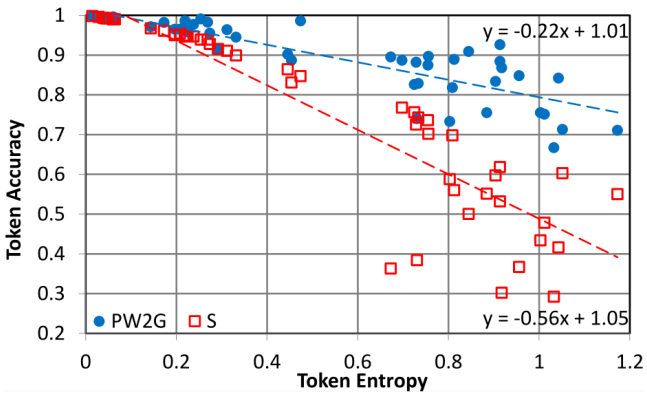

Figure 5: The relation between tag entropy and the resulting accuracy. Each point represents an English alphabet token. We also show the linear regression of both methods.

7.2 Performance of Each Token

For a consistent token w, we calculate the tag entropy of w in the test dataset to measure the diversity of w's semantic meaning, i.e.,

$$Entropy(w) = -\sum_t p_{(w,t)} \log p_{(w,t)} \quad (13)$$

where $p_{(w,t)}$ is the probability that w has tag t in the test dataset. We plot the relation between $Entropy(w)$ and the accuracy (of S and PW2G) on w in Figure 5.

Both S and PW2G perform well on low-diversity tokens. As the tag entropy increases, the semantic meaning of the token becomes diverse, making the STME problem on these tokens harder. Therefore, both methods' accuracies drop on high-diversity tokens, but S drops more seriously than PW2G. Actually, for a fixed level of tag entropy, PW2G performs strictly better than S over all tokens in that level.

7.3 Error Analysis

We observe closely the test data that our method PW2G answers correctly but S fails to tag. We briefly classify the reasons as follows:

1. The method S may return tagging that is unreasonable from the point of view of mathematics grammar. It relates to the example "$p(x) + p(-x) = 0$" of Wolfram Alpha in Section 1.

2. The method S performs poorly on high-entropy tokens. For example, S always marks tokens "p" and "h" as "num", but both of these two tokens are also used frequently as "function".

3. The method S treats a document as a *bag of tokens*, but it does not consider the relation between these tokens. Therefore, the expressions "$3(x + h)$" and "$h(x + 3)$" are identical to S.

We also observe the test data that PW2G fails to predict, and briefly classify the reasons as following:

Table 3: Examples in the test dataset. We show the actual tag t_i of token w_i in the expression, and the tags outputted by three methods `S`, `PW` and `PW2G`.

Expression	w_i	t_i	S	PW	PW2G
$B = \{1\}$	B	set	var	func	set
$p(x) + p(-x) = 0$	p	func	var	var	func
$U + V = U + W$	U	set	var	set	set
$P(x, y, z) \geq 0$	P	func	func	prob	prob
$\Phi^n = 0$	Φ	func	func	func	var
$f_n + f_{7n} = 14,379$	f	var	func	func	func

1. The tokens "P" and "E" have special tag "prob" that represents function of probability and expected value. We distinguish probability-related functions from ordinary functions because the former must use the fixed tokens "P" and "E", and have a different behavior from ordinary function. However, our method PW2G does not learn the difference of the behaviors. To make matters worse, it overfits probability-related functions too much. It almost marks "prob" to all tokens P's, which may be a normal function actually.

2. We do not have tag "matrix" in our grammar. A matrix can be operated like a number as well as a linear transformation (function). It allows a matrix "A" to be defined as a function "$A : \mathbb{R}^n \to \mathbb{R}^n$" yet multiply to a vector without parenthesis, i.e., "Ax", in a document. We find that many documents that lead PW2G to a fail are matrix-related. However, the tags of such tokens themselves are arguable.

3. In some situation, the consistency property does not hold. We will discuss this in more detail in Section 7.4.3.

We find few automatically-generated ground truth of test documents wrong. Most of them are caused by informal usage of operation. For example, lots of users in *AOPS* write function composition as "$fg(x)$", while the correct usage is "$f \circ g(x)$" or "$f(g(x))$". Since our grammar does not support the usage "$fg(x)$" for function, "f" and "g" will be recognized as "num" instead.

7.4 Issues of Mathematics Language

7.4.1 Document Frequency and Zipf's Law

Recall that the *Zipf's Law* [21] in NLP states that the frequency f of a word and its rank r follow the relationship $f = a/r^s$, where a and s are constants that vary with different languages. We are curious whether the language of mathematics also follows the Zipf's law.

We count the *document frequency* of each consistent token, i.e., the number of documents containing such a token (note that the *term frequency* of a consistent token is meaningless, since each consistent token always represents identical thing no matter how many times it appears in a document). We sort all of the consistent tokens by their frequencies, and plot the result on a *log-log graph*, as shown in Figure 6.

