# Recognising the English Language using Context Free Grammar with PyFormlang

Harshitha Nagarajan[1], Punitha Vancha[1], Supriya M[1]
[1]Department of Computer Science and Engineering
Amrita School of Engineering, Bengaluru,
Amrita Vishwa Vidyapeetham, India.
harshithan1301@gmail.com, punithareddy14@gmail.com, m_supriya@blr.amrita.edu

2022 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT) | 978-1-6654-9781-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/CONECCT55679.2022.9865855

***Abstract*—Natural language recognition is a sub-field of Natural Language Processing (NLP), a popular research playground that lies in the intersection of multiple areas of linguistics, computer science, artificial intelligence and machine learning. The purpose of NLP is to define a computer that can "understand" the contents of documents, including the language's contextual nuances. The first step, however, would be to pre-process and parse a particular statement to check if it is a legitimate sentence of the language or not. This is where Automata theory comes into the picture. Using the Python *PyFormlang* and *nltk* libraries, we develop an English language recognizer based on Context free grammar (CFG) representations of the English Language, with parts-of-speech (POS) tags making up the constituencies of the CFG. Syntactically accurate sentences are accepted if parsed without errors, else they are deemed invalid. We also layout the set of productions for the English language, which has proven to work well with most sentences including simple and complex, with an accuracy of 84.90%.**

***Keywords—Pyformlang, Context free grammars, parsing, English language recognition***

## I. INTRODUCTION

Natural Language Processing (NLP) is one of the most important Artificial Intelligence (AI) technologies that deals with giving a machine the cognitive ability to process the meaning behind complex human languages. Two of the most basic concepts related to NLP include lexical analysis and syntactic analysis. Lexical analysis refers to breaking down a string of characters into tokens for further processing. The arrangement of words in a sentence is called syntax, and syntactic analysis or "parsing" refers to analysing the logical meaning of the string of characters, given its tokens. This has proven to be extremely useful for applications like grammar checkers, spoken language understanding, question answering systems, automatic text generation, machine translation and so on.

In natural languages, words combine to form phrases, and phrases combine to form sentences. A constituency refers to a group of words that may be considered as a single unit, or a constituent. With this in mind, we can define a parser to be an entity that can process input sentences with respect to certain rules, defined for a grammar. A grammar is a declarative description of a set of well-structured rules that encompasses the properties of a language. Hence, a parser is a procedural interpretation of a grammar.

Language recognition is a widely explored field. Finite automata in NLP prove to be quite useful for tasks requiring language recognition, as they can provide rule-based decisions regarding the acceptance or rejection of strings. The main purpose of parsing is to find syntax errors, be it a natural language or a computer language. The focus of our work is to make use of the Pyformlang library to create and implement a Context Free Grammar (CFG) for the purpose of parsing sentences of the English language. The custom English CFG has been defined to the best of our abilities.

Both natural and programming languages employ CFGs as models of syntax. The concept of constituency is crucial in the development of grammars. Constituency is the abstraction of a group of words acting as a single unit/constituent. Finding constituents in a natural language is a crucial step in developing grammar for that language.

A CFG consists of a start symbol that guides the machine to a starting point for parsing, terminals and non-terminals, and productions. Terminals are the ending points of the parsing process, and non-terminals are variables that can recurrently divide to produce terminals ultimately. Productions are the set of rules that specify the order of derivation or generation of a sentence. Hence to parse a sentence, one would typically begin with string constituting the start symbol and apply one of the productions containing the start symbol, replacing it with terminals or non-terminals on the right-hand side. Upon repeating this process of selecting and replacing by following corresponding productions until all non-terminals have been replaced by terminal symbols validly, the string is parsed without errors.

Context-free languages are inherently powerful, more than regular languages too, in the sense that all regular expressions can be written as a context-free grammar, but not the other way around. Except if one of the context-free grammars is a regular expression or a finite automaton, the intersection of two context-free grammars is not a context-free grammar.

To have an in-depth understanding of formal languages under the hood, we make use of the recently introduced *Pyformlang* library. This is a pedagogical library that has been written exclusively in Python3. With an aim to allow students to use it as a practical learning tool, the library has a clear structure and implements some of the most used algorithms of this domain. It can be used to work with regular expressions, indexed grammars and finite state automata. The context-free grammar module of this library also contains closure property-functionalities such as concatenation, union, reversal etc.

