Technological University Dublin

Software Parser

By

Dylan Hallissey

B00089794

Introduction

The assignment asks to describe, design and implement in software a parser for Regular Expressions using the programming language of your choice. The parser should take input and then analyse a set of Regular Expressions, the selection we got to do was:

The/a boy/boys/girl/girls bite(s)/like(s) the white cat.

The input sentence should be parsed to be accepted or rejected as a valid Regular Expression. The output should be bracketed phrasal structure for the sentence for as far as it is grammatical. It should also output an appropriate parse tree displaying the sentence structure similar to below but suited to your inputted regular expression.

I choose this program to work on python because I am more knowledgeable when it comes to using python rather than java.

lexical

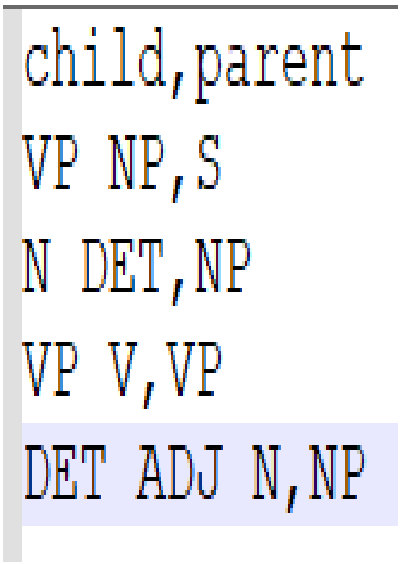
to create a lexical categories I first made a text file which it did not work, I researched more and realised that having the lexical file as a CSV is better because its easier for python, but I can write this using any ext editor and then while saving the name write it as .csv. We use this text file to add verbs, nouns etc because it will appear when a certain word appears. The lexical file is layout with 3 headings. The firs one word, lexical then type. The word will be all the words that we are set out to complete in the project which be The/a boy/boys/girl/girls bite(s)/like(s) the white cat.

Then lexical be category for all the nouns then the type will be if the word is plural or singular. We have the type plural and singular because singular noun refers to only one person or thing and a plural noun refers to more than one person or thing.

```
word,lexicon,type
the,DET, singular
a,DET, singular
boy,N, singular
boys,N,plural
girl,N, singular
girls,N, plural
like,V, singular
likes,V, plural
bite,V, singular
bites,V, plural
white ,ADJ, singular
cat,N, singular
```

There was a second CSV file was used which was for the rules. The rules that the possible sentence must go by. The rules_Syntax CSV file is broken into two a child and parent. The parent holds noun phrase and the child column has the values determiner. The child column has values to create a parent, this is important for building our project.

Layout of rules_Syntax:



Parser program

The parser program implemented by the python. In this section. I will explain the parser program and issues I had while tackling this task. The program uses

“words = "A boy bites white cat".lower().split()”

To return and split the string into a list. meaning we now have our sentence we just need to program to have it print out as tree and see is the sentence valid. Then we create a sentence tag and for plurals, we also create a tree and a root tree that will have the child create a parent which be a root tree. Then after we created that we will then make a code to split the child and parent rules:

```
while(x < len(ruleschild)):
    rulesparents[0].append(ruleschild[x].split())
    x+=1
```

That is the main now completed now we focus on the main parts to complete this task. We move on to the def POS_rulest his will make the program grows the lists to the input words.

We then program the lexicon to connect to the CSV file to have the lexicon rule to are program. Therefore the parser software should create whats on the lexicon rules csv which be words, lexicon and type. We then should be able to print out the rule for our code. The code be like this printed:

```
print("_____*Rules, Words, Lexicons and Types*_____")
print("RULES: " + str(rulesparents))
print("")
print("WORDS : "+ str(words))
print("")
print("LEXICONS : "+ str(list))
print("")
print("TYPES  : "+ str(type))
```

Then move on to the List_rule(child, syntaxtree). This is when we will use the list to pick each child in the tree to match the word. The child should match the word to the parent.

We use this code by using:

```
if (child[l] == "NP") and (child[l+1] == "VP"):
        for x in range(2):
            roottree[syntaxtree].append(collection)
            l+=1
```

we are trying to find the collection of childs that will create parent. In this case the two child be Np and VP and cause we using two child we use the “for x in range(2):” cause we used two childs and we use the l+=1 to add its current value to its self. That’s what code is used for this part.

