Institute of Technology Blanchardstown
Institiúid Teicneolaíochta Baile Bhlainséir

Applied Human Language Technology

Assignment: Sentence Parser

Using Java Programming

Student
Derek McCarthy B00007439

Lecturer
Irene Murtagh

Contents

Introduction

This report outlines the design and implementation of regular expression parser using Java programming. There are three main sections to this report Lexical, Rules and Parser implantation. In the Lexical section we will discuss the implementation and architecture of the Lexicon into the program. In the Syntax Rules section, we will discuss the rule system we have implemented to check if a sentence is grammatical. And finally, in the Parser Program section we will discuss how we implemented the parser into the program. Also, in the Program Walkthrough section we will explain how the program works with the different user interactions.

Lexical Categories

The lexicon was created from the following sentence, *"The/a man/men/woman/women bite(s)/like(s) the green dog".* To create the lexicon, we separated the words into a text file along with their Part of Speech (POS) tag. The POS tags we used to categorize the words was the Penn Treebank [1]. The POS tags we used in our lexicon file can be seen in table 1.

Table 1 - POS Tags and Descriptions

POS Tag	POS Tag Description
DT	Determiner
NN	Noun, singular or mass
NNS	Noun, plural
VBZ	Verb, 3rd person singular present
VB	Verb, base form
IN	Preposition or subordinating conjunction
JJ	Adjective

Table 2 is the lexicon which we implemented into the program. In the left column is all the words from the provided sentence. And in the right column is the POS tag we have assigned to the correspondent word. In total there are 12 words and POS tags.

Table 2 - Lexicon Words and Tags

Word	POS Tag
the	DT
a	DT
man	NN
woman	NN
dog	NN
men	NNS

women	NNS
likes	VBZ
bites	VBZ
bite	VB
like	IN
green	JJ

Syntax Rules

To create the ruleset for the program we first had to list the different combinations of words that could form a grammatically correct sentence. To help identify theses different combinations of words we used the link.cs.cmu.edu parser [2]. The sentences that we identified that could be grammatically constructed from the sentence are,

1. The/A Man/Woman Likes/Bites The/A Green Dog
2. The/A Man/Woman Like The/A Green Dog
3. The/A Men/Women Like The/A Green Dog
4. The/A Men/Women Likes/Bites The/A Green Dog

The syntax rules we devised that governs that sentences are grammatically correct from the above sentences are,

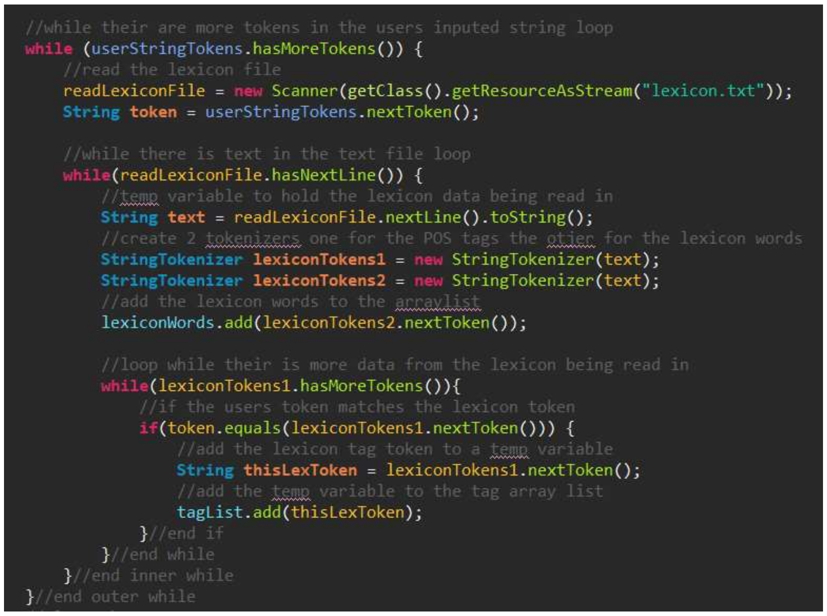
1. DT NN VBZ DT JJ NN
2. DT NN IN DT JJ NN
3. DT NNS IN DT JJ NN
4. DT NNS VBZ DT JJ NN

These rules are stored in a text file within the programs source files which are read into the program at run time to validate if a sentence syntax is acceptable. It does this by comparing the ruleset with the POS tags from the user's sentence.

Program Implementation

When the program is initially started it displays a Graphical User Interface (GUI) with a text field for the user to enter their sentence and a button to parse this sentence. Once the user has entered a sentence and clicked parse. The program then validates the inputted data. If valid data has being entered (i.e. at least one character) the program starts the parsing process.

