A Tool for Sentence Syntax Structure Markup for The Serbian Language

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

***Abstract*— Syntax analysis is an important part of natural language processing. The biggest challenge to defining a natural language syntax analyzer is the inability to define unambiguous formal grammars that describe the language. Because of this, rule-based syntax analyzers need to be enhanced using statistics to allow us to predict which syntax tree is most likely. In order to do this, a corpus of tagged sentences in the target language is needed. The creation of this corpus is long and tedious work. Because of this, this paper implements a visual tool for creating such a corpus for the Serbian language. A component of this tool is the syntax analyzer, which generates all the possible syntax trees based on the defined grammar such that an expert may choose one of them. The expert may also create entirely new syntax trees.**

***Index Terms*—Natural language processing (NLP); Syntax analysis; CYK; Annotated syntactic corpora; Serbian language**

I. Introduction

Natural language processing is a branch of computer science that teaches computers to understand and manipulate human language. Natural language processing is a combination of computer science, linguistics and machine learning. Many NLP techniques are already developed and applied for the English language but applying those techniques to different languages can be quite a challenge. Serbian language is under-researched in the context of natural language processing. Since the Serbian language and the English language do not belong to the same language group, many approaches designed for the English language need to be significantly modified in order to be used in the Serbian language or cannot be used at all.

Syntax analysis or parsing, in general, is the process of analyzing character strings according to the rules of a given formal grammar. It is typically encountered in fields of natural languages, computer languages or data structures. In Natural language processing, syntax analysis is one of the most important phases because it builds a great foundation to natural language understanding. Syntax analysis decides whether a sentence written in natural language conforms to the rules of a formal grammar and thus whether a sentence is valid or not. Designing a quality syntax parser is extremely significant for designing a semantical analyzer, since syntax parsing precedes semantic analysis. Also, syntax analysis has its own role in Rule-based Machine Translation, Information Extraction, Question Answering systems, etc.

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).

This paper describes designing a graphic tool for syntax analysis of the Serbian language based on the syntax analyzer designed in [1]. This tool is implemented as a web application that can analyze and visualize sentences using the implemented parser. It also enables the user to draw completely new syntax trees and save them.

II. Related Work

In linguistics, a corpus usually represents a collection of texts. The main purpose of a corpus is to be used as a tool in language study. In order to study and analyze language data, having a corpus is essential.

The English language is most widely researched language and thus the largest number of corpora exists for the English language.

Corpora contain texts that are sourced from natural contexts in order to be as close to the natural language as possible. There are many types of corpora that are used in natural language processing such as reference corpora, which are fairly balanced sets of texts that accurately describe a standard language, or specialized corpora which contain texts from a particular area, such as movie reviews, magazine texts, etc. A very important category of corpora are annotated corpora which contain additional information such as part of speech tags, lemmas, metadata, additional tags, etc. Annotated corpora can thus be used in supervised learning scenarios when attempting to infer this additional data based on the text given.

There are many publicly available corpora online for various languages. These corpora can be accessed either directly online via web browser, through specialized APIs to search the corpora, or can also be downloaded in their entirety.

Most modern language processing is done using computers. This means that modern corpora must be electronically readable documents. The first such document for the English language was The Brown Corpus of Standard American English [2]. This corpus consists of one million words of American English texts printed in 1961. In order to ensure high quality and to make the corpus useful for a wide range of applications, the corpus compiled texts from 15 different categories. Keeping in mind the huge increase in processing power, as well as that the Internet generates more linguistical data than ever before, this corpus is now considered small.

An example of a modern corpus of the English language that is quite big is the Corpus of Contemporary American English (COCA) [3]. COCA is probably the most widely used corpus of English, with over one billion words. Many corpora for the English language can be found at [4].

