Applied Human Language Technology Assignment

Dylan Seery B00098463

November 2019

1 Introduction

The following assignment asks us to describe, design and implement in software a parser for regular expressions using the programming language of your choice. Section one will discuss how lexical categories where implemented section two will discuss how we implemented our parser program and section 3 will consist of the system architecture. We will then finalise the document with screenshots of our program execution. The programming language chosen was python 3.6.

2 lexical categories

To assign our lexical categories we created a CSV file. Due to the ease of implementation for reading it in using python 3.6. Since we knew the input was limited to a series of given sentences we could structure our CSV file that it would only apply verbs, nouns, adjectives etc to words that could possibly appear. You can see the layout of the lexical category CSV file used in table 1.

Word	Lexical category	Type
a	DET	S
the	DET	S
man	N	S
men	N	P
woman	N	S
women	N	P
bite	V	S
bites	V	P
like	V	S
likes	V	P
green	ADJ	S
dog	N	P

Table 1: CSV File

Another CSV file was used for rules. This file contained all the rules that the possible sentence must follow. The rules CSV file is broken into two columns parent and children columns. The parent column, for example, would hold NP (noun phrase) while for this rule the child column would hold the values determiner and noun. These children values are nested to create a parent value this is how we create our structure. The layout of the rules CSV file is displayed at table 2.

parent	child
S	NP VP
NP	DET N
VP	V NP
NP	DET ADJ N

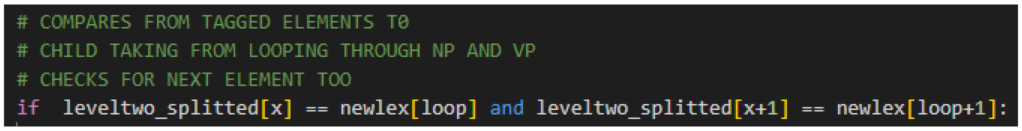
Table 2: CSV File Rules

3 Parser program

As mentioned the parser program is implemented using python 3.6 language. In this section, I will discuss how I implemented the parser program and issues I faced along the way. The parser program uses a top-down approach. On the first attempt, I used a bottom-up approach but I was having some difficulty so I switched to a top-down approach which proved more successful for me. But first, we needed to take the given sentence and tag each element to its lexical value. Once the program has the tagged elements it then checks if the sentence is valid using basic language rules. The tagging and checking are discussed more in-depth in the next section. Once these two functions are completed and valid the program can begin parsing.

We can assume that our first rule was the sentence rule with parent representation as S and children noun phrase and verb phrase represented as NP and VP. So the system adds the S element to a list. A new function named level one is called to now parse through NP children and VP children.

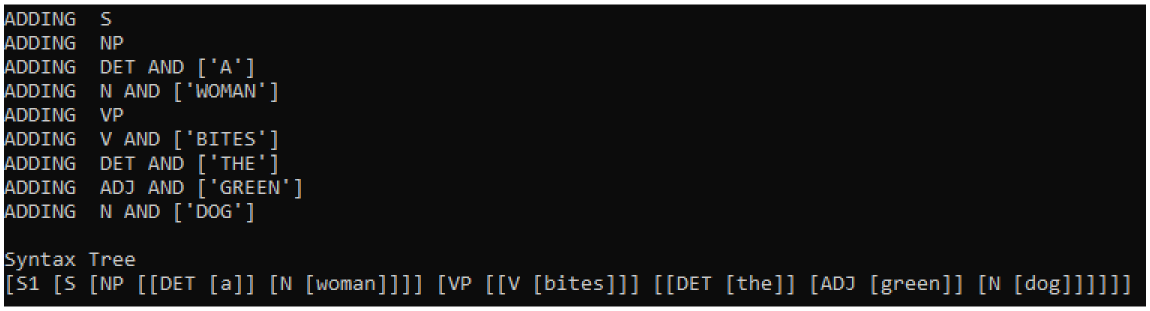
Once the function level one is initialized it adds NP and VP to a new list. We then look back at our rules CSV file to see which rows contain either NP or VP as parents to collect their children elements. Each child element and the tagged element is looped through so we can look at a comparison between the tagged element and the child element we gathered from our ruleset. This comparison is done using various if statements. The if statement checks the first and second elements of the tagged elements to see if they match one of NP or VP children see figure 1.



```
# COMPARES FROM TAGGED ELEMENTS TO
# CHILD TAKING FROM LOOPING THROUGH NP AND VP
# CHECKS FOR NEXT ELEMENT TOO
if  leveltwo_splitted[x] == newlex[loop] and leveltwo_splitted[x+1] == newlex[loop+1]:
```

