Different Approaches in Serbian Language Parsing using Context-free Grammars

Teodora Đorđević and Suzana Stojković, *University of Niš, Faculty of Electronic Engineering*

***Abstract*—Syntax analysis is an extremely significant phase of natural language processing. This paper presents a comparison of two methods for syntax analysis of the Serbian languages based on context-free grammars. First, it describes building a POS tagger with corpora. Secondly, it defines a context-free grammar for the Serbian language. After that, it explains how the syntax parser is created and compares its performance with a parser implemented using NLTK. Finally, it explains the post-processing layer which is added in order to reduce the number of syntax trees generated due to grammar ambiguity. The experiments showed that the implemented parser is on average 6115 times faster than the NLTK parser and that the post-processing reduced the number of syntax trees by 54% on average.**

***Index Terms*— NLP; POS tagging; Context-free grammars; Syntax analysis; CYK algorithm**

I. Introduction

Natural language processing has been an essential theme in computer science and engineering since its very inception as evidenced by the postulation of the Turing test which was first described by the renowned computer scientist and visionary Alan Turing in 1950. Serving as the *lingua franca* of the modern age the English language has been the most heavily researched language in the NLP context. That being said, many of the techniques developed for processing English cannot be directly applied to many different languages because of certain fundamental differences in language structure. Each language has its own set of peculiarities that need to be taken into consideration when developing computer programs capable of comprehending them.

Syntax analysis or parsing is the process of analyzing a sentence in natural languages conforming to the rules of a grammar. Creating a successful syntax analyzer grants a good foundation for building more sophisticated semantic analyzers. Syntax analysis has its own role in Information Extraction, Question Answering systems, Rule-based Machine Translation, etc.

This paper presents a comparison between two parsers for the Serbian language, one developed by using NLTK [1] and the other implemented using the CYK algorithm [2]. Before implementing a syntax parser, it is necessary to train a POS tagger and then define a Context-free grammar for the Serbian language. The next step after completing the parsers is adding a post-processing layer in order to minimize the number of syntax trees and to remove the trees that are completely inconsistent with the language.

Teodora Đorđević is with the Faculty of Electronic Engineering, University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia (e-mail: teodora.djordjevic@elfak.ni.ac.rs).
Suzana Stojković is with the Faculty of Electronic Engineering, University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia (e-mail: suzana.stojkovic@elfak.ni.ac.rs).

II. Related Work

Syntax analysis is a process of generating syntax trees for an input sentence. By applying syntax analysis, a sentence is given specific structure. There are certain approaches in syntax analysis or parsing. First, there is parsing with context-free grammars. A context-free grammar [3] represents a list of rules which are used to generate a syntax tree for a given input. This approach has various problems due to inability to predict every possible sentence structure. Also, adding more rules can generate more syntax trees and not lead to parsing improvements. The second group of algorithms is based on statistics. The first approach can generate a large amount of syntax trees, so adding statistics [4] is the best solution for this problem, because it helps determine which syntax tree is most likely to be correct. Before explaining algorithms for syntax analysis, it is necessary to present search methods that can be used. There are two strategies, top-down and bottom-up.

The top-down strategy is also called goal-directed search. This technique does not consider input itself, as it only considers whether a syntax tree can be generated from a given grammar. The top-down method starts with the starting symbol S and moves to the bottom. It is necessary to find all the rules that have the starting symbol on the left side and generate syntax trees with the starting symbol as the syntax tree root. After that, those constituents are used to expand the tree even more until the leaves are reached. At every level, the algorithm considers rules with the current symbol on the left side in order to expand a syntax tree. When the syntax tree with leaves is generated, it is necessary to rule out all the rules where the input sentence does not match the created syntax tree. Syntax trees that are not ruled out in this phase are the result of syntax analysis.

The bottom-up parsing represents a different approach. The parser starts from the input sentence, and tries to build syntax trees from input words, by going up. The parser attempts to move up the branch for each potential syntax tree by attempting to match the right side of the rule with the existing nodes. The rule that is a match is reduced to its left side until the starting symbol is reached. The parsing is successful if a syntax tree for a given input sentence exists and contains the starting symbol S as its root.

Each of these techniques has both its own advantages and disadvantages. The top-down method does not attempt to

create syntax trees that do not have the starting symbol as its root, but also generates trees that are not a match to a given input. The bottom-up method does not generate any syntax trees that do not match the sentence being parsed, but it does explore trees that do not begin with the starting symbol and are thus invalid.

By using either one of the aforementioned methods parsing can be implemented effectively. The first algorithms designed for syntax analysis of natural languages used context-free grammars as a set of rules for parsing. This approach requires creating a context-free grammar and then choosing an appropriate parsing algorithm to generate syntax trees. When choosing a parsing algorithm, it is necessary to consider the search method. The algorithms for parsing using a context-free grammar-based approach are CKY, the Earley algorithm [5] and Chart parsing [6]. This paper will later explain an implementation of the CYK parser.