Besides the English language many other languages of the world are being researched in the field of syntax and semantic

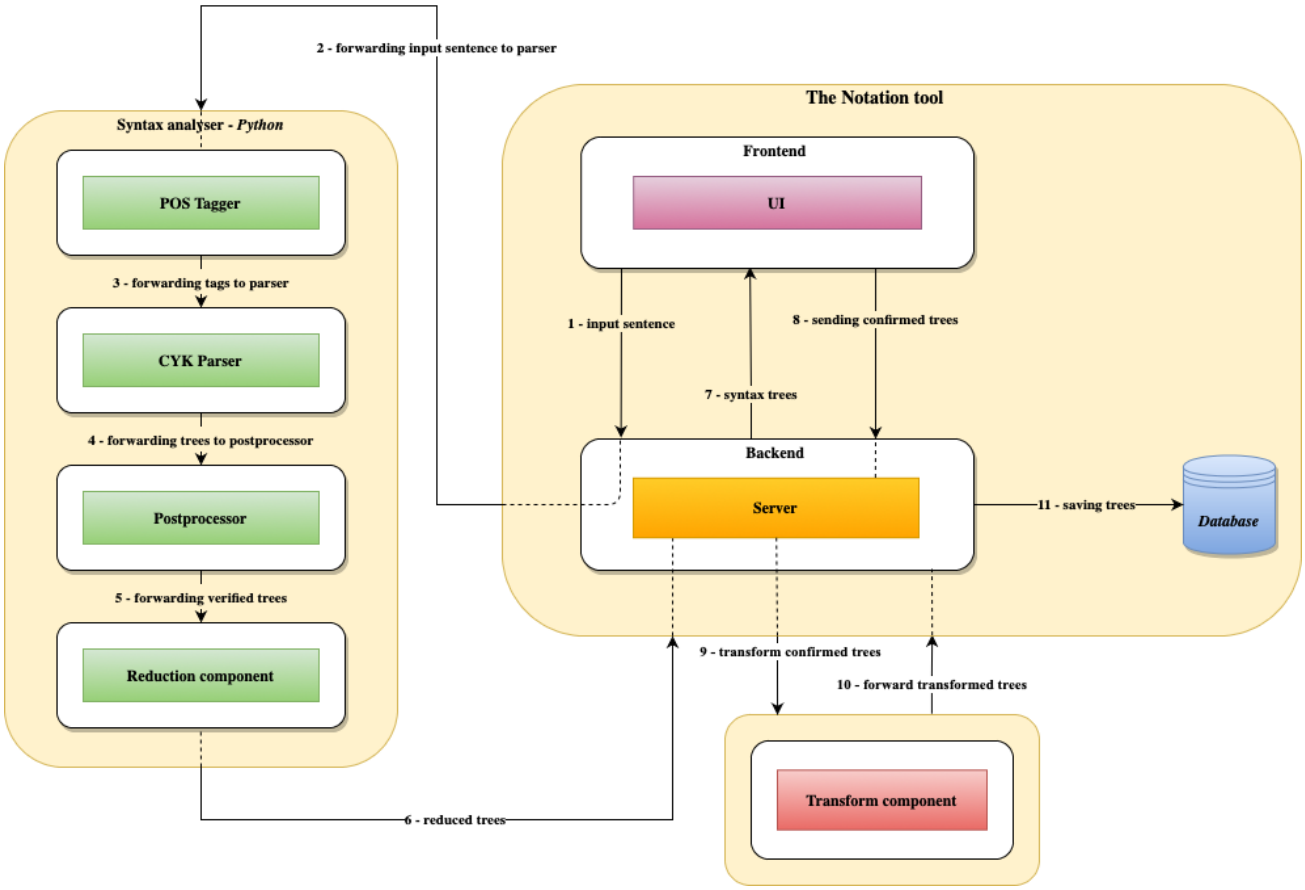

Fig. 1. The Architecture of Notation tool

analysis and are thus being compiled into language specific corpora. One such corpus is the Quranic Arabic Corpus [5], which is an annotated linguistic document, that shows the Arabic grammar, syntax and morphology for each word in the Quran. The research paper [6] describes a specialized annotated corpus for the Chinese language, used for analyzing clinical texts. This corpus is annotated with part-of-speech tags, syntactic tags, entities, assertions, and relations.