The next part was to get the valid or non valid which didn’t work I used many methods to achieve this but ran out of time trying to find the solution. It was supposed to see if word was valid it would of said it was accepted or not.

We now move on to the def Treesystem(): which be printing out the tree for the parser, itshould have the nouns going down a level and have the words at the very bottom with the nouns above them. We use a len() function in this method cause it returns the number of items in an object and then we use str(tree[l]) to print out the tree. We also did this for the root tree to find the index of the parser. Now we should have the lexicon trees printed out to the console.

The def syntaxbracket(): be the last method for this project, I struggled allot to get this working even without have a valid program in my project, it failed because the nouns in the bracket only appear in some of the words. We use the syntaxmatrices = [] to put the words in the bracket.

Conclusion

The conclusion the csv of the rule came up perfect for the project, it printed out console each type of rule.

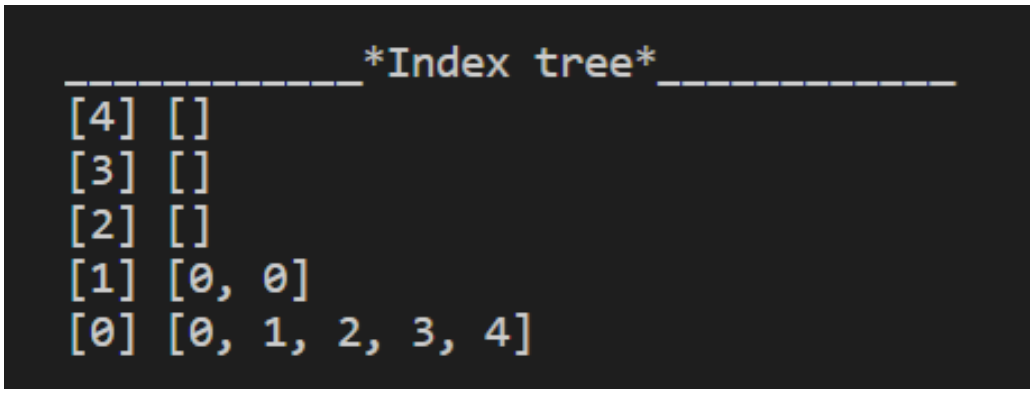
```
__________*Rules, Words, Lexicons and Types*__________
RULES: [[['NP', 'VP'], ['DET', 'ADJ', 'N'], ['DET', 'N'], ['V', 'NP']], ['S', 'NP', 'NP', 'VP']]

WORDS : ['a', 'boy', 'bites', 'white', 'cat']

LEXICONS : ['DET', 'N', 'V', 'N']

TYPES  : [' singular', ' singular', ' plural', ' singular']
```

Then because next should print out consoled be valid but couldn't get it working therefore the tree will print out next to the console.

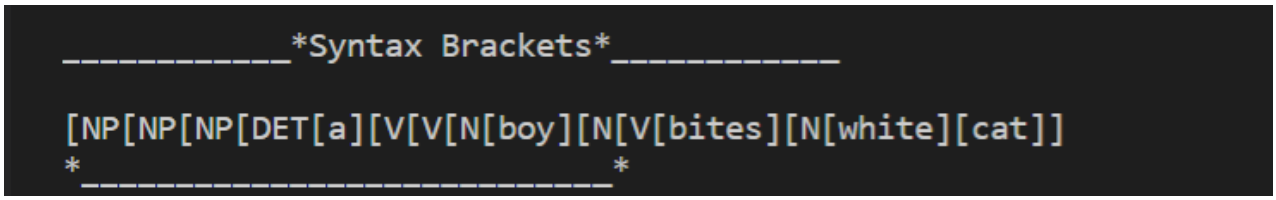
```
___________*Lexicon tree*___________
[4] ['NP', 'V', 'N']
[3] ['NP', 'V', 'N']
[2] ['NP', 'V', 'N']
[1] ['DET', 'N', 'V', 'N']
[0] ['a', 'boy', 'bites', 'white', 'cat']
```

This failed because at start I got S at top row the going down, it did work but when I added more code it didn't fully print out the correct tree.

I made root tree which be index of the tree but that also didn’t work sadly.



The last part be the bracket but because other tree didn’t work the bracket for sentence is also not fully correct.