Parsing with context-free grammars can result in generating multiple trees for a single sentence. This ambiguity problem in most cases cannot be solved by adding more grammar rules. It can be solved by adding statistics which leads to a different approach – statistical parsing. Statistical parsing calculates probabilities for every possible syntax tree and chooses a syntax tree with the largest probability. A possible way of assigning these probabilities to the syntax trees is by using a Probabilistic Context-free grammar (PCFG) [7]. In PCFG, the CFG rules are extended with an associated probability which determines how likely it is that that specific rule will be applied.

III. POS TAGGING

Tagging is an essential part of natural language processing. Structurally, it is executed before parsing takes place with the result of the tagging process being forwarded as an input to a syntax analyzer. Tagging is applied on a per-word level; the result of tagging is a tag which contains information regarding the word's Part of Speech. The Serbian language distinguishes between ten different parts of speech: nouns, verbs, numbers, adjectives, pronouns, conjunctions, prepositions, particles, adverbs, and interjections. POS tagging is implemented using classification. Classification is a supervised machine learning method which needs a set of data for training. In natural language processing this set of data is called a corpus of words. Those words are pre-tagged and used for training. Classification does not consist of only training, but it also requires an evaluation using the test set.

Syntax analysis for the Serbian language cannot be performed successfully by using only part of speech. Seeing as how many words of the Serbian language take many forms depending on their role in the sentence, a syntax analyzer needs more information in order to assign a syntax structure to a sentence. Some examples of the extra information needed are case for nouns, adjectives, pronouns, and numbers, whether a verb is a main or auxiliary verb, the type of pronoun, etc. There are not many corpora for the Serbian language. The one used for classification is srWaC [8]. srWac is a web corpus formed from .rs top-level domain. The first version contains 894 million of annotated tokens, and later versions around 600 million.

This corpus was chosen because it contains deep tags with lots of information. That being said, the corpus also contained too much information for some of the POS, so those values were filtered out in order to improve and simplify classification. After filtering out certain fields, an additional step of processing is applied in order to improve the usability of the corpus. The tags for pronouns contained information on pronoun type, whether it is personal, possessive, interrogative, indefinite, etc. This pronoun type is not as useful for syntax analysis as whether the pronoun is noun related or adjective related. This division impacts whether the pronoun can stand on its own or depends on some other word. Because of pronouns, srWaC corpus needed to be altered to only differentiate pronouns as noun or adjective related. These alterations are based on current type and some common word parts. For example, personal pronouns can only be noun related, demonstrative pronouns can only be adjective related, noun related pronouns often contain 'ko' or 'šta' inside the pronoun itself. The current pronoun tags were removed, and these rules are used to generate new tags for pronouns. After updating this corpus, it was necessary to train the classification model.

Classification is implemented using the nltk tool, more specifically nltk's Naive Bayes Classifier. First, the training was implemented without reducing some of the unnecessary information in tags, and this classification resulted in 89% accuracy. After tag reducing, accuracy rose to 90%. The final change in the corpus with pronoun tag simplification resulted in 91% accuracy.

IV. SYNTAX ANALYSIS

The nltk implementation of syntax analyzer uses a Context-free grammar of the target language. The definition of the grammar includes [3]:

- A starting symbol of the grammar,
- Terminal symbols – strings which contain tags that tagger recognizes,
- Grammar rules that have only one non-terminal symbol on the left side.

Nltk expects that the file which contains grammar rules also contains all the words that can be recognized in the target language as terminal symbols. The problem with this approach is that putting all words of one language in single file is nearly impossible, so the terminals of this grammar are not words, but rather tags that the POS tagger returned.

The CYK parser, on the other hand, uses a Context-free grammar in Chomsky Normal Form [9]. This means that every non-terminal can only be reduced as one terminal or two non-terminals. For rules

$$A \rightarrow B, \qquad (1)$$

where A and B are non-terminals the parsing algorithm needs to be extended.

