Difficulties in Handling Mathematical Expressions in Universal Dependencies

Lauren Levine
Georgetown University
lel76@georgetown.edu

Abstract

In this paper, we give a brief survey of the difficulties in handling the syntax of mathematical expressions in Universal Dependencies, focusing on examples from English language corpora. We first examine the prevalence and current handling of mathematical expressions in UD corpora. We then examine several strategies for how to approach the handling of syntactic dependencies for such expressions: as multi-word expressions, as a domain appropriate for code-switching, or as approximate to other types of natural language. Ultimately, we argue that mathematical expressions should primarily be analyzed as natural language, and we offer recommendations for the treatment of basic mathematical expressions as analogous to English natural language.

1 Introduction

Universal Dependencies (UD, Nivre et al. 2016, 2020; de Marneffe et al. 2021) is a project that aims to develop cross-linguistically consistent guidelines for multiple annotation layers, including syntactic dependency relations. Mathematical and numerical expressions comprise a particularly challenging class of cases, which require special attention to handle. Thus far, work on how to handle numerical expressions in UD has included analysis on annotating date and time cross-linguistically (Zeman, 2021), discussion of numbered entities in nominal expressions (Schneider and Zeldes, 2021), and discussion of different types of numeral related expressions in UD corpora of Uralic languages (Rueter et al., 2021).

However, there has been little discussion of how mathematical expressions, such as equations and other language which includes mathematical symbols and operators, should be handled in UD. As mathematical expressions are likely to appear as little more than edge cases in many corpus genres, this is understandable, but mathematical expressions can also feature prominently in corpora related to academic and scientific domains, such as the ACL Anthology Corpus (Rohatgi, 2022) and the academic section of the Corpus of Contemporary American English (COCA) (Davies, 2010). Unfortunately, the amount of corpora built for technical domains is limited, and the specialized nature of the language in such corpora has been a barrier in annotating them with more complex schemas, such as dependency relations. This means that there is a large gap in availability for annotated texts containing mathematical expressions that can be leveraged by NLP systems.[1]

As a result, technical texts with mathematical expressions can be viewed as a low-resource domain, and state-of-the-art systems trained on standard language will inevitably face a large drop in performance when handling such out of domain texts (Plank, 2016; Joshi et al., 2018). This is particularly an issue for real world applications of NLP technologies in technical domains, such as text mining or document processing in industrial engineering, where copious amounts of technical documents are generated by industry systems (Dima et al., 2021).

Pushing for the annotation of domain specific technical corpora will help to address this gap and provide more resources for NLP systems attempting to handle technical language. This will first require discussions on how to handle the annotation of such technical language, including standards for the handling of mathematical expressions. In this paper, we will first examine the current state of mathematical expressions in UD corpora, and then we will consider several possible approaches for handling such expressions. We

[1]While resources remain limited, we do note the release of a genre diverse UD test corpus, GENTLE, which contains dependency annotations and has a genre section for mathematical proofs: https://github.com/UniversalDependencies/UD_English-GENTLE/

will then give recommendations on how to handle the dependency relations for basic mathematical expressions, which we hope will encourage the inclusion of more mathematical texts in future annotation work.

It should be noted that while many arguments about syntactic analysis of mathematical expressions apply cross-linguistically, the focus of this paper is on mathematical expressions in English corpora, as mathematical English is the basis of most academic and professional STEM discourse, making it a logical place to start.

2 Prevalence and Existing Treatment of Mathematical Expressions in UD Data

In this section we will examine the prevalence of mathematical expressions in Universal Dependencies corpora (version 2.11),[2] as well as the distribution of dependency relations used to handle such expressions. We will also compare the prevalence of mathematical expressions in UD corpora with the prevalence of mathematical expressions in a subsection of the ACL Anthology Corpus, illustrating that expanding UD coverage more broadly into academic and technical domains would require a meaningful treatment of such expressions.

2.1 Prevalence in UD and ACL Data

In order to estimate the prevalence of mathematical expressions in UD corpora, we created a regular expression to query sentences containing combinations of numerical values and Unicode mathematical operators and symbols (a more detailed description of this query is provided in Appendix A). To determine the accuracy of this query, its performance was evaluated on a subsection of the ACL Anthology Corpus (which provides the full-text and metadata for papers and abstracts in the ACL (Association of Computational Linguistics) Anthology).[3]

From the 2021 papers in the ACL Anthology Corpus, 125 documents were randomly selected to be analyzed. The documents were sentence split and tokenized using Trankit (Nguyen et al., 2021),[4] and "gold" mathematical expressions where identified using the "formula" tag annotations included in the xml format of the corpus. After running our query on the ACL documents, the results were compared to the "gold" from the "formula" tag annotations. The resulting false positives and false negatives were then manually adjudicated for the actual presence/absence of mathematical expressions.

The performance of our query on this data sample was found to have a precision of 0.93, a recall of 0.88, and an f-score of 0.90, which we believe is accurate enough to give an estimate of the prevalence of mathematical expressions in UD corpora. However, it is worth noting that because many of the genres in the UD corpora are substantially different from the technical language in ACL papers, there is likely to be a somewhat higher proportion of false positives when we apply our query to the UD data.