We also lay out all the rules for lexical parsing of the English language that have been used, to ensure reusability of the process followed. Our contributions are as follows:

1. We are one among the first to demonstrate the usage of Pyformlang through a practical application - for English Language recognition, which is helpful for beginners who would like to understand the intricacies of parsing sentences using the concepts of formal languages.
2. We provide a detailed set of possible rules for the English Language.

## II. Literature Survey

The advantages of the pyformlang library over others, and its main functionalities have been detailed in the introductory tool paper [1]. The official documentation for pyformlang and code at [2]. Its great potential extends over various fields of computer science, such as the theory of computation, context-free languages, software engineering, and can be especially helpful to learners looking to build their own applications from scratch to understand core concepts. Most of the popular existing libraries for text parsing such as re, nltk, spacy etc. focus on the parsing of natural languages, rather than the intricate theoretical concepts behind them. In this paper, we focus on the usage of context-free grammars module of this library.

Parts of Speech (POS) tagging is an imperative step in NLP applications. Nltk's POS tagger is a pre-trained, Machine Learning-based greedy averaged-perceptron tagger that has a dictionary of weights associated with features. This is used to predict a suitable tag for a given token. Although it gives users optimal performance, it is not perfect. Advances in Deep Learning have made it possible for the creation of more robust POS-taggers. One such case of a POS-tagger for the Biomedical Domain is discussed in [3], where various deep neural networks (DNN) are compared. Their model achieves an accuracy of 94.80% as DNNs can easily access information on context, providing better predictions. POS tagging for Indian Languages is another field that is still being researched. An innovative Conditional Random Field (CRF) based framework using Scala's EPIC, for Malayalam, is proposed in [4]. For sentences belonging to general domain, this model achieves an accuracy of 87.35%. For Kannada, a practically feasible regular expression POS tagger is proposed in [5]. For code-mixed scenarios, where people tend to mix up regional languages with English while communicating on common social media platforms, existing POS taggers may fail. Hence, the authors of [6] propose a novel method of word embedding for POS tagging by using character-level representations as the features.

POS tags give us constituents of a sentence, and this generally falls under the category of constituency parsing. Similarly, dependency parsing refers to the analysis of sentence grammars based on local word dependencies. References [7,8] highlight use cases of dependency parsing in the real-world. The first paper proposes a method to obtain the dependency of verbs, by extracting information about that verb and performing sentiment analysis for phrases related to that verb, and the latter uses a modified conditional random field to bind dependency parsing with Semantic role labelling. This helps in improving the sematic relationships between predicates and the respective arguments.

Lots of research with respect to English language parsing has been done in the past. A novel genetic algorithm is proposed in [9], that works best for six to eight-word size to get all possible parse trees. In [10], a non-deterministic Push down automaton has been designed for English language by first writing the Context-free grammar in a more general form of Parts of speech tags. Then the CFG is converted to Chomsky normal form to formulate the NPDA conversion. In this implementation, the stack is initially empty and then the starting non-terminal is pushed into this empty stack. The starting symbol is then popped and a production starting from the start symbol is chosen and pushed on to the stack. Until the finish line of the input tape, this process continues. This paper also outlines a useful comparative study of parsing techniques for natural language recognition and has concluded that NPDA is better than other parsing techniques because it has excellent recognizing capabilities, large memory as compared to other parsing techniques and is fast. We have tried to understand their CFG and designed our CFG but with wider range of POS tags obtained using NLTK modules. We do not convert the CFG to Chomsky Normal form because we can directly parse it using the CFG with the help of Pyformlang [11]. Pyformlang has CFG implemented with all necessary operations that can be performed on them.

Reference [12] introduces a parser that is designed by creating a Context Free grammar which is left factored to remove ambiguity. A further understanding of the use of sentence structures for generating a set of production languages was obtained by examining parsers for other languages such as Tamil [13], [14].

## III. A CFG for the English Language

This section describes in detail, the Context free grammar that has been defined for recognising English sentences. CFGs can be used to define the set of rules that can be followed to generate sentences belonging to the English language. Not only are they powerful enough to express sophisticated relations of a natural human language, but are also computationally tractable and efficient.