To begin the parsing process the user's data is dissembled into tokens using the JAVA String Tokenizer library. The way the data is disassembled is by white space between words. After this we then start an iteration process through the tokens read in from the lexicon text file. As the lexicon file is being read into the system each lexicon entry is also tokenized. Still within the iteration process we use a conditional statement to check if each token from the user's data is the same as the token from the lexicon file. If the two tokens match the token is added to an array list this can be seeing in figure 1.



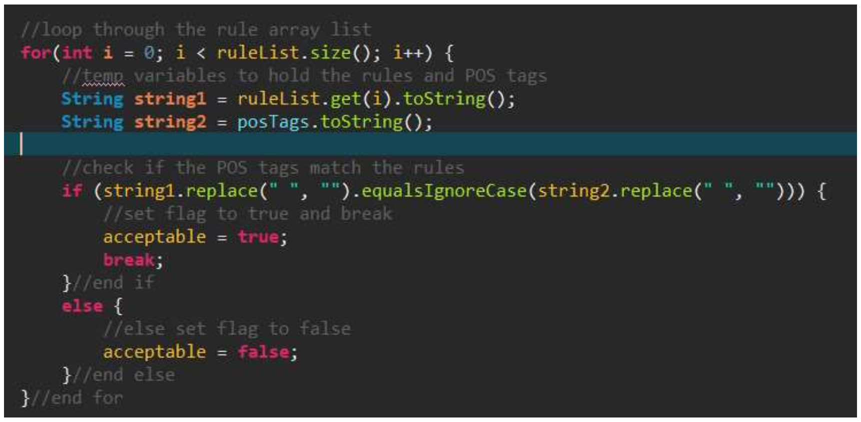
```
//while their are more tokens in the users inputed string loop
while (userStringTokens.hasMoreTokens()) {
    //read the lexicon file
    readLexiconFile = new Scanner(getClass().getResourceAsStream("lexicon.txt"));
    String token = userStringTokens.nextToken();

    //while there is text in the text file loop
    while(readLexiconFile.hasNextLine()) {
        //temp variable to hold the lexicon data being read in
        String text = readLexiconFile.nextLine().toString();
        //create 2 tokenizers one for the POS tags the otier for the lexicon words
        StringTokenizer lexiconTokens1 = new StringTokenizer(text);
        StringTokenizer lexiconTokens2 = new StringTokenizer(text);
        //add the lexicon words to the arraylist
        lexiconWords.add(lexiconTokens2.nextToken());

        //loop while their is more data from the lexicon being read in
        while(lexiconTokens1.hasMoreTokens()){
            //if the users token matches the lexicon token
            if(token.equals(lexiconTokens1.nextToken())) {
                //add the lexicon tag token to a temp variable
                String thisLexToken = lexiconTokens1.nextToken();
                //add the temp variable to the tag array list
                tagList.add(thisLexToken);
            }//end if
        }//end while
    }//end inner while
}//end outer while
```

Figure 1 - Code Snippet of Parsing Process

After the tokens have being added to the array list the rules that determine if a sentence is grammatical are read in from the text file (see Syntax Rules) to an array list. After this we then can check if the sentence is acceptable by iterating through the rule list and comparing the part of speech tags from the users sentence with the tags from the ruleset (see figure 2).



```
//loop through the rule array list
for(int i = 0; i < ruleList.size(); i++) {
    //temp variables to hold the rules and POS tags
    String string1 = ruleList.get(i).toString();
    String string2 = posTags.toString();

    //check if the POS tags match the rules
    if (string1.replace(" ", "").equalsIgnoreCase(string2.replace(" ", ""))) {
        //set flag to true and break
        acceptable = true;
        break;
    }//end if
    else {
        //else set flag to false
        acceptable = false;
    }//end else
}//end for
```

Figure 2 - Code Snippet of Rules Being Applied

Program Architecture

The program uses several classes and files in the architecture of the program design. The files the program uses are two text files which hold the lexicon and the rules and the Stanford NPL tagger [3]. The Stanford tagger is only used for sentences that are not grammatical or for words not found in the lexicon. The java classes the program uses are UserInterface, RunMain, TokenizeUserInput and BracketedPhrasalStructureBuilder.

Run Main – The class containing the main method to run the program.

User Interface Class – Main purpose is to provide a graphical user interface to the user. From this GUI the user can enters a sentence which this class then tokenizes and then passes it to the TokenizeUserInput class for processing.