A. Grammar

The designed grammar is intended for both the nltk and CYK parser, so it is chosen to be in Chomsky normal form. The grammar covers all the syntax units that can be detected in one sentence of the Serbian language. The first part of the grammar covers the sentence structure. One sentence consists of sentence members, and sentence members can be subject, object, predicate, adverbial provision, apposition, etc. [10] Part of grammar that describes the structure of the sentence is shown in Fig. 1.

```
S -> RecenicniClanovi
RecenicniClanovi -> RecenicniClan RecenicniClanovi
RecenicniClanovi -> RecenicniClan
RecenicniClan -> Subjekat
RecenicniClan -> Predikat
RecenicniClan -> Objekat
RecenicniClan -> PriloskaOdredba
RecenicniClan -> PomocneReci
RecenicniClan -> Apozicija
RecenicniClan -> Apozitiv
RecenicniClan -> DeoPredikata
```

Fig. 1. Part of the grammar rules

For each of the sentence members a new set of rules is defined to describe them:

- Subject – there are two different types of subjects in the Serbian language. One is logical and the other one is grammatical. The logical subject is most likely to be in genitive, dative or accusative. The grammatical subject is a performer of the action expressed by predicate.
- Predicate – can be a noun or verb related. A verb predicate can consist of a main verb, an auxiliary verb, affirmative and negative particles, a verb in passive, etc. The set of rules for a verb predicate considers a list that consists of some of those parts. The noun predicate consists of a verbal part and a noun part. A verbal part is an auxiliary verb and a noun part can be a noun, adjective, or adverb.
- Object – can be direct or indirect. An object is an addition to the predicate. The main difference between direct and indirect objects is the case and the preposition absence.
- Adverbial provision – has similar definition as the indirect object and also serves as an addition to the predicate. The difference between the indirect object and the adverbial preposition is meaning in the sentence. The indirect object is a supplement to predicate, while the adverbial provision is more focused on giving information related to the action that predicate designates.
- Apposition and appositive - represent a syntax structure that is surrounded by commas and gives an alternative meaning to a noun or adjective it follows, respectively.
- Helper words – the words that have no syntactic meaning but are often part of the sentence and they must be covered by the grammar.

The total number of rules in the grammar is 406 so not all of the rules can be displayed. An example of a syntax tree built by defined grammar is shown in Fig 2.

While testing the NLTK parser with different texts, it has been noticed that some of the sentences were not recognized,

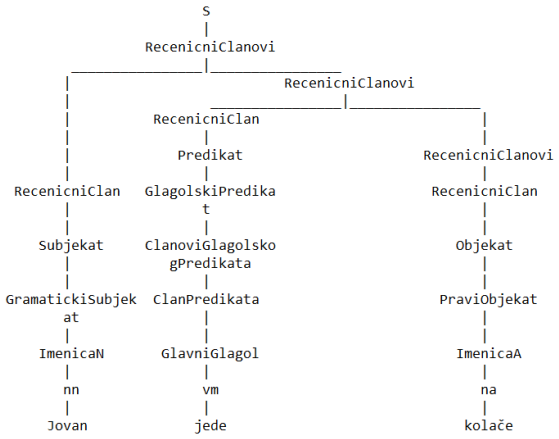

Fig. 2. Example of a syntax tree

which resulted in adding more rules in order to cover more sentence structures. A new problem that arose is the slowness of the NLTK parser, which led to a parsing duration of half an hour for a single sentence. Adding or altering the grammar rules became nearly impossible because the testing for 20-30 sentences lasted almost all day. The new parser using CYK algorithm was created as the solution for this problem.

B. CYK Parser

The CYK algorithm is a parsing algorithm for context free grammars, which uses a bottom-up search strategy. It uses a dynamic programming algorithm to tell whether a string is in the language of a grammar. There is a difference between the recognition and parsing algorithm. The recognition algorithm only shows whether a sentence is recognized by the grammar or not. The parsing algorithm returns all the possible syntax trees for a given input. The pseudocode that is used for CYK parser implementation is shown in Fig. 3.

```
function CYK-PARSE(words, grammar) returns table

for j ← from 1 to LENGTH(words) do
        table[j - 1, j] ← { A | A → words[j] ∈ grammar }
        for i ← from j - 2 downto 0 do
                for k ← i + 1 to j - 1 do
                        table[i,j] ← table[i,j] ∪
                                { A | A → BC ∈ grammar,
                                  B ∈ table[i,k]
                                  C ∈ table[k,j] }
                table[i,j] ← table[i,j] ∪
                                { D | D → A ∈ grammar }
```

Fig. 3. CYK parse pseudocode [4]

The CYK parser was implemented using Python. The parser implementation requires generating a table of dimensions $(n+1) * (n+1)$ where n is the length of an input sentence. The idea is to fill out the table's diagonal with every possible production that has the current input as a terminal symbol in the grammar. After that, the parsing phase implements matching certain cells according to algorithm in order to climb up the tree until reaching starting symbol. Every cell is designed to memorize a list of nodes. The

original pseudocode for CYK algorithm is expanded to fit the unary rules. Every time a new node is added to a list of nodes in a cell with indices [*i*][*j*], it is necessary to check whether that node is on the right side of some unary rule. If so, then the node to the left side of that unary rule is added to the node list. After parsing is completed, all possible syntax trees are found in cells with indices [*0*][*n*]. Generating the syntax trees consists of finding all the nodes that start with the starting symbol S and searching all the syntax trees by references.