For the Serbian language, given that it is spoken by only 12 million people in the world as a first language, there is not a large number of corpora such as English or Chinese. In the last few years, there has been development of open, freely available resources and technologies for computer processing of texts in the Serbian language. This includes annotated language corpora and some of the corpora are listed below.

1. *SETimes.SR* [7] – it is based on the *SETimes* parallel corpus of newspaper articles. This is a manually annotated corpus of texts written in the standard Serbian language. This corpus is used for training and evaluation of computer models on a number of natural language processing problems. It contains 3891 sentences. The *SETimes.SR* corpus is annotated using morphosyntactic notation, lemmas, syntactic dependencies, and named entities.
2. *srWac* [8] - the Serbian web corpus, which was built by crawling the *.rs* top-level domain in 2014. It contains 555 million tokens and over 25 million sentences arranged in about 1.3 million documents.
3. *MULTEXT-East* [9] - is a multilingual dataset for language research. This project consists of mainly Central and Eastern European languages, including Serbian.
4. *ReLDI-NormTagNER-sr 2.1* [10] - is a manually annotated corpus of Serbian tweets. It is meant for use in the fields of tokenization, sentence segmentation, word normalization, morphosyntactic tagging, lemmatization and named entity recognition of non-standard Serbian.

As mentioned earlier, there is a corpus that has marked dependency syntax at the sentence level, but there is still no corpus for the Serbian language that contains fully marked syntax trees. Precisely for this reason, the idea arose to create such a tool that will enable the creation of a corpus of syntactic trees for the Serbian language.

There are some visualization tools for drawing syntax trees, but they are mostly limited to inputting a syntax tree, and then getting a visualization of that tree. The tools which expect user to enter syntax trees and then visualize it are shown here [11, 12]. The tool where the user can draw syntax tree from scratch is TreeForm [13]. This tool offers wide palette of elements that can be drawn in order to create a syntax tree. There are several simpler solutions than TreeForm, such as [14][15]. All these tools are only intended for visualizing syntax trees. They do not support using a syntax parser in the background, which would generate syntax trees based on the entered sentence as suggested in this paper.

III. The Functionality Of The Notation Tool

A. Syntax Analyzer

The parser that was created in [1] achieved excellent performance and performed real-time parsing. This parser consists of three components:

1. **POS Tagger** – when a sentence is forwarded to the syntax analyzer it is necessary to extract POS tags first, because a syntax analyzer can only recognize tags, not actual words. This tagger is explained in detail in [1]. The tagger returns tags that have special meaning. For example, some of the valid tags are 'nn' (noun in nominative), 'vm' (main verb), 'sl' (preposition in front of locative). Every POS of the Serbian language has its own abbreviation, where every letter has its own meaning. These tags are then forwarded to syntax analyzer, and later displayed in syntax trees above actual words of the sentence.
2. **CYK Parser** – this parser is implemented to achieve optimal performances while analyzing sentences. Also, this parser, as defined in [1], is capable of recognizing all the syntax trees, like the parser in NLTK [16], but with significantly reduced parsing time.
3. **Postprocessor** – this layer is added because the number of syntax trees that are generated based on grammar designed for CYK Parser was large. To reduce this number, a series of rules is defined. These rules eliminate syntax trees that aren't consistent with Serbian grammar. The postprocessing phase reduced the number of syntax trees by 54%.