Tokenize User Input Class – This class receives the users tokenized sentence and then begins the process of validating the sentence. To check if it's a valid sentence the program iterates through the user's tokens and adds the POS tag to an array list. As its doing this it begins to read in the lexicon text file. As the lexicon is being read in it is also being tokenized and added to an array list. It then reads in the ruleset outlined in Syntax Rules which are added to another array list. Then the program checks these to see if the users POS tags match any of those within the ruleset. If they do match the program returns true to the UI class or else if they don't match, then false is returned. If true is returned the UI class displays that the sentence is acceptable and calls the BracketedPhrasalStructureBuilder Class to display the bracketed results of the sentence. If false is returned, then unacceptable is displayed on the UI.

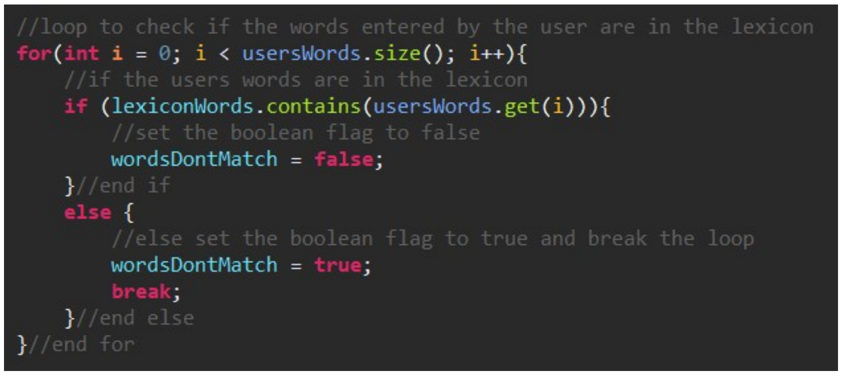
Bracketed Phrasal Structure Builder Class – The only purpose of the class is to add phrasal structure brackets to the users tags to be displayed on the GUI.

Challenging Aspects of the Program Architecture

Throughout the construction of the program there was several issues that needed to be addressed. These are the issues we faced and our solution to fix/implement them.

- **How to disassemble the users sentence** – To disassemble the users sentence we identified we need to use a String Tokenizer. This was also used for the reading in of the lexicon text file.
- **How to parse the sentence –** To parse the sentence we implement nested while loops to loop through the tokens from the user's sentence. And another to read in the lexicon and finally a third to add the lexicon tokens to an array list for comparison later.

- **Determine if words from sentence were part of the lexicon –** A problem we noticed while testing the program was when the user entered a valid sentence but added extra words to the sentence that were not part of the lexicon. The sentence was being accepted as being valid as it was ignoring the word(s) that was not art of the lexicon. To combat this, we created a loop which compared the user's words stored in an array list to the lexicon entries also stored in an array list. If the words matched, we set a Boolean flag to false which then continued to apply the ruleset. If the words did not match we set the flag to true and broke from the loop informing the user, the word entered was not found in the lexicon.



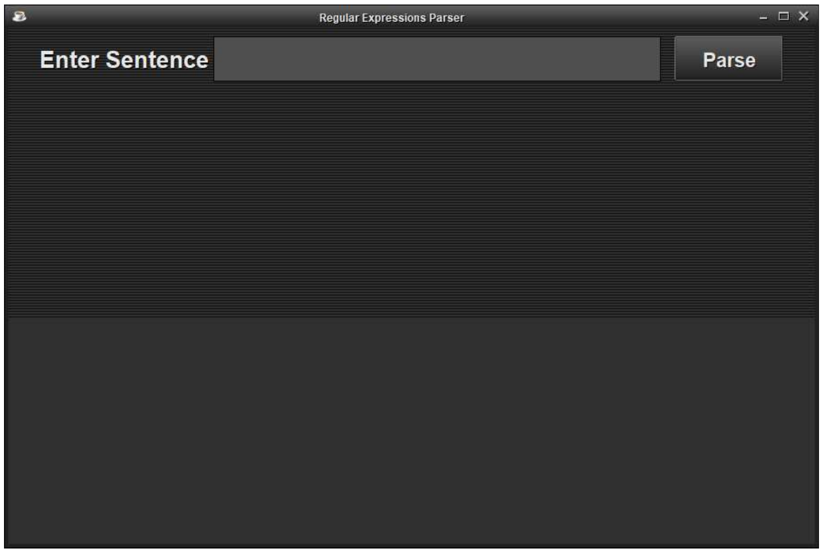
```
//loop to check if the words entered by the user are in the lexicon
for(int i = 0; i < usersWords.size(); i++){
    //if the users words are in the lexicon
    if (lexiconWords.contains(usersWords.get(i))){
        //set the boolean flag to false
        wordsDontMatch = false;
    }//end if
    else {
        //else set the boolean flag to true and break the loop
        wordsDontMatch = true;
        break;
    }//end else
}//end for
```