As mentioned before, a CFG is a 4-tuple set of variables or non-terminal symbols (V) , terminal symbols (T), a set of rules or productions (P) and a start symbol (S). The respective constituencies for a natural language such as English can be determined by considering the parts-of-speech (POS) tags for each word in a sentence, such as nouns, verbs, prepositions and so on. Thus, the CFG for an English Language Recognizer can be as given in eq. (1). All entries of T and V are discussed further in detail.

$$G = (V, T, S, P) \text{ where,} \quad (1)$$

$$T = \{CC, NN, NNS, NNP, NNPS, DT, PRP, PRP\_t, CD, IN, TO, RB, RBR, JJ, JJS, JJR, VB, VBP, VBN, VBG, VBD, VBZ, MD, RP, POS, EX, WDT, WP, WRB\}$$

$$V = \{S, NP, VP, PP, SBAR, ADJP, ADVP, VRB\}$$

### A. Sentence as the Start Symbol (S)

A sentence in English (S) usually has a noun phrase (NP) followed by a verb phrase (VP) or just a verb phrase. For Example, 'Max ran' and 'run'. Sometimes two sentences are seen joined with a conjunction (CC) or a sentence is followed by a preposition (IN) and a noun phrase or by a 'to' (TO) and a verb phrase. A sentence might also contain an

adjectival phrase (ADJP) followed by a noun phrase and verb phrase. For example, 'Max ran and Abigail sang', 'she finished before me' and 'Unfortunately the dog died'. There are sentences that contains the 'wh'-pronouns such as who and 'wh'-adverbs such as where and when. These kinds of sentences are a combination of a sub-sentence followed by a 'wh'-pronoun (WP) and a verb phrase or by a 'wh'-adverb (WRB) and another sub-sentence. For example, 'Not all people who wander are lost'. Table I contains the set of productions for S.

TABLE I. PRODUCTIONS FOR 'S'

| | | |
|---|---|---|
| S | → | NP VP |
| S | → | VP |
| S | → | S WP VP |
| S | → | S WRB S |
| S | → | S CC S |
| S | → | S IN NP |
| S | → | S IN VP |
| S | → | S TO VP |
| S | → | ADJP NP VP |

## B. Noun Phrase (NP)

A noun phrase (NP) can simply be a singular or plural noun (NN, NNS), a singular or plural proper noun (NNP, NNPS), a personal pronoun (PRP) or a possessive pronoun (PRP_t) followed by any other noun (Table II.). For example, son, Max, Americans, I, my desk etc. Other forms of a noun phrase are:

- Nouns preceded by determiners (DT) such as the, these, a, an etc. For example, 'The rain'.
- Nouns preceded by adjectival phrases (ADJP) or by a determiner and adjectival phrase. For example, 'Unfortunately she…', 'the hungry bear…'.
- A possessive pronoun followed by an adjective (JJ) and a noun. For example, 'my cute dog'.
- A cardinal number (CD) followed by a noun and sometimes preceded by a determiner. For example, 'The three musketeers.'
- Two nouns together are sometimes preceded by a determiner. For example, 'light bulb', 'These blue muffins.'
- A possessive pronoun followed by a noun and a proper noun. For example, 'my son Bill'.
- A proper noun with a possessive ending (POS) followed by a noun. For example, 'Asha's desk', 'Indians' craftsmanship'.
- Two noun phrases combined with a conjunction (CC). For example, 'John and Max'.
- A noun phrase followed by a prepositional phrase (PP) or a subordinate phrase (SBAR) or a verb phrase (VP). For example, 'The cow in my backyard', 'The bicycle that I found outside', 'A bicycle made of…'.

## C. Prepositional Phrases (PP)

Prepositional phrases (PP) are noun phrases preceded by a preposition (IN or TO). Two prepositional phrases can be combined together using a conjunction to form one single prepositional phrase. For example, 'At the beach', 'to John' , 'in time and under budget'. Table III illustrates the productions for a prepositional phrase.

## D. Subordinate clause (SBAR)

A Subordinate clause (SBAR) is a preposition followed by another sentence or just another small sentence that is a part of the entire sentence(Table IV). Two subordinate clauses can be joined together by using a conjunction to form a single subordinate clause. For example, 'Maya gave a speech after Max spoke', 'Asha said it was spicy'.

