AHLT Assignment Report: Regular Expressions Parser

Leonardo Maurins
B00107064

Department of Informatics
Technological University of Dublin,
Blanchardstown,
Dublin 15

Declaration of Academic Honesty

I/We declare that this material, which I/We now submit for assessment, is entirely my/our own work and has not been taken from the work of others, except where otherwise stated. I/We have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged, and the source cited are identified in the assignment references.

I/We understand that plagiarism, collusion, and copying are grave and serious offences and accept the penalties that would be imposed should I/we engage in plagiarism, collusion or copying. I acknowledge that copying someone else's assignment, or part of it, is wrong, and that submitting identical work to others constitutes a form of plagiarism. I/We have read and understood the colleges plagiarism policy 3AS08.

This material, or any part of it, has not been previously submitted for assessment for an academic purpose at this or any other academic institution. I/We have not allowed anyone to copy my/our work with the intention of passing it off as their own work.

Name:	**Signature:**
Leonardo Maurins	______________________

1 Introduction

This report will go into detail regarding the parsing of regular expressions from sentences passed to the interface. The application's parser receives sentences and analyses the text to judge whether it is to be accepted or rejected as a valid regular expression. Section 2 of the report covers a description of the lexicon categories implemented (i.e. verbs, nouns, adjectives etc.). In section 3, the implementation of the parser program and interaction with the lexical entries is reviewed. Finally, an overall review of the system architecture and design aspect of the program is outlined in section 4.

2 Lexicon

The sentences which were provided to use consisted of "The/a man/men/woman/women bite(s)/like(s) the green dog". Every word in the sentence arrangement was directly correlated with a part of speech category and a numerical association for the type of subject-verbs (Singular or plural). For every part of speech category there were abbreviations received from the CoreNLP library. It allows the users to obtain linguistic annotations for text passed to it, which includes tokens and sentence boundaries, parts of speech, named entities, numeric entities etc [1]. An example of the generated linguistic annotations can be seen in Figure 1.

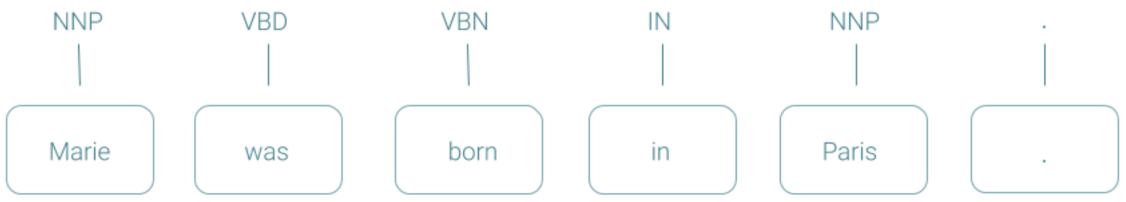


Figure 1 CoreNLP Part of Speech Annotations

The full list of linguistic annotations used; we can see in the list below.

- Common Nouns – NN
- Noun Plural - NNS
- Determiners - DT
- Verb 3rd Person Singular - VBZ
- Verb Base - VB
- Adjectives - JJ

Hence with the annotation linguistics detailed as part of speech, the words used in the sentence and the numbers associated with singular or plural form the table below.