Applying our query to all of the available UD corpora, we found 886 instances of sentences containing mathematical expressions, which corresponds of 0.05% of sentences in the UD corpora. These instances are spread over a total of 51 different corpora in 43 different languages, meaning that 20% of corpora and 31% of languages within UD contain some type of mathematical expression. While this may still seem like a marginal phenomenon, if we examine the prevalence of mathematical expressions by genre, as shown in Figure 1, we see that the proportion of sentences containing mathematical expressions rises to over 0.1% for several genres, including academic, legal, and medical. As data selected for UD coprora may purposely avoid difficult to annotate non-standard language such as mathematical expressions, it stands to reason that the typical proportion of mathematical expressions in these genres is likely even higher.

In order to further illustrate the genre dependent nature of the prevalence of mathematical expressions, we examined another subset of the ACL Anthology Corpus. We again used Trankit to sentence split and tokenize the 5847 paper documents from 2021 (those with both abstracts and paper bodies). Again using the "formula" tag annotations from the xml format of the corpus to identify sentences and tokens with mathematical expressions, we calculated the frequency of mathematical expressions in the data.

We found that 68% of the documents contained

[2] https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-4923
[3] https://github.com/shauryr/ACL-anthology-corpus
[4] https://github.com/nlp-uoregon/trankit

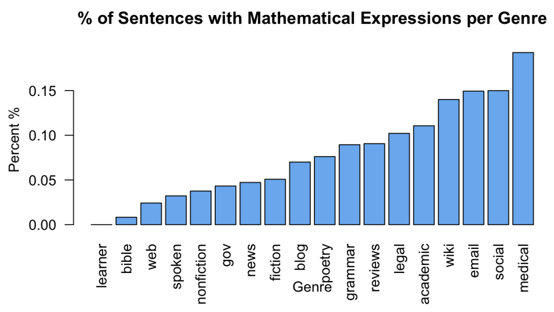


Figure 1: Proportion of sentences containing mathematical expressions for each genre category in UD.

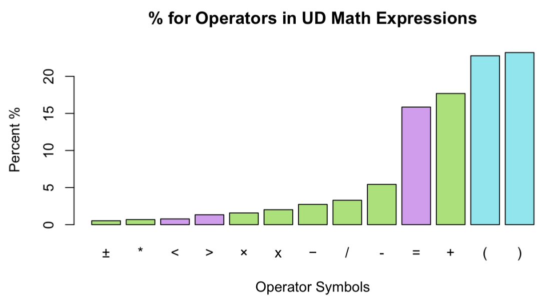


Figure 2: Frequency proportions (> 0.5%) for operators in UD mathematical expressions. (Green: Arithmetic operators, Purple: Predicate operators, Blue: Brackets.)

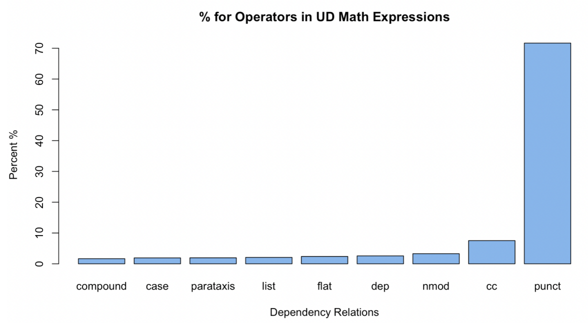


Figure 3: Dependency relation proportions (> 1%) for operators in UD mathematical expressions.

some kind of mathematical expression, that 3.7% of the sentences contained a mathematical expression, and that 4.5% of all tokens were contained within mathematical expressions. The prevalence of mathematical expressions in this data sample shows that we will need a standardized method of handling mathematical expressions if we want to expand UD corpora to cover such academic, scientific, and technical domains.

2.2 Frequencies for Mathematical Operators and Dependency Relations

We will now turn our attention to how the mathematical expressions we have identified in the UD data are currently being handled in terms of dependency relations. Searching the UD sentences we previously determined to contain mathematical expressions, we found that the expressions contained 33 unique mathematical symbols/operators, and we calculated the relative frequency of each of these symbols. The relative frequencies of some of these operators (those with relative frequency > 0.5%) are shown in Figure 2. In this Figure, we see that these operators are primarily those for basic arithmetic ("+", "-", "/", "*"), basic predicate relations ("=", "<", ">"), and parentheses ("(", ")").

Figure 3 shows the proportions of the dependency relations from all of the mathematical operators we observed in the previously identified mathematical expressions. We see that by far the most prominent relation is `punct` (punctuation), with a proportion of approximately 72%, and that `cc` (coordinating conjunction) is the next most frequent relation at 7.5%. Though `punct` is a generally uninformative dependency relation, this may not immediately strike us as an inappropriate handling of the operators we observed, considering that about 46% of them were parentheses.

However, looking at the relative frequencies of the dependency relations for the individual operators, we found that the conjoining operator "+" and the predicate operator "=" have `punct` at proportions of 32% and 41% respectively (the full cross table of mathematical operators and their dependency relation proportions is included in Appendix B). This demonstrates that in the current handling of mathematical expressions in UD, informative operators, such as "=", are frequently not analyzed meaningfully, instead being dismissed as syntactically uninformative punctuation.