TABLE II. PRODUCTIONS FOR 'NP'

| | | |
|---|---|---|
| NP | → | NN \| NNS \| NNP \| NNPS |
| NP | → | PRP \| WP \| NP EX |
| NP | → | DT NN \| DT NNS |
| NP | → | DT NNP \| DT NNPS |
| NP | → | ADJP NN \| ADJP NNS |
| NP | → | ADJP NNP |
| NP | → | ADJP PRP_t NN |
| NP | → | ADJP PRP_t NNS |
| NP | → | DT ADJP NN \| DT ADJP NNS |
| NP | → | ADJP NN NN \| ADJP NN NNS |
| NP | → | DT ADJP NN NN |
| NP | → | DT ADJP NN NNS |
| NP | → | PRP_t JJ NN \| PRP_t JJ NNS |
| NP | → | CD NNS \| DT CD NNS |
| NP | → | NN NN \| NN NNS |
| NP | → | NNP NN \| NNP NNS |
| NP | → | PRP_t NN NNP \| PRP_t NN |
| NP | → | PRP_t NNP \| PRP_t NNS |
| NP | → | NNP POS NN \| NNP POS NNS |
| NP | → | NNPS POS NN \| NNPS POS NNS |
| NP | → | DT NN NNS \| DT NN NN |
| NP | → | NP PP \| NP VP |
| NP | → | NP CC NP \| NP SBAR |
| NP | → | DT NN NN NN \| DT NN NN NNS |
| NP | → | NP POS ADJP NN \| NP POS ADJP NNS |
| NP | → | NP POS ADJP NNP NN |
| NP | → | NP POS ADJP NNP NNS |

### E. Adverbial phrase (ADVP)

An Adverbial phrase (ADVP) consists of an adverb (RB) followed by a comparative adverb (RBR). An adverb can be followed by a prepositional phrase. Two adverbial phrases can be combined with a conjunction into a single adverbial phrase, refer Table V. For example, 'They went considerably further', 'They went considerably further than before'.

TABLE III. PRODUCTIONS FOR 'PP'

| | | |
|---|---|---|
| **PP** | **→** | **IN NP** |
| **PP** | **→** | **IN WDT NP** |
| **PP** | **→** | **TO NP** |
| **PP** | **→** | **PP CC PP** |

TABLE IV. PRODUCTIONS FOR 'SBAR'

| | | |
|---|---|---|
| **SBAR** | **→** | **IN S** |
| **SBAR** | **→** | **S** |
| **SBAR** | **→** | **SBAR CC SBAR** |

TABLE V. PRODUCTIONS FOR 'ADVP'

| | | |
|---|---|---|
| **ADVP** | **→** | **RB RBR \| RB RB** |
| **ADVP** | **→** | **ADVP PP** |
| **ADVP** | **→** | **ADVP CC ADVP** |

### F. Adjectival Phrases (ADJP)

Adjectival phrases (ADJP) are the phrases that give descriptions of actions or things. They are made of adverbs or adverbs (simple and comparative) followed by simple or comparative adjectives (JJ, JJS). They may contain one or two adjectives and sometimes combined with a conjunction. Adjectival phrases may also contain an adverb followed by a verb of any form. Two Adjectival phrases can be combined into a single phrase using a conjunction, refer Table VI.. For example, 'increasingly unlikely', 'my pretty shiny shoes', 'not talking', 'fast and furious'.

TABLE VI. PRODUCTIONS FOR 'ADJP'

| | | |
|---|---|---|
| **ADJP** | **→** | **RB \| RBS \| VBG** |
| **ADJP** | **→** | **RB JJ \| RBR JJ \| RB JJR** |
| **ADJP** | **→** | **JJ \| JJ JJ \| JJS JJ \| JJ CC JJ** |
| **ADJP** | **→** | **RB VRB** |
| **ADJP** | **→** | **ADJP CC ADJP** |

### G. A verb form (VRB)

The custom defined variable VRB produces verb in any form- base verb (VB), past tense verb (VBD), present participle verb (VBG), past participle verb (VBN), singular present tense verb (VBP), third person singular present tense verb (VBZ). Table VII contains productions for VRB.