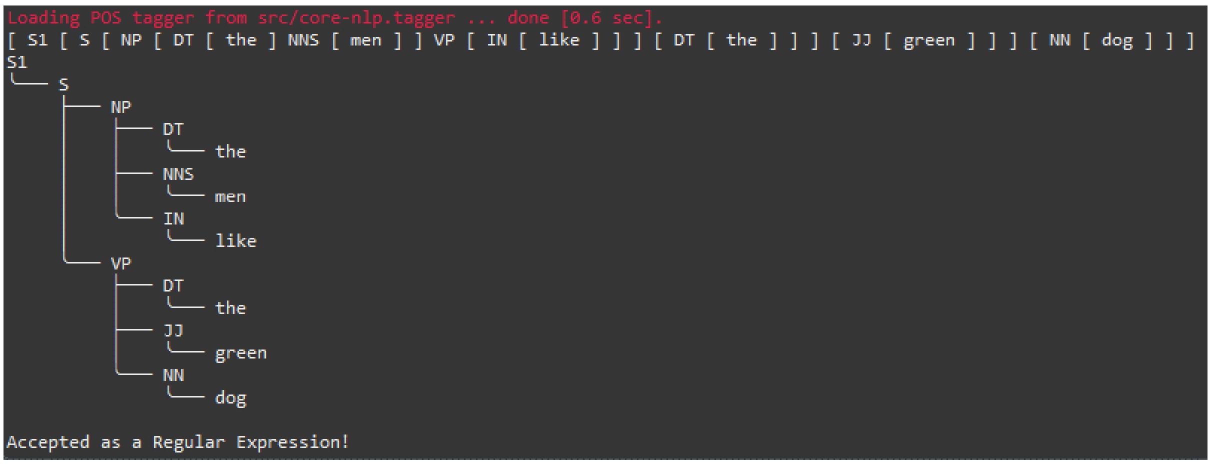
Words	Part of Speech	Subject-Verbs
The	DT	Singular
A	DT	Singular
Man	NN	Singular
Men	NNS	Plural
Woman	NN	Singular
Women	NNS	Plural
Like	VB	Singular
Likes	VBZ	Singular
Bite	VB	Singular
Bites	VBZ	Singular
Green	JJ	Singular
Dog	NN	Singular

Table 1 Lexicon Categories

3 Implementation

The parser program's original intention was to allow for user input using Scanner to provide sentences, but this was changed for a simpler testing application. Instead providing four sentences with two that are accepted and two rejected as regular expressions to demonstrate the functionality.

The input, which is considered as the passed sentences is passed towards the appropriate class to be tokenized by CoreNLP. Once annotations are associated with each word, the phrases are separated in while loop to create different arrays of words and part of speech tags. Using the java Scanner class, the tokens are compared across the Lexicon.txt file to verify for valid annotations to exist. If successfully verified, the part of speech tags are separated from the word phrases and put into a different ArrayList. The sentence is created in a bracketed phrasal structure and printed to the console. The part of speech tags are checked against a set of built in rules to be either accepted or rejected as a regular expression. An example of an accepted regular expression can be seen below in Figure 2.

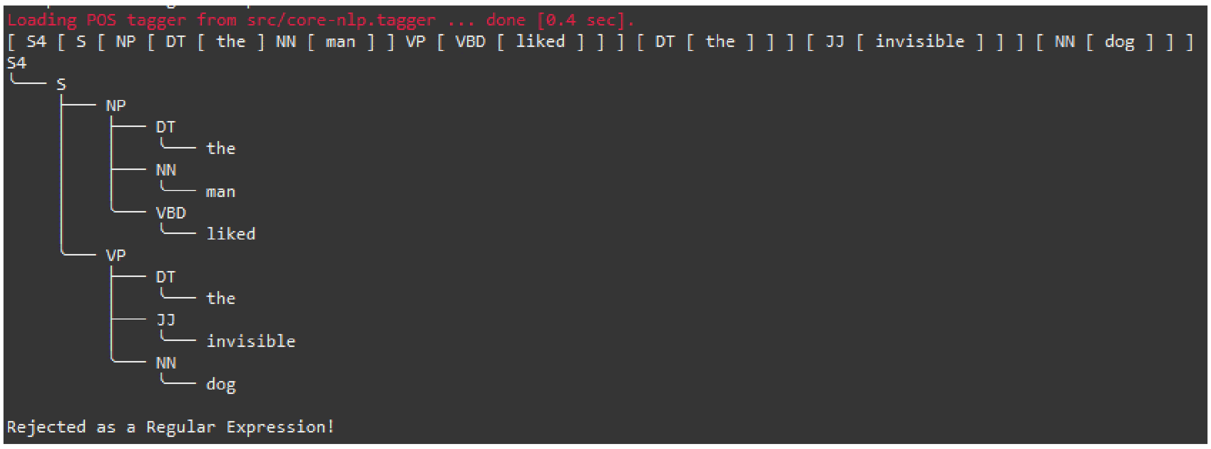


```
Loading POS tagger from src/core-nlp.tagger ... done [0.6 sec].
[ S1 [ S [ NP [ DT [ the ] NNS [ men ] ] VP [ IN [ like ] ] ] [ DT [ the ] ] ] [ JJ [ green ] ] ] [ NN [ dog ] ] ]
S1
└── S
    ├── NP
    │   ├── DT
    │   │   └── the
    │   ├── NNS
    │   │   └── men
    │   └── IN
    │       └── like
    └── VP
        ├── DT
        │   └── the
        ├── JJ
        │   └── green
        └── NN
            └── dog

Accepted as a Regular Expression!
```