3 Difficulties Presented by Mathematical Expressions

In this section, we will examine several types of mathematical expressions that present difficulties for analysis with Universal Dependencies.[5] Examples of these types of expressions were taken

[5]A list of the Universal Dependency relations discussed in this paper and their abbreviations can be found in Appendix C. A full list of UD relations can be found here: https://universaldependencies.org/u/dep/index.html

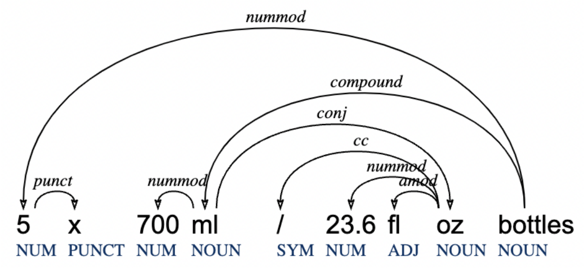

nummod
compound
conj
cc
punct
nummod
nummod
amod
5 x 700 ml / 23.6 fl oz bottles
NUM PUNCT NUM NOUN SYM NUM ADJ NOUN NOUN

Figure 4: Mathematical Expression without Predication *(Source: GUM)*

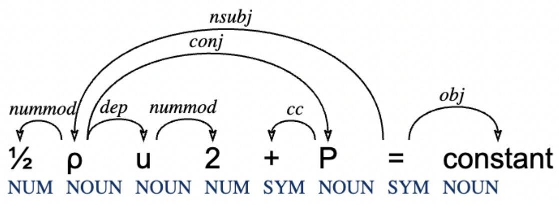


Figure 5: Mathematical Equation *(Source: GUM)*

from The Georgetown University Multilayer Corpus (GUM) (Zeldes, 2017).[6] Additional examples taken from the ACL Anthology Corpus, and the academic section of COCA[7] for each of the expression types discussed can be found in Appendix D.

3.1 Expressions without Predication

First, we will discuss mathematical expressions which lack predication. By this we mean expressions that lack a relational operator like "=" or "→", and as such could theoretically be evaluated down to a single mathematical term. Such expressions may be as simple as "3*5", but they can also become substantially more complicated (see additional examples in Appendix D.1). We single out this type of expression for discussion because it is a class of expression that can be thought of as a constituent unit, functioning essentially as a complex noun phrase. Such expressions may appear within a larger phrase of natural language, in which case they frequently occupy a syntactic position similar to that of a noun phrase.

They may also appear as individual units that can be combined to create more complicated mathematical expressions, such as equations. Because these expressions seem to act as individual units, it can be debated to what extent their internal structure should be represented. The headedness of an expression such as "8 - 6 / 2" is dubious, but it is clear that the order of operations which readers use to interpret the expression carries an understanding of hierarchy that we would want to include in a syntactic representation. Just as complex noun phrases receive internal syntactic analysis, we should also strive to provide an analysis to the internal structure of mathematical expressions without predication.

[6]`https://github.com/amir-zeldes/gum`
[7]`https://www.english-corpora.org/coca/`

Keeping these points in mind, we consider Figure 4, which includes an example of a mathematical "unit" expression and accompanying UD annotation taken from the GUM corpus. We see that this "unit" expression is used as a modifier to the noun "bottles", and that there is a mix of numbers, mathematical operators, and unit abbreviations. The main relation in the mathematical expression is handled with `conj` (which is reasonable), but it is the units that serve as the conjoined elements in the analysis rather than the numbers. The expression serves as an example of how in corpora we may find mathematical expressions with internal elements that we may not consider to be truly mathematical (such as units) and how they may create complications, as it is not immediately clear whether the numbers or the units should be preferred as the head in this example. Additionally, in this analysis, the mathematical operator "x" is dismissed as punctuation (in both POS and deprel) and "5" is treated as a regular counting determiner, which fails to capture how multiplication is overtly indicated in the expression.

3.2 Equations

The second type of mathematical expression we will consider is equations (see additional examples in Appendix D.2). While equations are largely composed of the sorts of predication lacking expressions that we discussed in the last section, they also contain mathematical operators, such as "<", ">", and "=", which define relationships between different expressions, and introduce a predicate structure to the main expression.

We see an example of an equation and accompanying UD annotation in Figure 5. In this example, the "=" operator is taken to be the root of the expression, which makes sense as it is proposing a relation between the elements to its left and to its right, just as the verb "equals" does in natural language. However, it is questionable whether or not the `nsubj` and `obj` relations are appropriate for

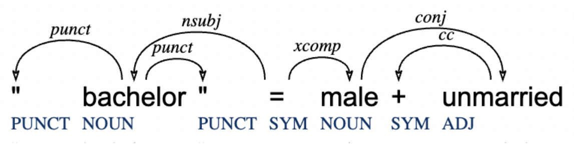

punct
nsubj
punct
xcomp
conj
cc
" bachelor " = male + unmarried
PUNCT NOUN PUNCT SYM NOUN SYM ADJ

Figure 6: Math and Natural Language Mixed Equation *(Source: GUM)*

the equals operator. We may consider the equals relation in mathematics to be similar to that of the verb "equals", which typically takes the `xcomp` relation in UD analysis rather than `obj`. In fact, we see an example of "=" being modeled in such a manner in Figure 6, which we will discuss below.

Another point worth noting in this example is that there is a tokenization issue where "u^2" is separated as "u" and "2" without any indication of how the two tokens were originally related, which creates an ambiguity which was not present in the original expression. Even knowing the intended relation, it is unclear what single dependency relation could be used to express the "to the power of" relation. While not directly part of the syntactic analysis, tokenization is a task that will inevitably have consequences on what options are available during the syntactic analysis. It is particularly important to keep in mind for equations and other mathematical expressions, which frequently have nonstandard formatting which could prompt many tokenization ambiguities and errors (further discussion of such issues can be found in Appendix E).