TABLE VII. PRODUCTIONS FOR 'VRB'

| | | |
|---|---|---|
| **VRB** | **→** | **VB \| VBZ \| VBD** |
| **VRB** | **→** | **VBN \| VBG \| VBP** |

### H. Verb Phase (VP)

A simple Verb Phrase (VP) is usually a verb in any form. The verb may be followed by a prepositional phrase or adverbial phrase or adjectival phrase or a particle (RP). Two verb phrases may be combined using a conjunction, refer table VIII. For example, 'talking', 'studied at night', 'go away' etc. Other forms of verb phrase are:

- Verb followed by a particle and noun phrase. For example, 'gave up their ill-gotten gains'.
- Verb followed by an adverb. For example, 'sat silently'.
- Modal verb (MD) followed by a verb. For example, 'She will snack'.
- 'To' followed by a verb. For example, 'she wants to snack'.
- Adverbial phrase followed by a verb. For example, 'she is not writing'.
- Verb phrase followed by a noun phrase. For example, 'ran every morning'.
- Verb followed by two noun phrases. For example, 'she taught her brother algebra'.
- Existential there (EX) or a Wh-determiner (WDT) such as which followed by a verb phrase. For example, 'there is a reason...', 'which was...'.

## IV. IMPLEMENTATION

With the grammar defined, implementation can now be completed by clubbing the advantages of the *pyformlang* and *nltk* libraries. The Pyformlang construction is close to text-book representation of Context Free Grammar, so it is easier to design and implement it. We can directly check if a word belongs to the given CFG without having to convert to Chomsky Normal form.

### A. Required Libraries

*1) Pyformlang:* One way to define a context free grammar involves representing variables, terminals, and production rules with internal 'representation' objects. This initialization procedure can be lengthy, and it is not always required. It is, nonetheless, close to textbook representations and aids in the comprehension of context-free grammars. Hence, we proceed with this method of grammar definition, with a focus on the Context Free Grammar section of pyformlang. Necessary imports include:

*a) Production:* To define the set of rules for CFG

*b) Variable:* To define non-terminals

*c) Terminal:* To define terminals

*d) CFG:* To create the grammar with the above constituencies

TABLE VIII. PRODUCTIONS FOR 'VP'

| | | |
|---|---|---|
| **VP** | → | **VRB** |
| **VP** | → | **VRB PP** |
| **VP** | → | **VRB ADVP** |
| **VP** | → | **VP ADJP** |
| **VP** | → | **VRB RP** |
| **VP** | → | **VP CC VP** |
| **VP** | → | **VRB RP NP** |
| **VP** | → | **VRB RB** |
| **VP** | → | **MD VRB** |
| **VP** | → | **MD RB VRB** |
| **VP** | → | **TO VRB** |
| **VP** | → | **ADVP VRB** |
| **VP** | → | **VP NP** |
| **VP** | → | **RB VP** |
| **VP** | → | **VRB RB NP** |
| **VP** | → | **VRB NP NP** |
| **VP** | → | **EX VP** |
| **VP** | → | **WDT VP** |

*2) Nltk:* 'Nltk' stands for Natural Language Toolkit. It is a Python-based set of modules and programmes for symbolic and statistical natural language processing in English. For our purpose, we make use of only two functionalities:

*a) word_tokenize:* To tokenize the input sentence

*b) pos_tag:* To get the parts of speech(POS) tags for each token as a dictionary

### B. Process of Parsing

Syntactic parsing of a sentence refers to analyzing it with respect to its grammatical constituents, by identifying the parts of speech and the syntactic relations between them. If the relations follow the grammar of the language, then it is deemed as valid. Such a sentence is considered to belong to the language in consideration. However, it is to be noted that the semantics of a sentence are not captured in this phase.

Context Free Grammar for the English language is defined as the first step, refer block diagram Fig. 1. The POS tags serve as terminals, which are instantiated using the Terminal module from *pyformlang.cfg*. Similarly, the Variable and Productions modules are used to define the rest of the grammar components that have been elaborated on in section III. Finally, the grammar is created for use with the help of the *CFG()* function, by passing in the variables, terminals, start symbol and productions as arguments. This completes the definition of the grammar.