Screenshots

Main:

```
rulesSyntax = p.read_csv('Rules_Syntax.csv', delimiter=",")
lexiconRules = p.read_csv('Rules_Lexicon.csv', delimiter=",")
words = "A boy bites white cat".lower().split()
ruleschild = rulesSyntax["child"].tolist()
rulesparents = [ [] , rulesSyntax["parent"].tolist()]
#words = "A boy likes the white cat".lower().split()
# words = "A girl bites the white cat".lower().split()
# words = "A girl likes the white cat".lower().split()
# words = "The boy bites the white cat".lower().split()
# words = "The boy likes the white cat".lower().split()
# words = "The girl bites the white cat".lower().split()
# words = "The girl likes the white cat".lower().split()
# words = "The girls bite the white cat".lower().split()
# words = "The girls like the white cat".lower().split()
# words = "The boys bite the white cat".lower().split()
# words = "The boys like the white cat".lower().split()



list = []
type = []
tree = [[],[],[],[],[]]
roottree = [[],[],[],[],[]]
x=0
while(x < len(ruleschild)):
    rulesparents[0].append(ruleschild[x].split())
    x+=1 ##split the childs and parent rules
```

POS_Rules:

```
def POS_rules(): # grows the lists to  input sentence thats on the rules.
    for word in words: # For each line in sentence
        filtered = lexiconRules[lexiconRules.word == word]
        for row in filtered[['lexicon']].values:
            list.append(row[0])
        for row in filtered[['type']].values:
            type.append(row[0])

    tree[0] = words
    tree[1] = list
    for x in range(len(words)): # Populate the first row of the treeParents with values
        roottree[0].append(x)

    print("___________*Rules, Words, Lexicons and Types*___________")
    print("RULES: " + str(rulesparents))
    print("")
    print("WORDS : "+ str(words))
    print("")
    print("LEXICONS : "+ str(list))
    print("")
    print("TYPES  : "+ str(type))
    print("")
    time.sleep(4)
```

List_rule:

```
def List_rule(child, syntaxtree): # will use the list to pick each child in the tree to match the word
    collection = 0
    l = 0
    while(l < len(child)): # each child


        if (child[l] == "NP") and (child[l+1] == "VP"):
            tree[syntaxtree+1].append("S")
            for x in range(2):
                roottree[syntaxtree].append(collection)
                l+=1
        elif (child[l] == "V") and (child[l+1] == "NP"):
            tree[syntaxtree+1].append("VP")
            for x in range(2):
                roottree[syntaxtree].append(collection)
                l+=1
        elif (child[l] == "DET") and (child[l+1] == "N"):
            tree[syntaxtree+1].append("NP")
            for x in range(2):
                roottree[syntaxtree].append(collection)
                l+=1
        elif (child[l] == "DET") and (child[l+1] == "ADJ") and (child[l+2] == "N"):
            tree[syntaxtree+1].append("NP")
            for x in range(3): # Since this rule uses 3 childs mean move on next one
                roottree[syntaxtree].append(collection)
                l+=1

        else:
            tree[syntaxtree+1].append(child[l]) # If none rule appear we move on
            l+=1
```

TreeSystem:

```
def Treesystem():
    time.sleep(5)
    print("")
    print("")
    print("___________*Lexicon tree*___________")
    l = len(tree)-1
    while l >= 0:
        print("["+str(l)+"] "+str(tree[l]))
        l=l-1
    print("")
    print("")
    time.sleep(3)
    print("___________*Index tree*___________")
    l = len(tree)-1
    while l >= 0:
        print("["+str(l)+"] "+str(roottree[l]))
        l=l-1
```

Syntaxbracket:

```
def syntaxbracket():

    print("")
    print("")
    time.sleep(4)
    print("___________*Syntax Brackets*___________")
    print("")

    syntaxmatrices = []
    childsize = 3
    treesize = 4
    l = 0
    while(treesize >= 0):
        if(treesize >= 0 or tree[treesize][l] != treesize[treesize-1][l]):
            print("[", end=""),
            print(tree[treesize][l], end=""),
            syntaxmatrices.append(tree[treesize][l])
            treesize = treesize-1
            if treesize < 0:
                print("]", end=""),
                treesize = childsize
                childsize  = childsize -1
                l = l+1
    print("]")


    print("*_____________________________*")
```