As one can see, the plot consists of two segments: high-frequency tokens and low-frequency tokens, with the boundary of 6,500 times. The statistical results of them are different. If we look closer at the two classes of tokens, the 43 high-frequency tokens are all English alphabet letters except "\pi" (14,704 times), obviously because π is a mathematical constant. The top-5 most frequent consistent tokens are "x", "a","n", "b", and "y". The 50 low-frequency tokens are all Greek and Hebrew letters, except for the 10 English alphabet letters (the frequencies of these 10 alphabets are from 1,353 to 6,362 times). The other consistent tokens do not appear in any documents.

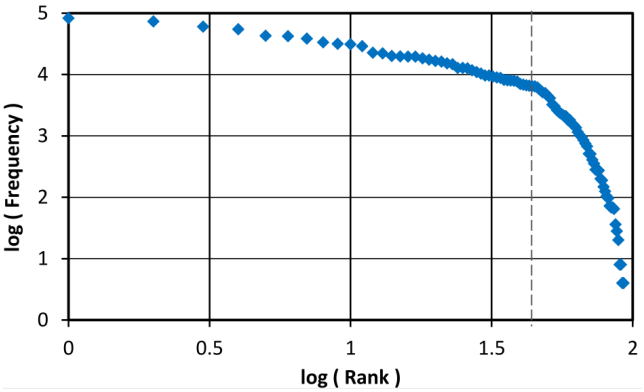

Figure 6: The document frequency of consistent tokens

The analysis shows that the usages of English alphabets and Greek letters are significantly different. People not only prefer to use English alphabets but also follow the Zipf's law with different values of a and s. Therefore, it may be proper to treat them as different language systems when processing mathematics expressions.

7.4.2 Merging Standard Function

In the first step of tokenization, we directly replace a standard function name by its corresponding token, like "log" to token "\log". It is reasonable to believe that multiplying three variables that are also assembled exactly as a standard function name rarely happens. We now demonstrate the validation of such a replacement. For a stream of characters S that form a standard function name, if more than half of the characters in S only appear in S, we believe that S is really a token of standard function; otherwise, such a replacement is doubtful. We have applied such combinations in 11,159 out of 227,948 (4.9%) documents. Only 732 out of 11,159 (6.6%) combinations are doubtful. The most common doubtful combination is "max", since "m", "a", and "x" are frequently-used tokens. Therefore, we believe the combination is valid.

7.4.3 Issue of Mathematics Language

Although mathematics emphasizes rigorous deduction, the language of mathematics is relatively loose. Mathematicians like to simplify expressions in order to clarify them, especially when the concepts become more abstruse and abstract. It highly increases the complexity of the mathematics language. Here we provide some examples found in our dataset to illustrate the limitation of our problem formalization, as well as why MLP like STME is challenging.

First, CFG cannot fully model the mathematics language. One reason is that some types of data are based on other types, like the concept of *template types* in programming

language. A token "S" can be a set of numbers, a set of pairs, or even a set of sets, so we actually learn nothing about variable "a" from expression "$a \in S$". CFG does not support template types and it is impossible to enumerate all data types into grammar rules.

Even though the types of the input and output of a function are clearly defined, it can take another type of variable as input once the meaning is somehow intuitive to everyone. For example, the function "$f(x) = x^2$" can also take a set of numbers as input and output like "$f([-1, 2]) = [0, 4]$". Although such usage is informal, it is still widely used since it is easy to understand.

Mathematicians always create new rules. Rules in specific fields are similar to the various dialects in nature languages. For example, the keyword "cyc" is frequently used for cyclic sum in high school mathematics competitions like $\sum_{\text{cyc}}(x + xy)$ equals $(x + xy) + (y + yz) + (z + zx)$. Even worse, these rules may be so local. They might only be used by an individual mathematician or appear in an article or even a proof. This flexibility makes the mathematics language too dynamic to fit any rule-based models.

Finally, the consistency property does not always hold. For example, the summation, integration or set definition can contain a **local variable**. Take "$\int_t t^2 dt$" as an example. Token "t" is only used in *integration* to represent a temporary number. One can use the same token "t" in *summation* somewhere else in the same expression. Tokens may also be confused with the characters of a function name, like the first "n" in "$sinx/n$" is a part of the sine function, while the second one is a number. If we insist the consistency of "n", we will recognize "sin" as the product of numbers "s","i", and "n". Although LaTeX has a special token "\sin" for the sine function and a token "\operatorname" for user-defined function names, lots of writers forget to use them when writing. The last example that violates the consistency property is that some variables consist of multiple tokens. One can define "p" in "$p(n)$" as a function and "$p_0 = p(0)$" as a number. Now p_0("p_0") is an atomic variable, which has different type to "p".