Figure 2 Accepted Regular Expression

If the sentence does not match the structure of the lexicon text when parsed, it will reject the regular expression as seen in Figure 3.



```
Loading POS tagger from src/core-nlp.tagger ... done [0.4 sec].
[ S4 [ S [ NP [ DT [ the ] NN [ man ] ] VP [ VBD [ liked ] ] ] [ DT [ the ] ] ] [ JJ [ invisible ] ] ] [ NN [ dog ] ] ]
S4
└── S
    ├── NP
    │   ├── DT
    │   │   └── the
    │   ├── NN
    │   │   └── man
    │   └── VBD
    │       └── liked
    └── VP
        ├── DT
        │   └── the
        ├── JJ
        │   └── invisible
        └── NN
            └── dog

Rejected as a Regular Expression!
```

Figure 3 Rejected Regular Expression

4 Architecture

The system architecture primarily consists of four classes responsible for the parsing of sentences as seen in the following sections. Section 4.1 covers the main class, Parser, which is responsible for creating the sentences and passing them to be parsed. In section 4.2, the ParseSentence class is reviewed regarding the sentences passed to it and use the of taggers. For section 4.3, the output of the sentence after being split and arranged in the bracketed structure with the tree is outlined. Finally, the verification of the regular expression is explained under Section 4.4. A class diagram of the architecture can be seen in Figure 4 below.