Figure 3 - Code Snippet of Checking if Words Are Part of Lexicon

Program Walkthrough

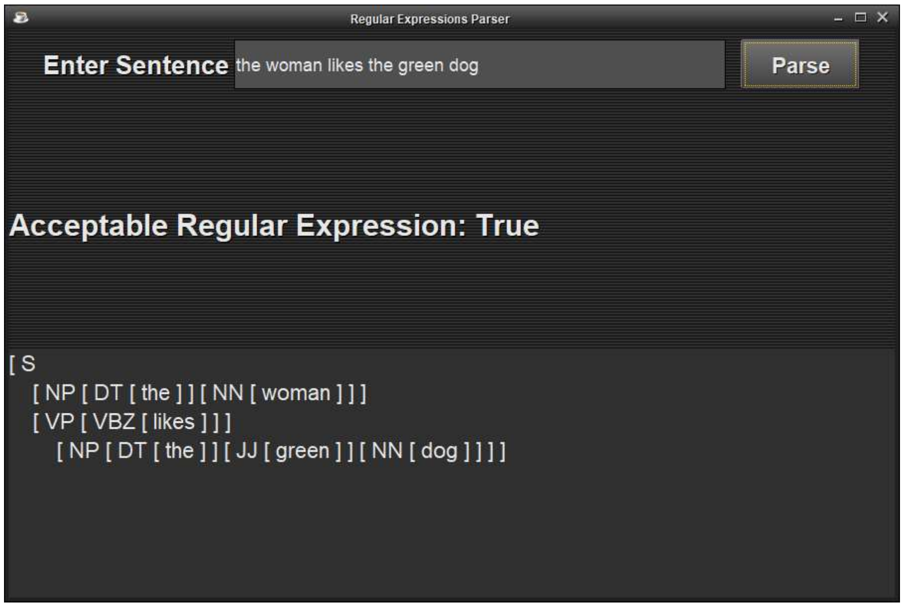
In this section we will give a detailed walk through of using the parser program. We will show the results of entering grammatically correct and incorrect sentences.

Opening Screen



Initially when the program executes the user is met with a graphical user interface with a text box to input a sentence and a button to check if its grammatically valid or invalid. In the following screenshots we will demonstrate entering data into the text box and clicking the parse button.

If the user enters a grammatically valid sentence

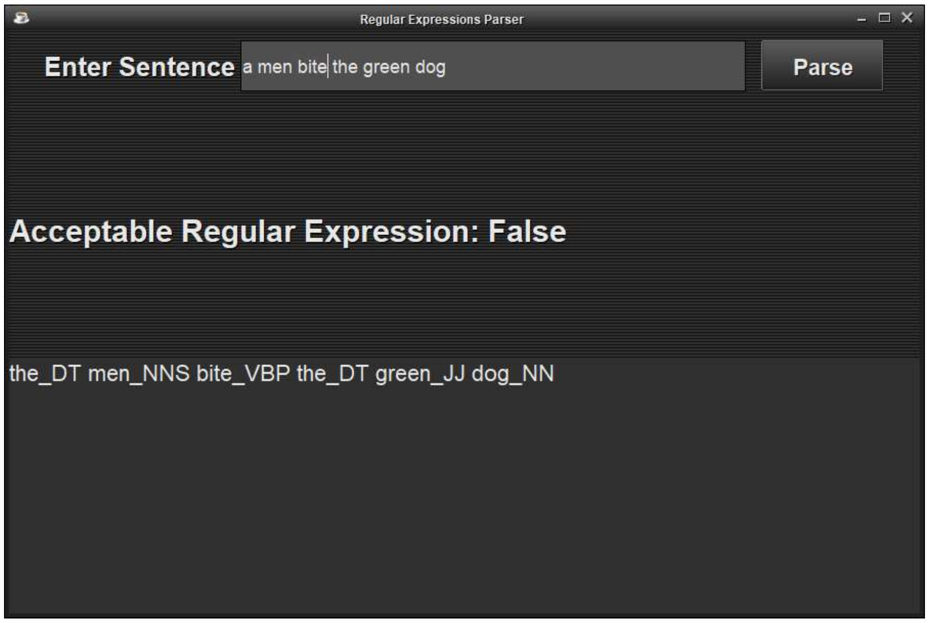


Acceptable Regular Expression: True

```
[ S
   [ NP [ DT [ the ] ] [ NN [ woman ] ] ]
   [ VP [ VBZ [ likes ] ] ]
      [ NP [ DT [ the ] ] [ JJ [ green ] ] [ NN [ dog ] ] ] ]
```

If the user enters a grammatically correct sentence such as *"the woman likes the green dog",* the program states that the sentence is valid, and the bracketed result of the sentence is displayed.