The problem with this syntax analyzer, despite adding a postprocessing phase, is that it generated multiple trees for a single sentence. In order to solve this problem, it is necessary to add statistics that will enable to generate only one syntax tree as a result of a syntax analysis. To be able to implement statistical parsing, it is necessary to have a corpus of marked sentences, which is not the case for the Serbian language. For this reason, the idea of creating a visual tool arose. This tool will enable simple drawing and visualization of syntax trees and thus lead to the generation of a corpus of marked sentences that will be used further.

The notation tool works as follows:

1. The user enters the sentence they want to tag
2. The sentence is forwarded for processing to a parser that returns the resulting syntax trees
3. The syntax trees are displayed to the user
4. The user can choose one of the following options:
   - Select the correct tree,
   - Change the tree that is the most similar to the correct tree - by adding nodes, changing the node name, deleting nodes, and switching places with nodes, or
   - Create a new tree in case all the suggested trees are wrong
5. The correct tree is uploaded and stored in the database.

The architecture of the implemented system is shown in Figure 1.

The new component of the syntax analyzer is called reduction component. This component is added specifically for this tool.

The grammar created for the Serbian language contains a huge number of rules because the Serbian language is very complex. Considering that due to the implementation of the CYK algorithm, it was also necessary to transform the grammar so that it would be in Chomsky's normal form, a large number of auxiliary shifts were introduced. The syntax tree created in this way was too large to be displayed to the user of any system and this is the reason for introducing a reduction component.

This component aims to transform the syntax tree so that it no longer contains auxiliary rules, as well as that it does not contain shifts that have been introduced to make syntax analysis simpler and more robust.

The goal of reduction is to transform the syntax tree, generated by using a more complex grammar, into a simpler tree, corresponding to a simpler grammar. The main purpose for introducing the reduction component is to visualize the trees in a way that domain experts would expect by abstracting away implementation details. Also, reduced syntax trees are smaller and easier to display. After confirming the final tree for input sentence, it is necessary to return syntax tree to original form. This is achieved by using transform component. This component accepts syntax tree in simpler grammar and transforms that tree to original grammar. The transformed tree is then forwarded to backend application and saved in a database.

Figures 2 and 3 show how a part of the syntax tree looks with and without reduction. The reduced syntax tree is significantly smaller and thus much easier to display. An entire syntax tree without reduction would be impossible to fit in the page of the notation tool. The syntagm shown in figures 2 and 3 is "Moja divna drugarica", meaning "my wonderful friend" in Serbian.