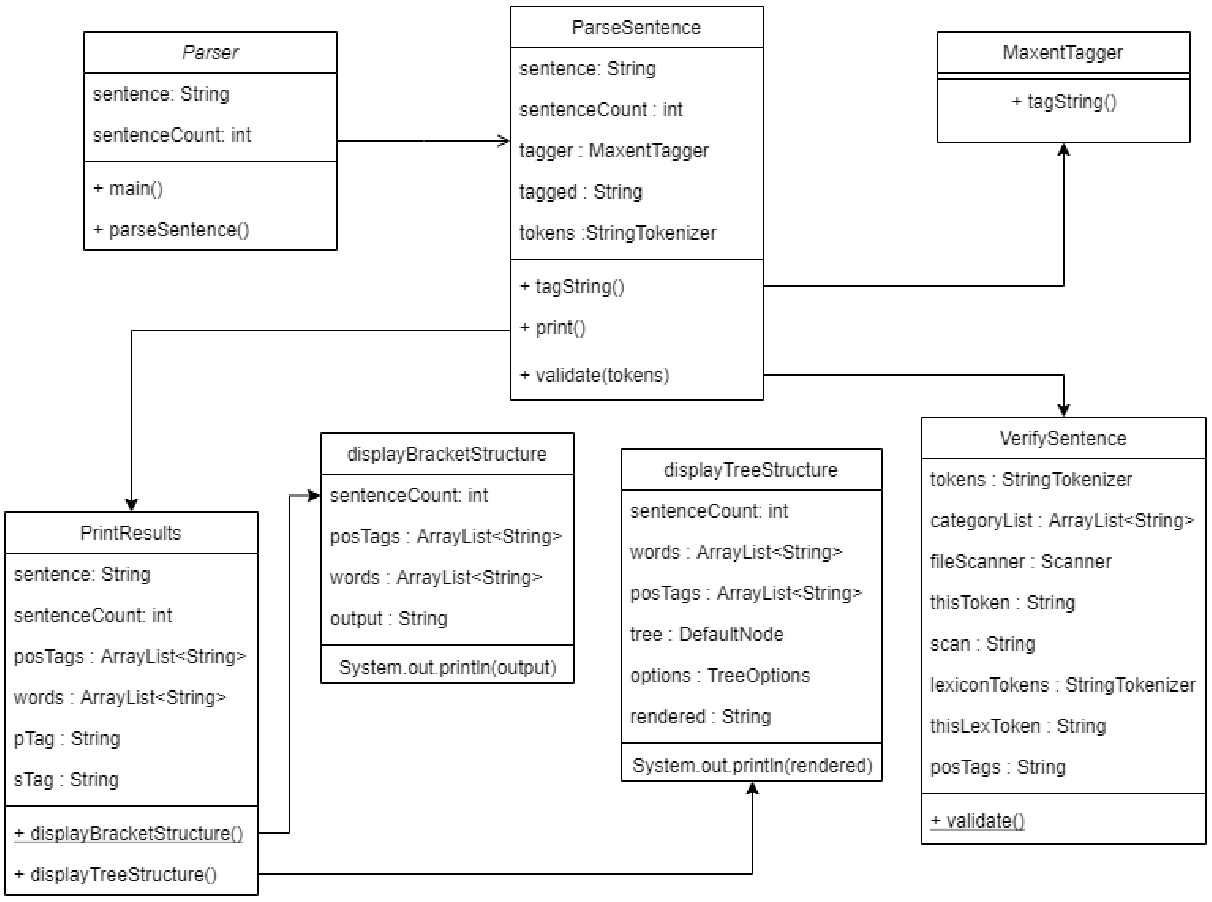


Figure 4 System Architecture Class Diagram

4.1 Parser

The main parser application creates a list of strings that are sentences and adds them to an ArrayList. It then for loops through the array list passing each sentence to the ParseSentence class to be parsed and compared against the lexicon sentence.

4.2 ParseSentence

The ParseSentence class receives the string sentences from the previous class and compared it to the lexicon text file. It proceeds to tag any words with linguistic annotations if found by the CoreNLP library. Once the tagged set of phrases are created and added to a new string, it is passed to the PrintResults class. There are also tokens created by StringTokenizer which are passed to VerifySentence class to determine if it is an accepted or rejected regular expression.

4.3 PrintResults

This class is responsible for the bracketed phrasal structure and parsed tree display. It takes in the tokenized sentence and splits the words from the part of speech annotations into arraylists. This is required due to how the CoreNLP library tokenizes the phrases as “the_DT”. Once successfully separated, the displayBracketStructure method is called to arrange the sentence. Alongside with the displayTreeStructure method to create a tree structure using nodes. A text-tree library [2] was used for this process to provide customisation such as styles and rendering.

4.4 VerifySentence

Finally, the VerifySentence class was used to validate if the sentence passed to it was an acceptable regular expression and returned the result. The class receives tokens when called which contains the different lexicon categories. This is compared against the lexicon text file using java’s Scanner utility. Once all the part of speech tags are collected and added to an array list for categories, it is passed for a loop to cross-check against set rules of verification. If the categories meet a specific set of requirements, the sentence is parsed as an accepted regular expression.

5 Conclusion

In conclusion, the software was capable of parsing sentences passed to it and determining if it was valid as a regular expression or not. The sentence was printed as a bracketed phrasal structure in the console along with the tree structure. The appendix for the source code can be found below in the Appendix section 7.