If the user enters a grammatically invalid sentence

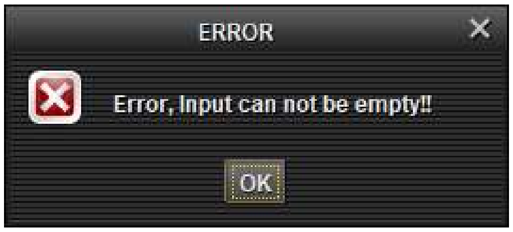


Acceptable Regular Expression: False

```
the_DT men_NNS bite_VBP the_DT green_JJ dog_NN
```

If the user enters a grammatically incorrect sentence such as *"a men bite the green dog",* the program states that the sentence is invalid, and just displays each word with its correspondent POS tag.

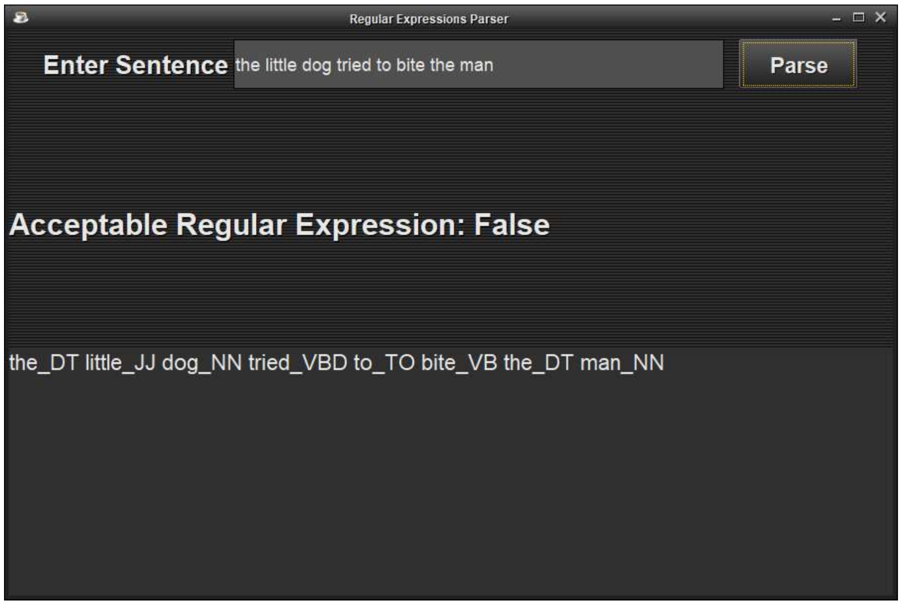
If the user doesn't enter any text



Error, Input can not be empty!!

An error message appears informing the user that the input cannot be empty.

If the user enters a sentence that contains words not found in the lexicon



the little dog tried to bite the man

Acceptable Regular Expression: False

```
the_DT little_JJ dog_NN tried_VBD to_TO bite_VB the_DT man_NN
```

If the user enters a sentence that contains words not found in the lexicon such as *"the little dog tried to bite the man",* the program states that the sentence is invalid, and just displays each word with its correspondent POS tag. To be able to display the correct tag for words that are not found in the lexicon the Stanford NLP tagger is used.

Conclusion

In conclusion of the finished program we are pleased with how the program has being developed. It correctly identifies syntax correct sentences based on the ruleset we outlined. If time was not a factor we would have included the ANTLR suit to generate a visual tree of the phrase structure. Also, we would have preferred to implement some way of distinguishing between ‘the’ and ‘a’. As both these words are determiners we couldn’t find a way to distinguish between them when constructing sentences. So, sentences such as “A women like the green dog” and “The women like the green dog” are acceptable by our ruleset and the program but only the latter is grammatically acceptable so to speak. As both these sentences use the same POS tags, DT NNS IN DT JJ NN. The only way we could envisage of fixing this was to omit this POS tag rule out of the ruleset but decided against this. Other than this we happy with the finished program.

References

[1] https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html [Online: Accessed on 28/10/2018]

[2] https://www.link.cs.cmu.edu/cgi-bin/link/construct-page-4.cgi#submit [Online: Accessed on28/10/2018]

[3] https://nlp.stanford.edu/software/pos-tagger-faq.html [Online: Accessed on 29/10/2018]