V. Post-processing

The grammar is designed to recognize as many sentences as possible, but that also led to generating a lot of syntax trees for a single sentence. The NLTK parser and the CYK parser always generated the same trees, but that number for the longer sentences was often very large. In order to solve ambiguity, a new layer is added after parsing and generating the syntax trees and that layer is called the post-processing. The post-processing was implemented by eliminating trees that cannot be possible in the Serbian language. The rules that generate these kinds of trees couldn't be left out from the grammar, so the solution was to search all the trees and to rule out the impossible ones. Some examples of post-processing rules are eliminating sentences that had multiple subjects and multiple predicates (every simple sentence has only one predicate). Also, the post-processing considered checking whether recognized apposition really has a noun in the same case on the left or the right side and whether recognized appositive has an adjective in the same case on the left or the right side, etc.

VI. Experimental results

The results of the implemented parser are shown with a comparison of how much time was needed for both parsers to generate the syntax trees for the same sentence. Besides the performance of the parser, it is tested how post-processing affected the reducing of the syntax trees.

The results for twenty sentences are shown in Table 1. The results show that parser implemented using CYK algorithm is thousands of times faster than the parser already implemented in the NLTK tool. Also, in most of the cases the post processing eliminated more than 50% of the trees and improved parsing results.

VII. Conclusion

In this paper, it is explained how the corpus for the Serbian language is updated and how the POS tagger for the Serbian language that uses updated corpus is created. It is described how the context-free grammar in Chomsky Normal Form for the Serbian language is created. Finally, it is presented creating of the syntax parser in two different ways using the previously mentioned context-free grammar. Based on the test results the CYK parser was 6115 times faster than the NLTK on average. Also, the post-processing layer after parsing reduced the number of syntax trees by 54% on average.

From all of the above, it can be concluded that although creating a new syntax analyzer based on the CYK algorithm and applying post-processing improved parsing and generated mostly three or less syntax trees there is still room for improvement. Improvement can be achieved by eliminating ambiguity entirely and creating a syntax parser that will only generate one syntax tree per sentence. One way to do that is by using Probabilistic Context-free grammars.

TABLE I

PARSER COMPARISON

Len	CYK trees	PP	PP imp. (%)	CYK time (s)	NLTK time (s)	NLTK / CYK (time)
6	2	1	50%	0.07	176	2514
10	8	2	75%	0.29	957	3300
7	2	1	50%	0.10	305	3050
7	26	9	65%	0.24	2301	9587
3	4	1	75%	0.01	224	22400
5	4	2	50%	0.07	350	5000
12	16	2	87%	0.76	1841	2422
10	52	9	83%	0.72	4975	6909
7	2	2	0%	0.12	179	1492
10	30	3	90%	0.48	2910	6062
6	6	6	0%	0.08	787	9837
4	1	1	0%	0.02	123	6150
5	2	1	50%	0.05	203	4060
8	10	2	80%	0.20	944	4720
5	10	2	80%	0.09	583	6477
5	4	2	50%	0.06	464	7733
6	2	1	50%	0.08	450	5625
8	18	6	66%	0.30	1813	6043
6	2	2	0%	0.08	212	2650
7	20	3	85%	0.21	1318	6276
Average:			54.3%			6115.3

Acknowledgment

This work has been supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia.

References

[1] S. Bird, E. Klein, E. Loper, *Natural Language Processing with Python*, Sebastopol, USA, O'Reilly Media, 2009.

[2] D.H. Younger, "Recognition and parsing of context-free languages in n^3", *Information and Control*, vol. 10, no. 2, pp. 189-208, 1967.

[3] M. Stanković, S. Stojković, Ž. Tošić, *Programski prevodioci*, Niš, Serbia, Elektronski fakultet, 2018.

[4] Jurafsky, J. Martin, *Speech and Language Processing, 2nd edition*, New York, USA, Prentice-Hall, 2009.

[5] J. Earley, "An Efficient Context-Free Parsing Algorithm", Ph.D. dissertation, Carnegie-Mellon University, Pittsburgh, USA, 1968.

[6] R. M. Kaplan, *Natural Language Processing*, New York, Algorithmics Press, 1973.

[7] T. L. Booth, "Probabilistic representation of formal languages", 10th Annual Symposium on SWAT, pp. 74-81, Waterloo, Canada, 1969.

[8] srWaC – Serbian web corpus, http://nlp.ffzg.hr/resources/corpora/srwac/, 12.07.2020.

[9] N. Chomsky, "On certain formal properties of grammar", *Information and Control*, vol. 2, no 2, pp. 137-167, 1959.

[10] Ž. Stanojčić, Lj. Popović, *Gramatika srpskog jezika za gimnazije i srednje škole*, Belgrade, Serbia, Zavod za udžbenike, 2011