Figure 1: If Statement

The comparison of the elements was a cumbersome challenge for me. I used a test file to constantly try new ways for comparing these elements to gather the correct results. I feel as if the use of the if statements used are not as efficient as if implemented a check for when looping through each child element I checked to valency so I would know how many elements would need to be compared for each child element. If the elements match these will be added to a new list. The purpose of these lists is to output the values in syntactical structure each level of the structure will contain a list these lists are then contained within one another see fig one to see the output of the parsing program 2.



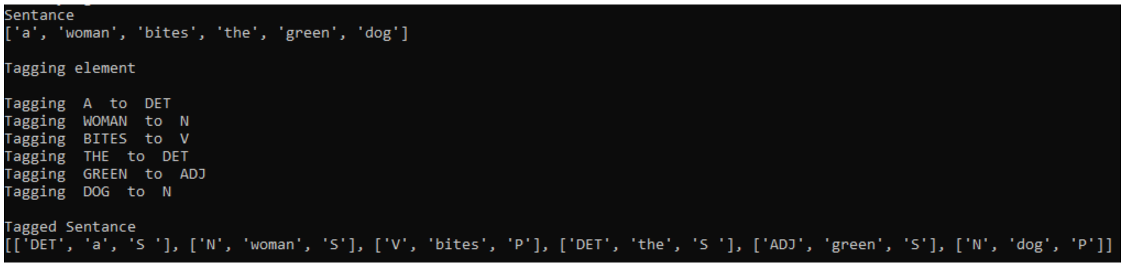
```
ADDING  S
ADDING  NP
ADDING  DET AND ['A']
ADDING  N AND ['WOMAN']
ADDING  VP
ADDING  V AND ['BITES']
ADDING  DET AND ['THE']
ADDING  ADJ AND ['GREEN']
ADDING  N AND ['DOG']

Syntax Tree
[S1 [S [NP [[DET [a]] [N [woman]]]] [VP [[V [bites]]] [[DET [the]] [ADJ [green]] [N [dog]]]]]]
```

Figure 2: Parsing

Instead of using a nested list structure I first used a binary tree to store each parent and its children. I ran into the issue of only being able to store a maximum of two child elements for each parent this caused an issue when trying to apply my last NP rule see table 2.

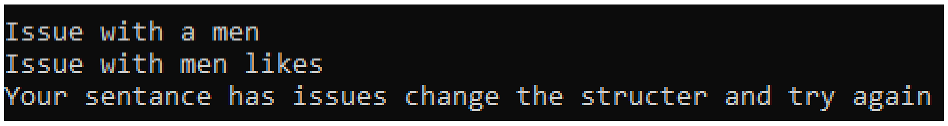
4 System architecture

The system contains multiple functions that call one another. Firstly the program reads in each CSV file and takes a string sentence. When the program is run a function called begin parsing initializes this function takes a string value of the sentence. The function begin parsing first step is to call a function called tag elements. This function takes the sentence and splits each word it then checks our lexicon CSV file to see if the word appears if it does it will then be added to a list containing the word, lexical value and if its a plural or noun see table 1 for reference and see fig 3 for program output. Once these elements are tagged the begin parsing calls a function called checker this checks the sentence structure by taking the tagged sentence and checks if a plural comes after a plural or if a sentence contains the word 'A' followed by a plural see fig 4 for the output of the checker program.



```
Sentance
['a', 'woman', 'bites', 'the', 'green', 'dog']

Tagging element

Tagging  A  to  DET
Tagging  WOMAN  to  N
Tagging  BITES  to  V
Tagging  THE  to  DET
Tagging  GREEN  to  ADJ
Tagging  DOG  to  N

Tagged Sentance
[['DET', 'a', 'S '], ['N', 'woman', 'S'], ['V', 'bites', 'P'], ['DET', 'the', 'S '], ['ADJ', 'green', 'S'], ['N', 'dog', 'P']]
```

Figure 3: Tagging



```
Issue with a men
Issue with men likes
Your sentance has issues change the structer and try again
```

Figure 4: Checker

The checker function will then return true or false to begin parsing function. If true it means the sentence is valid and the parsing can commence if not it means the program will stop. Once the sentence is valid we can then parse the sentence and output it in a syntactical structure. See fig 5 for the system architecture diagram. I used the python library time to have a delay time to different outputs so the person observing the work can interpret the process in a more efficient manner. Screenshots of the output can be seen on the next page.