In order to parse a sentence of the English language, every input sentence is first tokenized. Parts of Speech (POS) tags for each token are then obtained using the *nltk* library. The tags are then passed onto *Pyformlang's* CFG module, to check for membership to the defined English grammar. If the tags follow the sequence of a valid production, then the sentence is considered valid. In the opposite case, the sentence is deemed invalid.

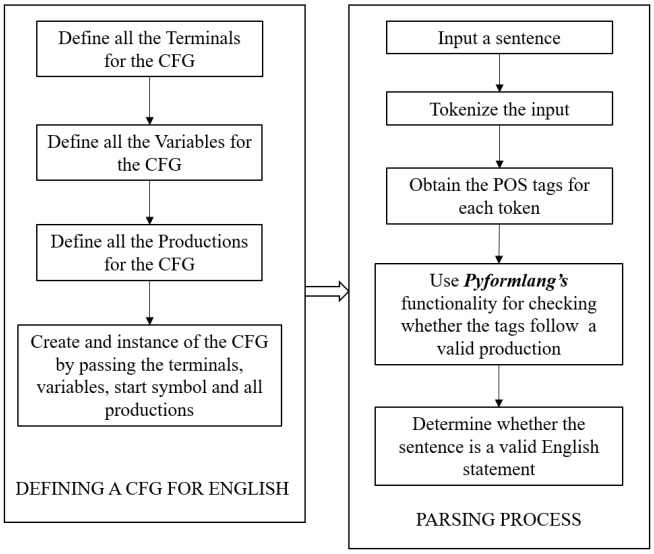


Fig. 1. Block diagram of the process flow for recognizing an English sentence

## V. RESULTS AND ANALYSIS

The recognizer was tested on a custom labelled dataset that contains two columns named 'Sentences' and 'DesiredOutput'. As the name of the second column suggests, it contains information on whether the respective sentences are acceptable (labelled 1) or not acceptable (labelled 0). The dataset contains 717 such acceptable sentences and 800 unacceptable sentences.

To evaluate the performance of the recognizer, we obtain and plot a confusion matrix (Fig. 2) after parsing each sentence of the dataset. We also obtain other metrics such as accuracy score, precision score, recall score and f1_score. These metrics are aimed at testing the accuracy of the CFG as an English language recognizer.

Out of 800 unacceptable sentences 728 sentences were predicted as unacceptable and only 72 as acceptable. Out of 717 acceptable sentences, sentences were predicted 157 as unacceptable and 560 were predicted as acceptable. The accuracy score is 84.90%, meaning that out of the total predictions, 84.90% were predicted correctly. The precision value is 88.61% which means 88.61% of total positive

predictions are correctly predicted. Recall score tells us how many of the observations of positive class are actually predicted as positive and, in our case, it is 78.10%. The f1-score, or the harmonic mean of recall and precision is observed to be 88.02% but this metric is not significant because our dataset is well balanced.

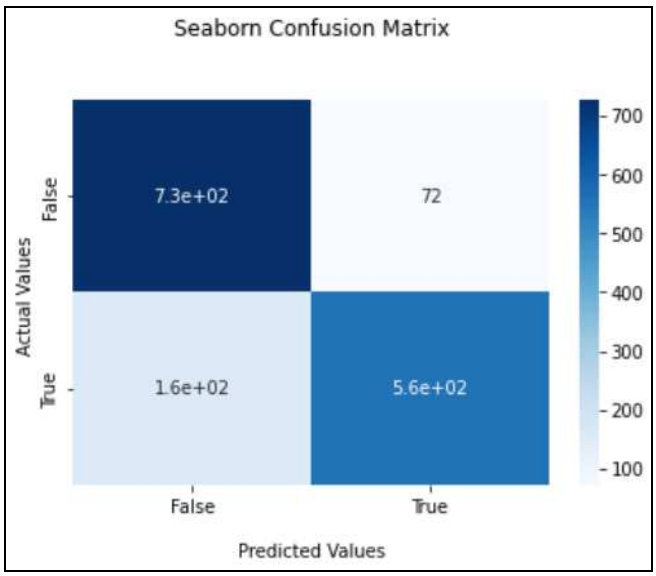


Fig. 2. Confusion Matrix obtained upon evaluating the English Language Recognizer on the custom dataset

## VI. Conclusion and Future Study

We have created an English Language Recognizer (ELR) using the Pyformlang and nltk modules, which tells us whether a set of tags corresponding to a given sentence is accepted in the English Language or not. The parser works by the set of rules given through the designed CFG.