6 References

[1] C. D. M. S. J. B. J. F. S. J. B. a. D. M. Manning, “The Stanford CoreNLP Natural Language Processing Toolkit,” Stanford University, 2014.

[2] Barfuin, “text-tree,” GitLab, [Online]. Available: https://gitlab.com/barfuin/text-tree.

7 Appendices

7.1 Parser

```java
import java.util.ArrayList;
import java.util.List;

public class Parser {

    public static void main(String[] args) {

        // Creates a list of strings to be checked for acceptable regular expressions
        List<String> sentences = new ArrayList<String>();
        sentences.add("The men like the green dog"); // Accepted
        sentences.add("The woman pet the blue dog"); // Rejected
        sentences.add("A woman bites the green dog"); // Accepted
        sentences.add("The man liked the invisible dog"); // Rejected

        ParseSentence parseSentence = new ParseSentence();
        String sentence;
        int sentenceCount = 1;

        // Loops through the sentences to be parsed
        for(int i = 0; i < sentences.size(); i++) {
            sentence = sentences.get(i);
            parseSentence.parseSentence(sentence, sentenceCount);
            sentenceCount++;
        }
    }
}
```

7.2 ParseSentence

```java
import java.util.StringTokenizer;


import edu.stanford.nlp.tagger.maxent.MaxentTagger;

public class ParseSentence {

    public void parseSentence(String sentence, int sentenceCount) {

        // Made lower case to prevent interference with lexicon
        sentence = sentence.toLowerCase();

        // Tag the users input using the CoreNLP library
        MaxentTagger tagger = new MaxentTagger("src/core-nlp.tagger");
        String tagged = tagger.tagString(sentence);

        // Calls the print results class in order to print the
        // bracketed phrase structure and the tree architecture
        PrintResults printResults = new PrintResults();
        printResults.print(tagged, sentenceCount);

        // Calls the verify sentence class to check if the passed sentence
        // is a valid regular expression using tokens
        StringTokenizer tokens = new StringTokenizer(sentence);
        VerifySentence validateSentence = new VerifySentence();
        validateSentence.validate(tokens);
    }
}
```