3.3 Math and Natural Language Mixed Expressions

Within corpora which are primarily natural language, it is possible that mathematical expressions may appear as an isolated block, entirely divorced and alien from the rest of the text. However, it is more frequent for such expressions to be integrated into the natural language of the rest of the document to various degrees. In fact, there are many instances where segments of mathematical expressions are so deeply integrated into the natural language of the surrounding text that the decision of whether or not to call them mathematical expressions at all becomes uncertain. We will refer to such instances as mixed expressions (see additional examples in Appendix D.3).

One such example is shown in Figure 6. In this example, we see mathematical operators being used convey a definition, where all of the units being related are natural language terms. Again, we see the addition operator being treated as a coordinating conjunction, though, as previously noted, this time the equals operation is treated in a syntactically different manner than we saw in the example in Figure 5. It is worth questioning whether the equals operation in these two cases is the same, in which case they should have the same analysis.

We also may question whether the units example in Figure 4 is also an example of a mixed expression. The involvement of natural language elements in both of these expressions illustrates that the line between mathematical expressions and natural language can be blurry.

4 Approaches for Handling Mathematical Expressions

Now that we've examined several examples of the types of mathematical expressions that can appear in scientific and academically oriented corpora, we will discuss various approaches we can take to analyze these expressions with dependency relations.

4.1 Multi-word Expressions

Multi-word expressions (MWEs) are expressions that are made up of multiple tokens that are considered to be syntactically idiosyncratic and can be analyzed as a single unit (Sag et al., 2002). At first glance, this may seem like a reasonable way to consider mathematical expressions, which frequently appear as analogous to a single nominal unit when they are integrated with natural language. However, simply deciding to treat mathematical expressions as analogous to MWEs would not give an immediate solution. Previous work on the handling of multi-word expressions in UD has indicated that MWEs are not treated uniformly across UD corpora, but are frequently analyzed with the relations `compound`, `fixed`, and `flat` (Kahane et al., 2017).

First, `compound` would be difficult to apply to any complicated expression consisting of more than a few terms. Additionally, `compound` implies a right headedness (in English) that is not representative of the structure of most mathematical expressions, many of which are composed of relations such as multiplication and addition, which are commutative in nature, defying the notion of headedness.

Next, `fixed` is an analysis that works for MWEs which are idiomatic set phrases that are not open to general extension. However, mathematical expressions have endless variation through the use of established operations. In short, the general notion of a mathematical expression is productive, so the application of `fixed` seems misguided. While some internal parts of a mathematical expressions could still independently be considered to be `fixed`, this would need to be considered at the level of individual languages, since many UD languages independently keep a closed list of expressions that can make use of the `fixed` relation.

Finally, `flat` completely gives up on the intent to represent the internal complexities of mathematical expressions. It it an unsatisfying simplification, but it is not without its merits. First, it requires minimal effort to implement and apply to isolated mathematical expressions, and can work as a hold over while more in depth standards for analysis are developed. However, it will not be able to account for grey areas where mathematical expressions or symbols are integrated into the surrounding natural language.

4.2 Code-Switching

Code-switching refers to a process in which two or more languages are switched between over the course of a single communication (Myers-Scotton, 2017). In Universal Dependencies, when an instance of code-switching is identified, the methods of analysis can be completely switched over from the standards of the first language to the standards of a second language using the `Foreign` feature and the `Lang` MISC attribute. As such, we could consider math to be its own completely independent language, like English or French, which deserves its own separate analysis, rather than trying to incorporate its analysis into the scope of the natural language that surrounds it.

However, if the syntax of the language is unknown, as would currently be the case with mathematical expressions, the UD guidelines recommend that the `flat` or `flat:foreign` label be used for all dependency relations in the code-switched segment (Sanguinetti et al., 2022). Such an analysis would be syntactically uninformative and would still leave open the need for developing UD standards to handle mathematical expressions.

Additionally, while there are some contexts where prolonged use of mathematical expressions may more strongly suggest that code-switching is warranted, such as multi-line proofs or derivations, we would still need a way to handle the use of mathematical operators and symbols integrated with natural language. Such instances could be considered as intra-sentential code-switching, where the switching happens within a clause or phrase, and the individual symbols could be marked with `Foreign`, but we would still need a means of determining the dependency relations needed to connect these tokens with the rest of the sentence.

Furthermore, treating math as a separate language would open up questions of when a niche domain can be considered independent enough to merit being handled though code-switching. If math can be its own language in UD, we might also extend the same consideration to domains like chemistry or computer programming which are rife with specialized jargon.

4.3 Natural Language

To treat a mathematical expression as natural language means to represent its internal structure as completely as possible with the existing relations of UD. We see evidence that mathematical expressions should be treated as analogous to natural language through the existence of mixed math and natural language expressions, such as the example in Figure 6, and through examples of mathematical expressions being integrated into passages of natural language. These examples show us that the line of what should and should not be considered a mathematical expression is not always clear. Because this line is not clear, code-switching or MWEs alone would not be sufficient to handle mathematical expressions or elements that are integrated with natural language. As such, it is worthwhile to develop standards of how to treat mathematical expressions in a manner analogous to natural language.