The system can be improved in the future by addressing current limitations. Using a more accurate POS tagger can improve the quality of the system. For example, the POS tagger used currently tags words like talk, walk, studies etc as nouns even in a context where they are being used as a verb. Also, sentences where determiners such as 'this', 'that', 'these' etc. are generally used as noun phrases, give rise to false negatives. For example, "this is a sentence" is a valid sentence according to general English but since the POS tagger tags them as DT i.e., determiner and that doesn't qualify as a noun phrase, they do not pass unless they are followed by another noun. Hence, the same example would be accepted if it was "this string is a sentence". It must be noted that an ELR does not consider the grammar of the English language so sentences with bad grammar will be passed too if they are in an acceptable structure.

The ELR that has been designed is not robust to punctuations in a sentence. Hence, sentences with punctuation are rejected. This can be addressed by removing punctuations before passing to the CFG or by designing a more complex CFG that understands the usage of punctuations in English language.

As observed in the results section, the current system has an accuracy score of 84.9% and a precision of 88.61%. It is better than [11] where the accuracy score was 78.48%. The accuracy can be increased further with a greater number of precise rules in CFG. As part of future study, we also wish to expand the production rules to other forms of sentences such as questions and those containing punctuations.

## References


[1] Romero, Julien. (2021). Pyformlang: An Educational Library for Formal Language Manipulation. 10.1145/3408877.3432464.

[2] https://github.com/Aunsiels/pyformlang

[3] A. Gopalakrishnan, K. P. Soman and B. Premjith, "Part-of-Speech Tagger for Biomedical Domain Using Deep Neural Network Architecture," 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT), 2019, pp. 1-5, doi: 10.1109/ICCCNT45670.2019.8944559.

[4] Kumar S, Sachin & Kumar, M. & Kp, Soman. (2016). Experimental analysis of malayalam pos tagger using epic framework in scala. 11. 8017-8023.

[5] K. M. Shiva Kumar and Dr. Deepa Gupta, "Regular expression tagger for Kannada parts of speech tagging", 2nd International Conference on Computational Intelligence and Informatics, ICCII 2017, Advances in Intelligent Systems and Computing, vol. 712. Springer Verlag, Hyderabad, India, pp. 121-130, 2018.

[6] Kumar, M. & Padannayil, Soman. (2021). Transfer learning based code-mixed part-of-speech tagging using character level representations for Indian languages. Journal of Ambient Intelligence and Humanized Computing. 1-12. 10.1007/s12652-021-03573-3.

[7] G. Veena, A. Vinayak and A. J. Nair, "Sentiment Analysis using Improved Vader and Dependency Parsing," 2021 2nd Global Conference for Advancement in Technology (GCAT), 2021, pp. 1-6, doi: 10.1109/GCAT52182.2021.9587829.

[8] Gangadharan, Veena. (2018). An Extended Model for Semantic Role Labeling Using Word Sense Disambiguation and Dependency Parsing. Journal of Engineering and Applied Sciences. 12. 10.3923/jeasci.2017.7508.7513.

[9] Menon, Vijay & Kp, Soman. (2018). A New Evolutionary Parsing Algorithm for LTAG. 10.1007/978-981-10-3373-5_45.

[10] Pasha, Madiha. (2014). To Design a English Language Recognizer by using Nondeterministic Pushdown Automata (ELR-NPDA) General Terms. International Journal of Computer Applications. 105. 975-8887. 10.5120/18345-9467.

[11] https://pyformlang.readthedocs.io/en/latest/_modules/pyformlang/cfg/cfg.html#CFG.contains

[12] Magdum P. G., Kodavade D. V. / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com ,Vol. 3, Issue 4, Jul-Aug 2013, pp.306-312

[13] Liyanage, Chamila & Ariaratnam, Ifancy & Weerasinghe, Ruvan. (2014). A Shallow Parser for Tamil. 10.1109/ICTER.2014.7083901.

[14] K. Saravanan, P. Ranjani, and T. V. Geetha, ―Syntactic Parser for Tamil, ‖ in Proceedings of Sixth Tamil Internet 2003 Conference, Chennai, India. 2003. pp. 28-37.