7.3 PrintResults

```
import java.util.ArrayList;
import java.util.StringTokenizer;

import org.barfuin.texttree.api.DefaultNode;
import org.barfuin.texttree.api.TextTree;
import org.barfuin.texttree.api.TreeOptions;
import org.barfuin.texttree.api.style.TreeStyles;

public class PrintResults {

    ArrayList<String> posTags, words;

    public void print(String sentence, int sentenceCount) {

        // Tagged output from the MaxentTagger is tokenized
        StringTokenizer taggedTokens = new StringTokenizer(sentence);

        // Arraylists of the posTags and words are split for parsing
        posTags = new ArrayList<String>();
        words = new ArrayList<String>();

        // Separates the tagged regular expresisons from the stanford library
        // Splits posTags and words into different arraylists
        while (taggedTokens.hasMoreTokens()) {
            String pTag = taggedTokens.nextToken();
            String sTag = pTag;
            pTag = pTag.replaceAll("[a-z]+_", "");
            sTag = sTag.replaceAll("_[A-Z]+", "");
            posTags.add(pTag);
            words.add(sTag);
        }
        displayBracketStructure(sentenceCount);
        displayTreeStructure(sentenceCount, posTags, words);
    }

    // Parses the words and nouns, adjectives, verbs etc then sorts them into a
    // bracketed phrasal structure
    public void displayBracketStructure(int sentenceCount) {
        String output = "[ S" + sentenceCount + " [ S [ NP [ " + posTags.get(0) + " [
" + words.get(0) + " ] "
                + posTags.get(1) + " [ " + words.get(1) + " ] ] VP ";
        for (int i = 2; i < posTags.size(); i++) {
            output += "[ " + posTags.get(i) + " [ " + words.get(i) + " ] ] ] ";
        }

        System.out.println(output);
    }

    // Creates a tree architecture using passed POS tags and words from parsing
    // We build the tree using a java library "text-tree"
    public void displayTreeStructure(int sentenceCount, ArrayList<String> posTags,
ArrayList<String> words) {
    // Creates a tree architecture using passed POS tags and words from parsing
    // We build the tree using a java library "text-tree"
    public void displayTreeStructure(int sentenceCount, ArrayList<String> posTags,
ArrayList<String> words) {

        DefaultNode tree = new DefaultNode("S" + sentenceCount);
        DefaultNode node1 = new DefaultNode("S");
        DefaultNode node2 = new DefaultNode("NP");
        DefaultNode node3 = new DefaultNode("VP");
        DefaultNode node4 = new DefaultNode(posTags.get(0));
        DefaultNode node5 = new DefaultNode(posTags.get(1));
        DefaultNode node6 = new DefaultNode(posTags.get(2));
        DefaultNode node7 = new DefaultNode(posTags.get(3));
        DefaultNode node8 = new DefaultNode(posTags.get(4));
        DefaultNode node9 = new DefaultNode(posTags.get(5));

        tree.addChild(node1);
        node1.addChild(node2);
        node1.addChild(node3);
        node2.addChild(node4);
        node2.addChild(node5);
        node2.addChild(node6);
        node3.addChild(node7);
        node3.addChild(node8);
        node3.addChild(node9);

        node4.addChild(new DefaultNode(words.get(0)));
        node5.addChild(new DefaultNode(words.get(1)));
        node6.addChild(new DefaultNode(words.get(2)));
        node7.addChild(new DefaultNode(words.get(3)));
        node8.addChild(new DefaultNode(words.get(4)));
        node9.addChild(new DefaultNode(words.get(5)));

        TreeOptions options = new TreeOptions();
        options.setStyle(TreeStyles.UNICODE_ROUNDED);
        String rendered = TextTree.newInstance(options).render(tree);

        System.out.println(rendered);
    }
}
```

7.4 VerifySentence

```java
import java.util.ArrayList;

import java.util.Scanner;
import java.util.StringTokenizer;

public class VerifySentence {

    public void validate(StringTokenizer tokens) {

        Scanner fileScanner = null;
        ArrayList<String> categoryList = new ArrayList<String>();

        // While traversing the tokens from sentences compare them to lexicon text file
        while (tokens.hasMoreTokens()) {

            fileScanner = new Scanner(getClass().getResourceAsStream("Lexicon.txt"));
            String thisToken = tokens.nextToken();

            // While traversing the lines in the text file
            // Tokenizes the word for each line
            while (fileScanner.hasNextLine()) {
                String scan = fileScanner.nextLine().toString();
                StringTokenizer lexiconTokens = new StringTokenizer(scan);

                // While traversing the created tokens, adds them to
                // the POS categories list for verification
                while (lexiconTokens.hasMoreTokens()) {
                    if (thisToken.equals(lexiconTokens.nextToken())) {
                        String thisLexToken = lexiconTokens.nextToken();
                        categoryList.add(thisLexToken);
                    }
                }
            }
        }
        fileScanner.close();

        // Add each POS to a concatenated string
        String posTags = "";
        for (int i = 0; i < categoryList.size(); i++) {
            posTags += categoryList.get(i) + " ";
        }

        // Checks if the order of POS categories is valid
        // for the sentence and either accepts or rejects it
        if (posTags.equals("DTS NN VBZ DT JJ NN ") ||
            posTags.equals("DT NN VBZ DT JJ NN ") ||
            posTags.equals("DT NNS IN DT JJ NN ")) {
            System.out.println("Accepted as a Regular Expression!");
        } else {
            System.out.println("Rejected as a Regular Expression!");
        }
    }
}
```