The most intuitive strategy for analyzing mathematical expressions as natural language is to treat the written expression the same as its spoken form. As most mathematical expressions can be verbalized in conversation, it stands to reason that we should be able to syntactically analyze them as language as well. Of course, when we verbalize mathematical expressions there may be instances where the words in the verbalized expression do not map neatly onto the written symbols, or where different speakers (particularly speakers of different languages) may not verbalize things in same way. Additionally, even once the expressions are considered in their spoken forms, it may still not be obvious which dependency relations should apply, as is often the case with technical, jargon filled natural language. As such, it is worthwhile to develop additional guidelines for the treatment of mathematical expressions as natural language.

5 Preliminary Recommendations for Analysis of Mathematical Operators as Natural Language

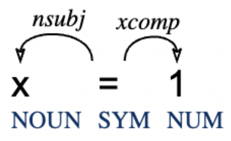
In this section we offer some brief guidelines on how to treat mathematical operators as analogous to English natural language in the application of dependency relations. As previously shown in Figure 2, mathematical operators present in UD corpora are primarily those for basic arithmetic, predicate relations, and parentheses. As such, these operators will be the focus of our recommendations. Since functions are a fundamental means of expressing relations in mathematics, we also give brief recommendations for the treatment of function application.

In these guidelines, we follow the view put forward by Schneider and Zeldes, 2021 that the relation `nummod` should be strictly used for quantity modification, as opposed to being a more general modifier to be used in any situation involving numbers. As such, we generally treat free standing numbers in mathematical expressions (e.g., "4" in "x + 4") syntactically the same as we treat variables like "x": as nominal terms.

5.1 Predicate Operators (e.g., =, <, >)

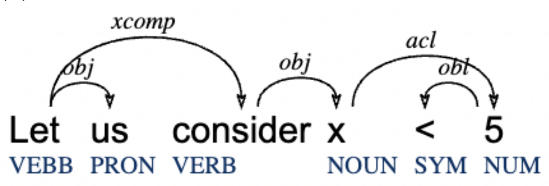
If we want to handle mathematical operators in a manner analogous to handling natural language, we may start by considering how the operators would be realized in spoken language. The predicate operator "=" is pronounced as the verb "equals" in spoken language, and it seems reasonable to treat "=" similarly. As discussed in Section 3.2, the verb "equals", is generally analyzed with arguments taking the `nsubj` relation and the `xcomp` relation. An example of such an analysis is shown below in (1) for the expression "x = 1":

(1)



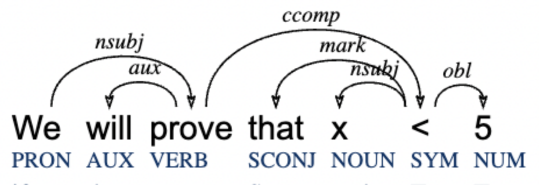
Similarly, we may consider that "<" is generally pronounced as "less than" when spoken aloud. However, we must additionally take into account that the natural language context surrounding the operator will influence what syntactic analysis we want to apply to it. For instance, in the example "Let us consider x < 5", "x < 5" is itself a term where "x" is the head and "< 5" is a modifying expression for the type of "x". In this example, we can analyze "< 5" as an adnominal clause headed by "5". The clause then functions as a modifier to the leftmost term "x", and we give its head the relation `acl`. "<" is then an extent modifer for the nominal term "5", so we give it the relation `obl`. The analysis for this example is shown in (2) below:

(2)



We may also consider the example "We will prove that x < 5", where "x < 5" is itself an equation with predication. In this instance, "<" must be the predicate and serve as the head of "x < 5". It follows that we may treat "x" as the `nsubj` of the predicate, and because the predicate relation is that of a comparative adjective, we may treat "5" as an `obl` argument to "<". The analysis for this example is shown in (3) below:

(3)

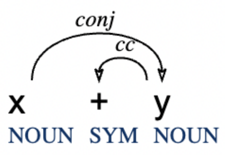


The ">" operator can be analyzed in a manner analogous to the above examples.

5.2 Conjoining Operators (e.g., +, -, *, /)

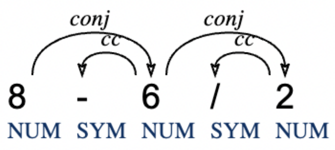
Next, we will consider the operators for the mathematical operations of addition ("+"), subtraction ("-"), multiplication ("*"), and division ("/"). It is worth noting here that some of these operations may be represented in multiple forms, not all of which will have the same syntactic analysis (e.g., in mathematical expressions, multiplication can be implied by the adjacency of terms, as well as by the use of the "*" or "x" operators). We primarily consider these operations to be conjoining relations, and as such, we link the terms to the left and right of the operator with the `conj` relation, and the operator itself can be labeled with the `cc` relation. An example of this analysis is shown in (4) for the expression "x + y":

(4)



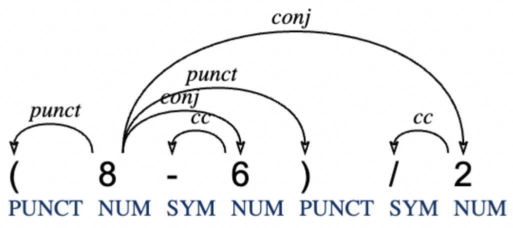
One benefit of this analysis is that it allows us to distinguish the scope of certain operations. For instance, consider the expression "8 - 6 / 2", which evaluates to 5. In accordance with the order of operations for mathematical expressions, in which division occurs before subtraction, the division by 2 should just be applied to the 6. We can express that by making "2" a dependent of "6". This analysis is show below:

(5)



In contrast, consider the expression "(8 - 6) / 2", which used parentheses to force the subtraction to occur first so the expression evaluates to 1 rather than 5. We can express this difference by making "2" a dependent of "8" rather than "6". The added parentheses are treated as punctuation to the head. This analysis is show below:

(6)



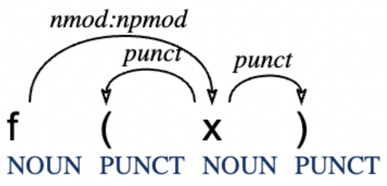
As previously mentioned, multiplication can be implied by the adjacency of terms, and in such cases, there is no operator to assign the `cc` relation. Even so, if it is two adjacent variable terms (such as "xy"), we believe it is still appropriate to apply the relation `conj`. However, if it is a coefficient adjacent to a variable, as in "2x", then we believe the coefficient can be treated as `nummod` to the variable. This is because "2x" (pronounced "two x") is generally interpreted as quantity modification on the number of "x", similar to how "5 bottles" is quantity modification on the number of bottles.

While in this section we have treated basic mathematical operators as conjoining relations, we also note that it is possible to view them as instances of more general function application (for which we offer a recommended treatment in the next section). While this is reasonable from a semantic perspective, we believe that in the formulation of mathematical expressions the verbalizations of these basic operators typically occupy syntactic positions more similar to natural language conjunctions (which themselves could be modeled as simple functions if desired), and as such can be handled using the `conj` and `cc` relations in most instances.

5.3 Function Application

We will now consider how to analyze function application in expressions such as "f(x)". This expression can be pronounced as "F of X", and so we may analyze "f" as the head of the expression, and "x" as a nominal extent modifier to the function using the `nmod:npmod` relation. The parentheses are treated as punctuation attached to "x". This analysis is show below:

(7)



6 Conclusion

In this paper, we gave an high level overview of the current treatment of mathematical expressions in UD corpora, and considered various difficulties that arise when attempting to handle mathematical expressions with dependency relations by examining different types expressions attested in corpora related to academic and scientific domains. We argued that in most cases mathematical expressions should be treated in a manner analogous to natural language, rather than being treated as multi-word expressions with minimal internal structure, or as instances of an entirely separate "language" that would be handled via code-switching. As a part of this argument, we provided guidelines for using dependency relations to analyze basic mathematical expressions as natural language.

The main purpose of this paper is to raise awareness of the problems presented by mathematical expressions, and present various alternative philosophies for how to address them. We also wish to highlight the current lack of UD resources containing mathematical and technical texts. We believe that the adoption of the philosophy to treat mathematical expressions as natural language and the further development of such guidelines will help to facilitate the inclusion of such technical texts in future UD corpora and expand the resources available for the under resourced domain of technical language.

Limitations

As previously mentioned, while many of our arguments regarding syntactic analysis of mathematical expressions can apply cross-linguistically, this paper has primarily discussed how to analyze mathematical expressions as analogous to English natural language. As we argue that mathematical expressions should be treated as natural language in general (not just English), mathematical expressions in non-English texts should be analyzed as analogous to the primary natural language used in that document. However, the recommendations in this paper focus only on English language verbalization of mathematical expressions.

Additionally, the guidelines offered here only cover basic mathematical expressions, and more substantial guidelines will need to be developed in order to inform the annotation of texts containing more complicated mathematical expressions. This paper also does not include annotations for a significant amount of data, which would be useful in demonstrating the validity of analyzing mathematical expressions as natural language. Future work will need to include the further development of guidelines and a demonstration of their application on a substantial amount of data.

References

Mark Davies. 2010. The Corpus of Contemporary American English as the first reliable monitor corpus of English. *Literary and Linguistic Computing*, 25(4):447–464.

Marie-Catherine de Marneffe, Christopher D. Manning, Joakim Nivre, and Daniel Zeman. 2021. Universal Dependencies. *Computational Linguistics*, 47(2):255–308.

Alden Dima, Sarah Lukens, Melinda Hodkiewicz, Thurston Sexton, and Michael P Brundage. 2021. Adapting natural language processing for technical text. *Applied AI Letters*, 2(3):e33.

Vidur Joshi, Matthew Peters, and Mark Hopkins. 2018. Extending a parser to distant domains using a few dozen partially annotated examples. *arXiv preprint arXiv:1805.06556*.

Sylvain Kahane, Marine Courtin, and Kim Gerdes. 2017. Multi-word annotation in syntactic treebanks - propositions for Universal Dependencies. In *Proceedings of the 16th International Workshop on Treebanks and Linguistic Theories*, pages 181–189, Prague, Czech Republic.

Carol Myers-Scotton. 2017. Code-switching. *The handbook of sociolinguistics*, pages 217–237.

Minh Van Nguyen, Viet Dac Lai, Amir Pouran Ben Veyseh, and Thien Huu Nguyen. 2021. Trankit: A light-weight transformer-based toolkit for multilingual natural language processing. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal Dependencies v1: A multilingual treebank collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1659–1666, Portorož, Slovenia. European Language Resources Association (ELRA).