8. CONCLUSIONS

In this work, we propose a novel approach to labelling semantic tags to mathematical expressions. We show two key points of the STME problem: the consistency property and the user habits. The former is a special property that does not hold in common tagging problems. The latter is a common concept to NLP, but it has not been introduced to MLP in the past. Our experimental results indicate that by considering the two key points, the performance of STME can be increased significantly. Also, the existence of the Zipf's law in mathematics language reveals that it is possible to introduce the concepts and methodologies of NLP into MLP.

As for future work, it is necessary to find a more general grammar than CFG to simulate advanced mathematics grammar such as supporting template types. It is also useful if there are hierarchical tags for the same tokens, e.g., we can mark a token as a "function" as well as a "polynomial".

9. ACKNOWLEDGMENTS

This work was partially sponsored by the Ministry of Science and Technology, Taiwan, under Grant 103-2221-E-002-182.

10. REFERENCES

[1] M. Adeel, H. S. Cheung, and S. H. Khiyal. Math go! prototype of a content based mathematical formula search engine. *Journal of Theoretical & Applied Information Technology*, 4(10):1002–1012, 2008.

[2] J. K. Baker. Trainable grammars for speech recognition. *The Journal of the Acoustical Society of America*, 1979.

[3] J. Cocke and J. T. Schwartz. *Programming languages and their compilers: Preliminary notes*. Courant Institute of Mathematical Sciences, New York University, 1970.

[4] Y. Goldberg, M. Adler, and M. Elhadad. Em can find pretty good hmm pos-taggers (when given a good start). In *Proc. of Annual Meeting of the ACL*, pages 746–754, 2008.

[5] S. Goldwater and T. Griffiths. A fully bayesian approach to unsupervised part-of-speech tagging. In *Proc. of Annual Meeting of the ACL*, pages 744–751, 2007.

[6] M. Grigore, M. Wolska, and M. Kohlhase. Towards context-based disambiguation of mathematical expressions. In *Proc. of the Joint Conference of ASCM*, pages 262–271, 2009.

[7] S. Kamali and F. W. Tompa. Retrieving documents with mathematical content. In *Proc. of ACM SIGIR Conference*, pages 353–362. ACM, 2013.

[8] M. Kohlhase and I. Sucan. A search engine for mathematical formulae. In *Artificial Intelligence and Symbolic Computation*, pages 241–253. Springer, 2006.

[9] G. Y. Kristianto, M.-Q. Nghiem, and A. Aizawa. The mcat math retrieval system for ntcir-10 math track. In *Proc. of the NTCIR Conference*, pages 680–685, 2013.

[10] R. Munavalli and R. Miner. Mathfind: a math-aware search engine. In *Proc. of ACM SIGIR Conference*, pages 735–735. ACM, 2006.

[11] M.-Q. Nghiem, G. Y. KRISTIANTO, and A. Aizawa. Using mathml parallel markup corpora for semantic enrichment of mathematical expressions. *IEICE Transactions on Information and Systems*, 96(8):1707–1715, 2013.

[12] M.-Q. Nghiem, G. Y. Kristianto, G. Topic, and A. Aizawa. Which one is better: presentation-based or content-based math search? *Intelligent Computer Mathematics*, pages 200–212, 2014.

[13] M.-Q. Nghiem, G. Yoko, Y. Matsubayashi, and A. Aizawa. Automatic approach to understanding mathematical expressions using mathml parallel markup corpora. In *Proc. of Annual Conference of the Japanese Society for Artificial Intelligence*, 2012.

[14] T. T. Nguyen, K. Chang, and H. S. Cheung. A math-aware search engine for math question answering system. In *Proc. of ACM CIKM Conference*, pages 724–733, 2012.

[15] M. N. Quoc, K. Yokoi, Y. Matsubayashi, and A. Aizawa. Mining coreference relations between formulas and text using wikipedia. In *Workshop on NLP Challenges in the Information Explosion Era*, pages 69–74, 2010.

[16] S. Ravi and K. Knight. Minimized models for unsupervised part-of-speech tagging. In *In Proc. of the Joint Conference of Annual Meeting of the ACL and IJCNLP of AFNLP*, pages 504–512, 2009.

[17] A. M. Rush, R. Reichart, M. Collins, and A. Globerson. Improved parsing and pos tagging using inter-sentence consistency constraints. In *Proc. of the Joint Conference of EMNLP and CoNLL*, pages 1434–1444, 2012.

[18] K. Yokoi and A. Aizawa. An approach to similarity search for mathematical expressions using mathml. *Towards a Digital Mathematics Library*, pages 27–35, 2009.

[19] A. S. Youssef. Methods of relevance ranking and hit-content generation in math search. In *Towards Mechanized Mathematical Assistants*, pages 393–406. Springer, 2007.

[20] R. Zanibbi and B. Yuan. Keyword and image-based retrieval of mathematical expressions. In *Proc. of Document Recognition and Retrieval*, pages 1–9, 2011.

[21] G. K. Zipf. *Human behavior and the principle of least effort*. Addison-Wesley Press, 1949.