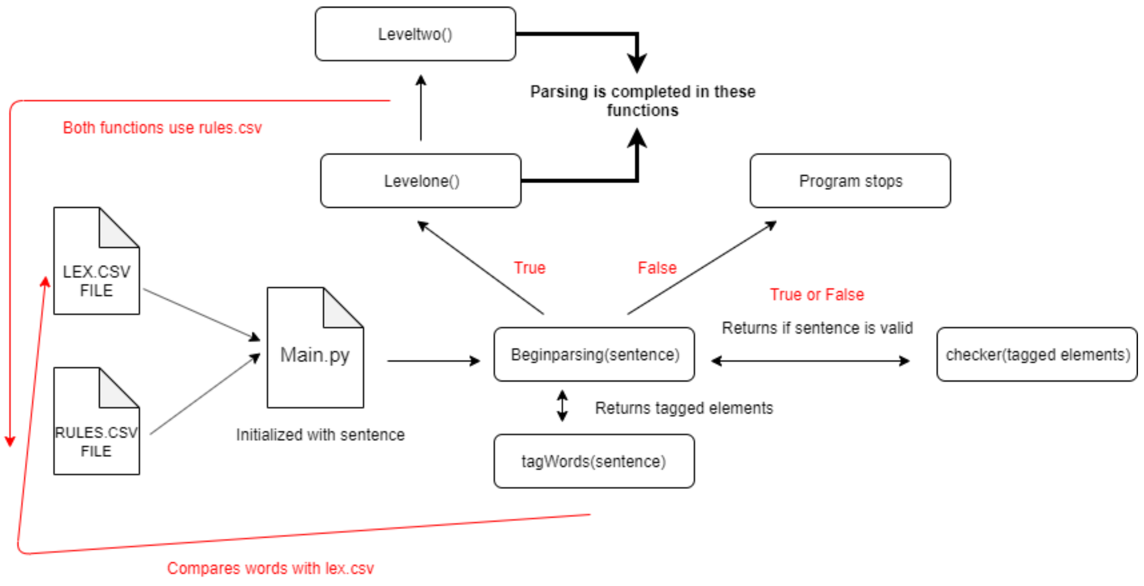
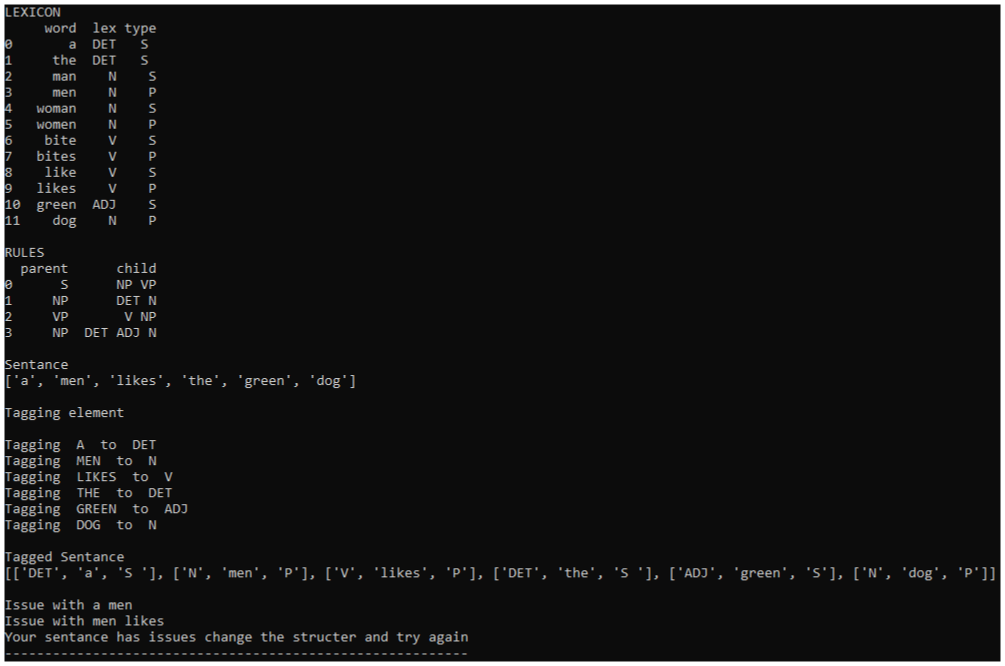