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Jan Hajič, Christopher D. Manning, Sampo Pyysalo, Sebastian Schuster, Francis Tyers, and Daniel Zeman. 2020. Universal Dependencies v2: An evergrowing multilingual treebank collection. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 4034–4043, Marseille, France. European Language Resources Association.

Barbara Plank. 2016. What to do about non-standard (or non-canonical) language in NLP. *arXiv preprint arXiv:1608.07836*.

Shaurya Rohatgi. 2022. Acl anthology corpus with full text. Github.

Jack Rueter, Niko Partanen, and Flammie A. Pirinen. 2021. Numerals and what counts. In *Proceedings of the Fifth Workshop on Universal Dependencies (UDW, SyntaxFest 2021)*, pages 151–159, Sofia, Bulgaria. Association for Computational Linguistics.

Ivan A Sag, Timothy Baldwin, Francis Bond, Ann Copestake, and Dan Flickinger. 2002. Multiword expressions: A pain in the neck for NLP. In *International conference on intelligent text processing and computational linguistics*, pages 1–15. Springer.

Manuela Sanguinetti, Cristina Bosco, Lauren Cassidy, Özlem Çetinoğlu, Alessandra Teresa Cignarella, Teresa Lynn, Ines Rehbein, Josef Ruppenhofer, Djamé Seddah, and Amir Zeldes. 2022. Treebanking user-generated content: a UD based overview of guidelines, corpora and unified recommendations. *Language Resources and Evaluation*, pages 1–52.

Nathan Schneider and Amir Zeldes. 2021. Mischievous nominal constructions in Universal Dependencies. In *Proceedings of the Fifth Workshop on Universal Dependencies (UDW, SyntaxFest 2021)*, pages 160–172, Sofia, Bulgaria. Association for Computational Linguistics.

Amir Zeldes. 2017. The GUM corpus: creating multilayer resources in the classroom. *Language Resources and Evaluation*, 51(3):581–612.

Daniel Zeman. 2021. Date and time in Universal Dependencies. In *Proceedings of the Fifth Workshop on Universal Dependencies (UDW, SyntaxFest 2021)*, pages 173–193, Sofia, Bulgaria. Association for Computational Linguistics.

A Query to Identify Mathematical Expressions

This section gives a description of the search criteria in the query we used to identify mathematical expression in UD corpora. Our query identified sentences containing the at least one of following combinations of numerical values and Unicode mathematical operators and symbols:

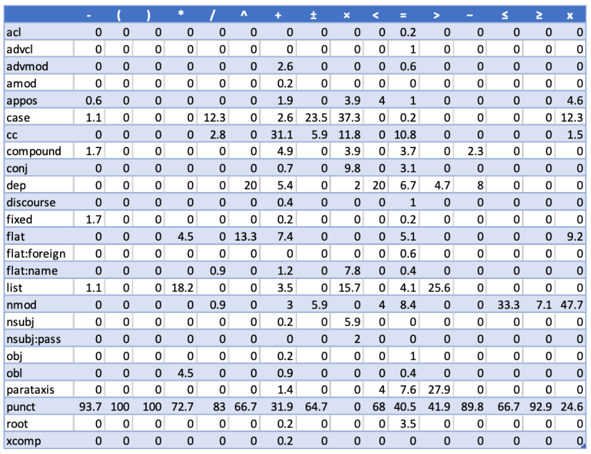
1. At least 1 token that is included in one of the following Unicode blocks:
   - Mathematical Operators
   - Supplemental Mathematical Operators
   - Mathematical Alphanumeric Symbol
2. Or, at least 2 basic mathematical operators
3. Or, at least 1 number, 1 basic mathematical operator, and 1 ambiguous mathematical operator

where "basic mathematical operators" are defined as the following set of symbols: +, ×, ÷, =, ±, >=, <=, and "ambiguous mathematical operators" are defined as the following set of symbols: -, /, <, >, x, *, ˆ, (,).

Abbreviation	**Relation**
`acl`	clausal modifier of noun
`cc`	coordinating conjunction
`compound`	compound
`conj`	conjunct
`fixed`	fixed multiword expression
`flat:foreign`	foreign words
`nmod:npmod`	NP as adverbial modifier
`nsubj`	nominal subject
`nummod`	numeric modifier
`obj`	object
`obl`	oblique nominal
`punct`	punctuation
`xcomp`	open clausal complement

Table 1: Abbreviations of dependency relations discussed in this paper.