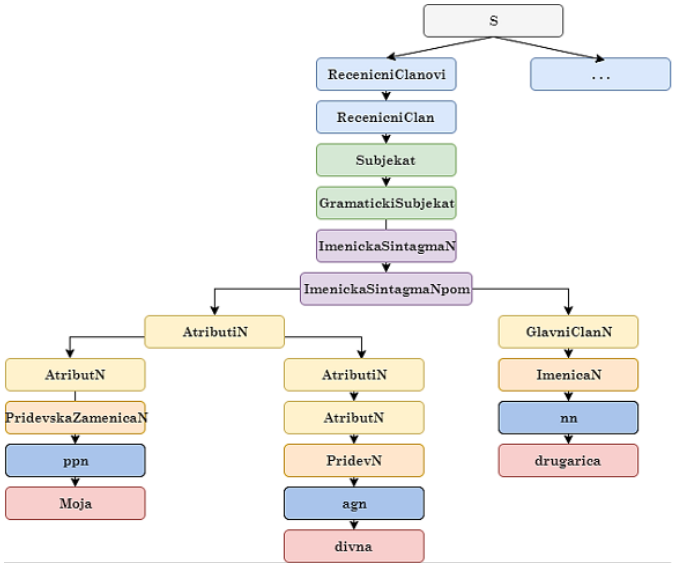

Fig. 2. Part of the syntax tree without reduction

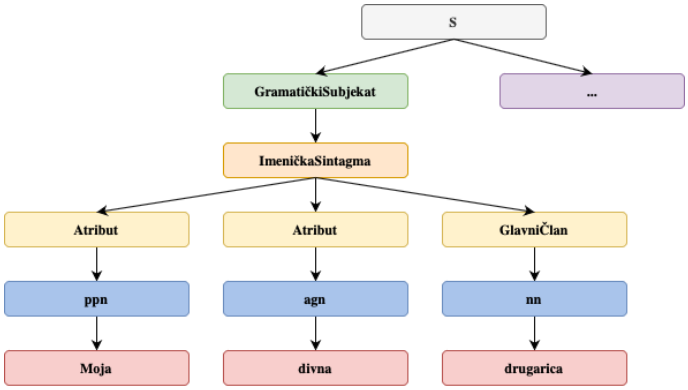

Fig. 3. Part of the syntax tree with reduction

B. *The Notation Tool*

This component is implemented as a web application so that users can use it as easily as possible. This approach was chosen to avoid any installation. The application itself is divided into three parts:

1. Frontend
2. Backend
3. Database.

The role of the Frontend is to enable:

1. **Entering a sentence whose analysis should be performed**
2. **Displaying of all syntax trees generated by the parser**
3. **Selecting a syntax tree that is correct** - the user can view a list of all syntax trees that the parser returned and check the one that is correct. This sends a request to the server with the intention to save that tree in the database.
4. **Syntax tree modification** - if the syntax tree is not completely correct, but with a few minor changes it could become correct, the tool offers the possibility to make the following syntax tree changes:

- *adding a new node* - it is necessary to select the node to which we want to add a new descendant and select the name that will be in that node. After interacting with the component, the tree structure is automatically updated to display the changes.
- *deleting a node* - if it is necessary to remove a node, the tool offers the option to select that node and then delete it.
- *renaming a node* - if the tree structure is adequate and an element is incorrectly recognized and it has the wrong name, the tool allows the user to rename that node.
- *swapping nodes* - this option exists in case the nodes are correctly recognized but they have been misplaced. It is possible to swap the places of these nodes, but only if they have the same parent. This option was introduced because a new node is always added to the end of the list of children, and if the node is deleted, a new node should be added in its place. Since the new node is always added as the last child, this functionality allows the user to place the node in any arbitrary position.

5. **Drawing a completely new tree** - the tool offers space for drawing a new tree, where on one side of the control there is a list of possible nodes and arrows for connecting nodes, and on the other side there is a space for drawing - canvas. It is possible to transfer nodes from the palette to the drawing space, as well as to connect these nodes with arrows. When nodes are added and names are populated, the tool offers the ability to make a tree structure out of these nodes, as well as to send that tree further to the server to be stored in the database.
6. **Sending a tree to the database** – within the tool there is a service whose methods are called to interact with the server.

The role of the backend application is to enable:

1. **Route for frontend application where a sentence can be analyzed** – when a frontend application sends GET request the backend application forwards this sentence to the syntax analyzer described earlier. This syntax analyzer is written in Python, so it is necessary to call Python script which returns generated syntax trees for given input.
2. **Route for saving the chosen tree in the database** – when an expert reviewed the syntax trees and chose or drew the correct one. This syntax tree is saved along with tags and sentence that has been analyzed.

IV. The Examples Of The Notation Tool

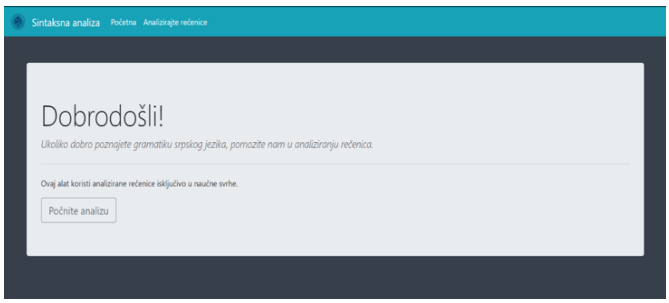

Fig. 4. Welcome page

Figure 4 shows the welcome page. There is a start analysis button that a user can click, and this will open a form for entering the sentence.

