Analyzing Language With Math

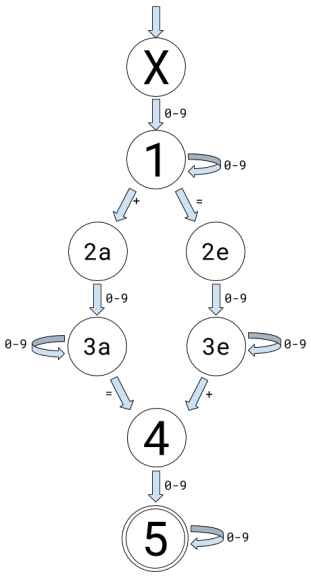
Ian Chen-Adamczyk

Introduction

Formal Languages

Many have made the analogy of mathematical notation being a language in itself. It is an understandable analogy, considering how math has an almost universal notation (though with a few differences depending on field or stylistic choice), clearly-defined symbols, and rules for using the symbols. Mathematical linguistics makes the link back, using math to describe language, instead using of language to describe math. This step is definitely complicated, considering how convoluted and illogical grammar might seem at times, or how difficult it is to put math to something so subjective as semantics. However, many mathematicians have tried, and they have been largely successful, with a huge influx of groundbreaking papers in 1961. This paper will trace through some of the research areas generated and developed under the broad umbrella of mathematical linguistics.

Formal Languages



Let's start with the basic notation used to describe formal languages. We define formal languages by an alphabet (a set of symbols, usually denoted by Σ) and the rules for the formation of the language (usually by means of regular expressions, or more generally, grammar). Symbols from the alphabet are combined to form words, and the words are considered "well-formed" if they follow the grammar of the language.

For example, we can define a formal language out of an alphabet of $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, =\}$. We can give it the rules of formation that it always contains exactly one "+" and one "=", not next to each other or at the beginning or end of a string, and the string never begins with "0". We've defined a language for addition statements by establishing a language and a grammar for addition statements. However, whether the statements are true is more complicated and falls under formal semantics.

Verifying Grammar Rules

How would one mathematically verify whether a word follows the grammar for that language? For the simpler case that the rules of the language can be expressed in a regular expression, the words can be validated by using Nondeterministic Finite Automatons (NFAs), usually represented in a diagram such as the one to the left. NFAs are defined by a finite set of states (here, $\{X, 1, 2a, 2e, 3a, 3e, 4, 5\}$), a finite set of input symbols (the word being tested), a transition function (mapping current state and the next symbol in the word to the next state, as represented by the arrows), an initial state (the X state), and a set of accepting/final states (here, $\{5\}$). If a word fulfills the rule of the language corresponding to the NFA, then inputting the word into the NFA should end with an accepting state.

Formal grammar rules are a bit more complicated than regular expressions. Grammar rules are defined by a binary relation that effectively reduces or builds a sequence of strings. A sentential form in formal grammar is a string that can be derived in a finite number of steps from the starting string. If the string has no "nonterminal" symbols, it is considered a sentence, and the language of a formal grammar is all possible sentences that can be derived from a starting symbol.

Language Generation

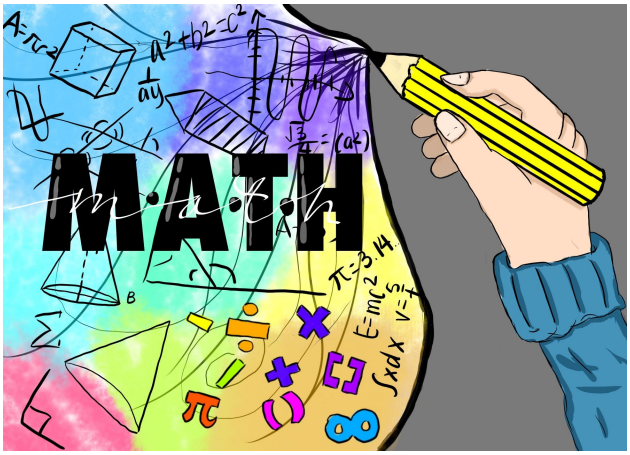


Image by Joey Chen

What about generating language? The most well-known method to do so is the Markov Model, which is used in many fields, but is also applied to language. It can generate text, using the assumption that the word that will appear next only depends on the word before it. A Markov chain is generated by creating something similar to a NFA, with each state representing a word, and each transition to another word depending on the probability that the other word appears after the first one. It is statistically correct, but what it generates might not make sense to us semantically. However, Markov chains are very basic, and machine learning has improved this tremendously.

Syllabication

There are less general, but still difficult, questions when trying to find algorithms to model language. For example, it is surprisingly nontrivial to find an algorithm that knows where to insert hyphens into words that spill into the next line. In 1983, author Franklin Mark Liang published a paper on "word hy-phen-a-tion by com-put-er," where he explains the

algorithm he developed to be used in the TEX82 typesetting system. There was a previous "time magazine" algorithm which required finding every four-letter potential hyphenation point and looking up a table of probabilities for the three possible spaces between the letters for a hyphen. Naturally, that seems very inefficient. The article researches a approach the recognizes patterns of where hyphenations commonly occur (quick-ly), and then patterns of exceptions to the previous patterns (prob-a-bly), and then patterns of exceptions to the previous patterns of exceptions..., and etc. The article also proposed a packed trie data structure to compress the data further [1].

Conclusion

Understanding language has many applications. Consequently, there is definitely motivation to study language in computer science and mathematics. Mathematical linguistics is a relatively recent and interesting field.

1 References

1. "Formal Grammar." *Wikipedia*, Wikimedia Foundation, 21 Jan. 2021, en.wikipedia.org/wiki/Formal_grammar.
2. "Formal Language." *Wikipedia*, Wikimedia Foundation, 31 Jan. 2021, en.wikipedia.org/wiki/Formal_language. Liang, Franklin Mark. *Word Hy-phen-a-tion by Com-put-er*, Department of Computer Science, Stanford University, Aug. 1983, www.tug.org/docs/liang/liang-thesis.pdf.

[1] If you're interested in reading more about the trie data structure used, look for the article titled "Word Hy-phen-a-tion by Com-put-er" by Franklin Mark Liang