B Dependency Relation Proportions for Mathematical Operators



	-	(	)	*	/	^	+	±	×	<	=	>	−	≤	≥	x
acl	0	0	0	0	0	0	0	0	0	0	0.2	0	0	0	0	0
advcl	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
advmod	0	0	0	0	0	0	2.6	0	0	0	0.6	0	0	0	0	0
amod	0	0	0	0	0	0	0.2	0	0	0	0	0	0	0	0	0
appos	0.6	0	0	0	0	0	1.9	0	3.9	4	1	0	0	0	0	4.6
case	1.1	0	0	0	12.3	0	2.6	23.5	37.3	0	0.2	0	0	0	0	12.3
cc	0	0	0	0	2.8	0	31.1	5.9	11.8	0	10.8	0	0	0	0	1.5
compound	1.7	0	0	0	0	0	4.9	0	3.9	0	3.7	0	2.3	0	0	0
conj	0	0	0	0	0	0	0.7	0	9.8	0	3.1	0	0	0	0	0
dep	0	0	0	0	0	20	5.4	0	2	20	6.7	4.7	8	0	0	0
discourse	0	0	0	0	0	0	0.4	0	0	0	1	0	0	0	0	0
fixed	1.7	0	0	0	0	0	0.2	0	0	0	0.2	0	0	0	0	0
flat	0	0	0	4.5	0	13.3	7.4	0	0	0	5.1	0	0	0	0	9.2
flat:foreign	0	0	0	0	0	0	0	0	0	0	0.6	0	0	0	0	0
flat:name	0	0	0	0	0.9	0	1.2	0	7.8	0	0.4	0	0	0	0	0
list	1.1	0	0	18.2	0	0	3.5	0	15.7	0	4.1	25.6	0	0	0	0
nmod	0	0	0	0	0.9	0	3	5.9	0	4	8.4	0	0	33.3	7.1	47.7
nsubj	0	0	0	0	0	0	0.2	0	5.9	0	0	0	0	0	0	0
nsubj:pass	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
obj	0	0	0	0	0	0	0.2	0	0	0	1	0	0	0	0	0
obl	0	0	0	4.5	0	0	0.9	0	0	0	0.4	0	0	0	0	0
parataxis	0	0	0	0	0	0	1.4	0	0	4	7.6	27.9	0	0	0	0
punct	93.7	100	100	72.7	83	66.7	31.9	64.7	0	68	40.5	41.9	89.8	66.7	92.9	24.6
root	0	0	0	0	0	0	0.2	0	0	0	3.5	0	0	0	0	0
xcomp	0	0	0	0	0	0	0.2	0	0	0	0	0	0	0	0	0

Table 2: Dependency relation proportions for the most frequent mathematical operators in Universal Dependencies corpora (UD version 2.11, contains 243 treebanks, 138 languages).

C Dependency Relation Abbreviations

Table 1 lists the UD dependency relations discussed in this paper. The left column lists the deprel abbreviation used in the paper and the right column gives a short description of the relation.

D Examples of Mathematical Expressions in Corpora

This section gives additional examples of mathematical expressions in corpora and includes discussion on notable aspects of each example.

D.1 Expressions Lacking Predication

Examples taken from COCA:

(1)

(n + 1)

(2)

Square root of 37

(3)

1 + 2 + 4 + 8 + 16 + 32 + 64 cents

Examples (1) and (2) above may function as independent nominal phrases, while the mathematical expression in example (3) acts as a modifier to the word "cents". Example (2) also includes an expression composed primarily of word tokens rather than symbols, which provides additional motivation for a natural language based analysis.

D.2 Equations

Examples taken from the ACL Anthology Corpus:

(1)

λ v = log (D / D(v))

(2)

β = 1 + β 2 × C × E β 2 × C + E

(3)

γ = 1/1.3 = 0.77

While examples (1) and (2) show expressions with a single predicate operator, example (3) shows that it is possible to have multiple predicate operators in a single expression. Also, we note that because "x" indicates multiplication in example (2), it is likely that "β" is squared rather than multiplied by "2", but the formatting has introduced ambiguity into the expression.

D.3 Mixed Expressions

Examples (1) and (2) taken from COCA. Example (3) taken from the ACL Corpus:

(1)

The total number of productions (unintelligible + simplified + correct)

(2)

Global fee = hospital costs + hospital profits + physician fees

(3)

king - man + woman = queen

The above examples show expressions that mix mathematical operators and words to convey various kinds of definitions. Notably, in example (1) we see parentheses serving a similar function to the equals signs in the other two examples.

E Expressions Unfit for Syntactic Analysis

We would also like to highlight that there are issues regarding how mathematical expressions are represented and tokenized in corpora, which need to be figured out before syntactic analysis can be applied. While searching for example mathematical expressions to use in this paper, we came across numerous examples where the reformatting done to import the equation into the corpus leaves it mangled and unintelligible to the extent that attempting to apply a syntactic analysis would be meaningless.

Here are example equations taken from the the ACL Anthology Corpus, which were heavily altered upon being imported into the corpus. For the sake of comparison, the original equations from the corresponding ACL papers are included directly below each equation:

(1)

) , cos(1 2 1 2 1 Σ Σ Σ = = = × × = n i n i n i bi ai bi ai B A

$$\cos(A,B) = \frac{\sum_{i=1}^{n} ai \times bi}{\sqrt{\sum_{i=1}^{n} ai^2} \times \sqrt{\sum_{i=1}^{n} bi^2}}$$

(2)

0 H : (|) (|) i i P t R p P t R = = 1 1 2
H : (|) (|) i i P t R p p P t R = $\neq$ =

$$\begin{aligned} &\mathrm{H}_0: \; P(t_i \mid R) = p = P(t_i \mid \tilde{R}) \\ &\mathrm{H}_1: \; P(t_i \mid R) = p_1 \neq p_2 = P(t_i \mid \tilde{R}) \end{aligned}$$

The examples above illustrate that importing complex mathematical expressions into corpora without taking into account how the formatting should be represented and how the expressions should be tokenized can result in expressions that cannot be interpreted and thus cannot be syntactically analyzed. While it is essential to have some means of handling ambiguous or mangled expressions in an analysis of mathematical expressions, it will also be important to consider representation and tokenization issues separately.