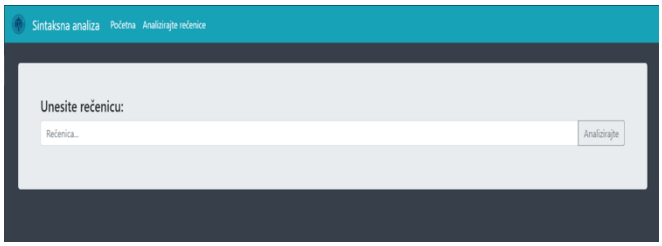

Fig. 5. Enter sentence form

Figure 5 shows a form where the user can enter a sentence for syntax analysis. That sentence is forwarded to the backend application. The backend application sends the sentence to the syntax analyzer by calling Python scripts.

The drawing of syntax trees was implemented using the canvas element in HTML and Canvas API in JavaScript. Below are shown pictures of different options which this tool offers.

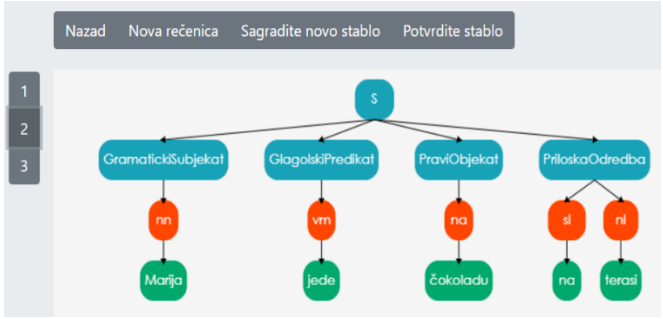

Fig. 6. One of the syntax trees that parser generated

Figure 6 shows the result that parser returned. As can be seen there are three syntax trees generated for this

sentence. The second syntax tree is shown in figure 4. The start symbol of the grammar is S, the level below S represents syntactic structures. After that, there are POS tags that tagger returned and finally words of the sentence. The menu above the drawn syntax tree has three options:

1. Interrupting the current analysis and analyzing a new sentence (leads to the form where the sentence is entered),
2. A page where it is possible to build a completely new tree for the current sentence,
3. Confirmation of the current tree - forwarding the selected tree by sending a POST request to the server, where the syntax tree is sent as the request body, after which the syntax tree for the entered sentence is stored in the database.

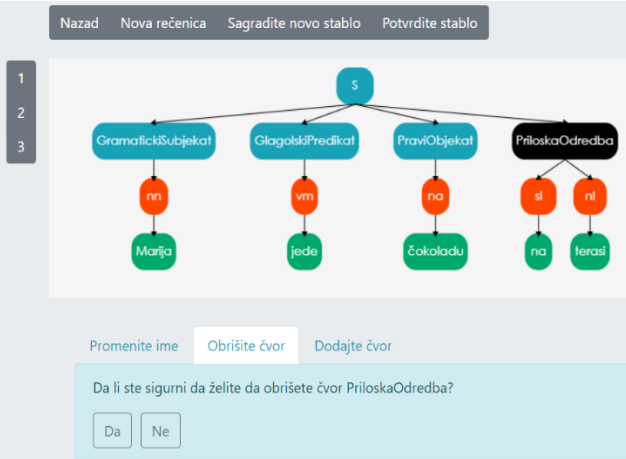

Fig. 7. Node removal

Figure 7 displays the menu for deletion of a node. First, it is necessary to select the node that is going to be altered. The selected node is colored black to stand out from other nodes. After node selection, the application opens the menu where the user can add a new node to the selected one, change the selected node's name or delete the selected node. Figure 7 shows that option for deletion is chosen. If the user wants to rename the node, it is necessary to select the rename option from the displayed menu. After that, the user needs to enter a new node name and confirm it. The third option in the menu is to add a new node. When a node to which a new node is added is selected, it is necessary to enter a name for the new node to be added.