Figure 5: System Diagram

5 Screenshots

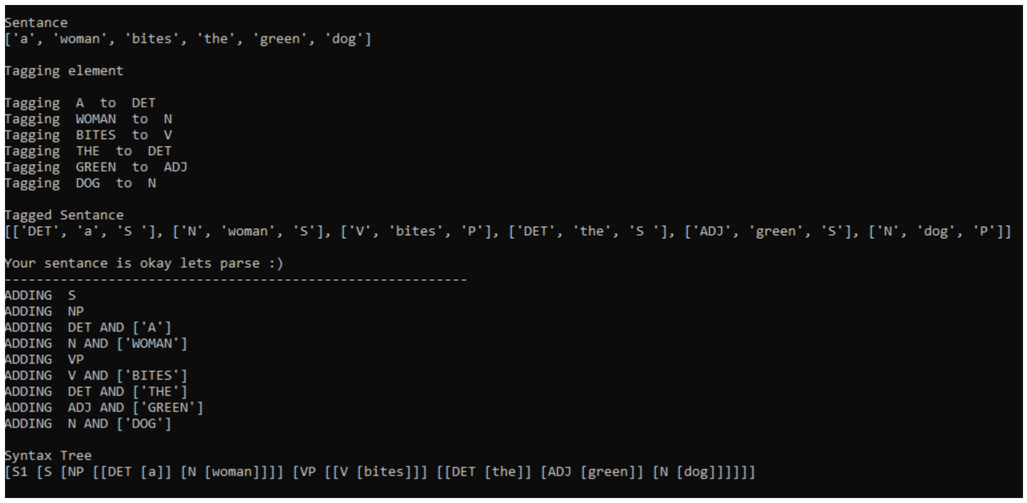
This is the output of an invalid sentence.



```
LEXICON
     word  lex type
0       a  DET   S
1     the  DET   S
2     man    N    S
3     men    N    P
4   woman    N    S
5   women    N    P
6    bite    V    S
7   bites    V    P
8    like    V    S
9   likes    V    P
10  green  ADJ    S
11    dog    N    P

RULES
  parent      child
0      S      NP VP
1     NP      DET N
2     VP       V NP
3     NP  DET ADJ N

Sentance
['a', 'men', 'likes', 'the', 'green', 'dog']

Tagging element

Tagging  A  to  DET
Tagging  MEN  to  N
Tagging  LIKES  to  V
Tagging  THE  to  DET
Tagging  GREEN  to  ADJ
Tagging  DOG  to  N

Tagged Sentance
[['DET', 'a', 'S '], ['N', 'men', 'P'], ['V', 'likes', 'P'], ['DET', 'the', 'S '], ['ADJ', 'green', 'S'], ['N', 'dog', 'P']]

Issue with a men
Issue with men likes
Your sentance has issues change the structer and try again
----------------------------------------------------------
```

Figure 6: Invalid sentence

This is the output of a valid sentence.



```
Sentance
['a', 'woman', 'bites', 'the', 'green', 'dog']

Tagging element

Tagging  A  to  DET
Tagging  WOMAN  to  N
Tagging  BITES  to  V
Tagging  THE  to  DET
Tagging  GREEN  to  ADJ
Tagging  DOG  to  N

Tagged Sentance
[['DET', 'a', 'S '], ['N', 'woman', 'S'], ['V', 'bites', 'P'], ['DET', 'the', 'S '], ['ADJ', 'green', 'S'], ['N', 'dog', 'P']]

Your sentance is okay lets parse :)
----------------------------------------------------------
ADDING  S
ADDING  NP
ADDING  DET AND ['A']
ADDING  N AND ['WOMAN']
ADDING  VP
ADDING  V AND ['BITES']
ADDING  DET AND ['THE']
ADDING  ADJ AND ['GREEN']
ADDING  N AND ['DOG']

Syntax Tree
[S1 [S [NP [[DET [a]] [N [woman]]]] [VP [[V [bites]]] [[DET [the]] [ADJ [green]] [N [dog]]]]]]
```

Figure 7: valid sentence