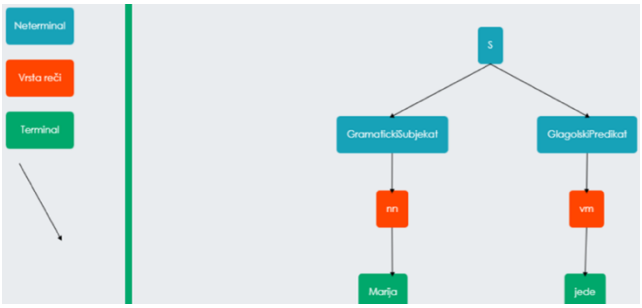

Fig. 8. Drawing syntax tree using canvas

Figure 8 shows the canvas where the user can draw a syntax tree from scratch.

V. Conclusion

The notation tool has been carefully created so that it has the simplest interface with the intention of being used primarily by domain experts - philologists. By using this tool, users are able to tag sentences in the simplest possible way, and thus quickly and efficiently create a corpus for the Serbian language.

After launching this site on the web, it is necessary to hire a set of domain experts who will tag sentences using the tool and create a corpus of sentences. After collecting a sufficient number of sentences, it is expected that these sentences will be used to further improve the Serbian language parser.

Acknowledgment

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

References

[1] T. Đorđević, S. Stojković: "Different Approaches in Serbian Language Parsing using Context-free Grammars", Proceedings of 7th International Conference on Electrical, Electronic and Computing Engineering IcETRAN, Etno-Selo Stanišići, Bosnia and Herzegovina (Online conference), pp. 588-591, September 28-30. 2020.

[2] N. W. Francis, H. Kucera, "Brown Corpus Manual" , Technical report, Department of Linguistics, Brown University, Providence, Rhode Island, US, 1979.

[3] M. Davies, *The Corpus of Contemporary American English*, 2008, www.english-corpora.org/coca/, 13.06.2021.

[4] English Corpora, https://www.english-corpora.org, 12.06.2021.

[5] The Quranic Arabic Corpus, https://corpus.quran.com, 10.06.2021.

[6] B. He, B. Dong, Y. Guan, J. Yang, Z. Yang, Q. Yang, J. Cheng, C. Qu, "Building a comprehensive syntactic and semantic corpus of Chinese clinical texts", *Journal of Biomedical Informatics*, vol. 69, pp. 203-217, 2017.

[7] V. Batanović, N. Ljubešić, T. Samardžić, "SETimes.SR – A Reference Training Corpus of Serbian", Conference on Language Technologies & Digital Humanities, Ljubljana, Slovenia, pp. 11-17, 2018.

[8] N. Ljubešić, F. Klubička, "{bs,hr,sr}WaC - Web Corpora of Bosnian, Croatian and Serbian", *Proceedings of the 9th Web as Corpus Workshop (WaC-9)*, Gothenburg, Sveden, pp. 29-35, April, 26. 2014.

[9] MULTEXT-East, https://www.clarin.si/repository/xmlui/handle/11356/1041, 10.06.2021.

[10] ReLDI-NormTagNER-sr 2.1, https://www.clarin.si/repository/xmlui/handle/11356/1240, 05.06.2021.

[11] Syntax tree generator, http://mshang.ca/syntree/, 26.07.2021.

[12] phpSyntaxTree, http://www.tycho.iel.unicamp.br/phpsyntaxtree/?, 26.07.2021.

[13] TreeForm, https://sourceforge.net/projects/treeform/, 26.07.2021.

[14] Trees, https://www.ling.upenn.edu/~kroch/gifdir/Trees3-animation.GIF, 27.07.2021.

[15] Linguistic Tree Constructor, https://ltc.sourceforge.io/screenshots.html, 27.07.2021.

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