CE314/887 – NATURAL LANGUAGE ENGINEERING

ASSIGNMENT 2: PARSING AND WORD SIMILARITY

Submitted by

ISHWAR VENUGOPAL

(REGISTRATION NUMBER: 1906084)

MSc Artificial Intelligence

(Assignment done in pairs with Shreya Jadhav [1702121])

December 2019

Part 1: Extending a Grammar

Use the following grammar as starting point for parsing sentences S1 to S3 with NLTK, extending the grammar as needed according to the grammatical rules of English, as discussed in the textbook. Use the Chart parser available in NLTK.

S -> NP VP
NP -> Det Nom | PropN | NP PP
Nom -> Adj Nom | N
VP -> V NP | V S | VP PP
PP -> P NP
PropN -> ’Bill’
Det -> ’the’ | ’a’ | an
N -> ’bear’ | ’squirrel’ | ’park’
Adj -> ’angry’ | ’frightened’
V -> ’chased’ | ’saw’ | ’put’ | ’eats’ | ’eat’
P -> ’on’

S1. Put the block on the table

S2. Bob chased a bear in the park along the river

S3. Bill saw Bob chase the angry furry dog

a) Which rules do you need to add to the grammar to parse S1 to S3?

Solution:

Explanation:

To parse the above given sentences we, need to add the following rules:

S -> VP NP
VP-> V
PropN -> ‘Bob’
Adj-> ‘furry’
P-> ‘in’ | ‘along’
V-> ‘chase’
N -> ‘river’| ‘dog’ | ‘block’ | ‘table’

The updated grammar will look like this:

S -> NP VP | **VP NP**
PP -> P NP
Nom -> Adj Nom | N
NP -> Det Nom | PropN | NP PP
VP -> V NP | V S | VP PP | **V**
Det -> "the" | "a" | "an"

PropN -> "Bill" | **"Bob"**
Adj -> "angry" | "frightened" | **"furry"**
N -> "bear" | "squirrel" | "park" | **"river" | "dog" | "block" | "table"**
V -> "chased" | "saw" | "Put" | "eats" | "eat" |**"chase"**
P -> "on" | **"in" | "along"**

b) How many derivations can you get for each sentence?

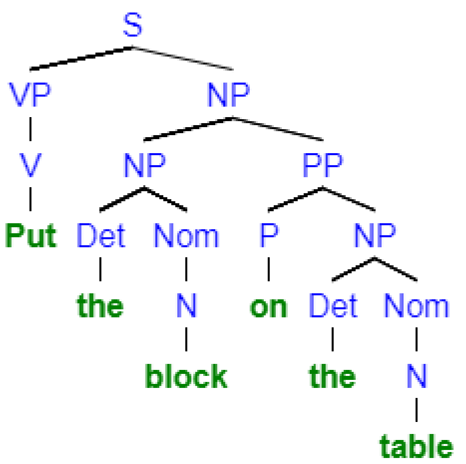
Solution:

We were able to obtain 1 derivation for S1, 5 derivations for S2 and 1 derivation for S3.

Sample Output: (The sentence trees for the outputs are obtained from http://mshang.ca/syntree/ [1])

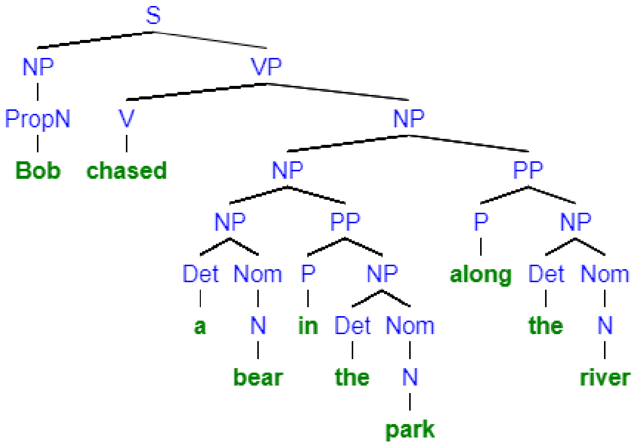
Sentence 1:

```
(S
  (VP (V Put))
  (NP
    (NP (Det the) (Nom (N block)))
    (PP (P on) (NP (Det the) (Nom (N table))))))
```

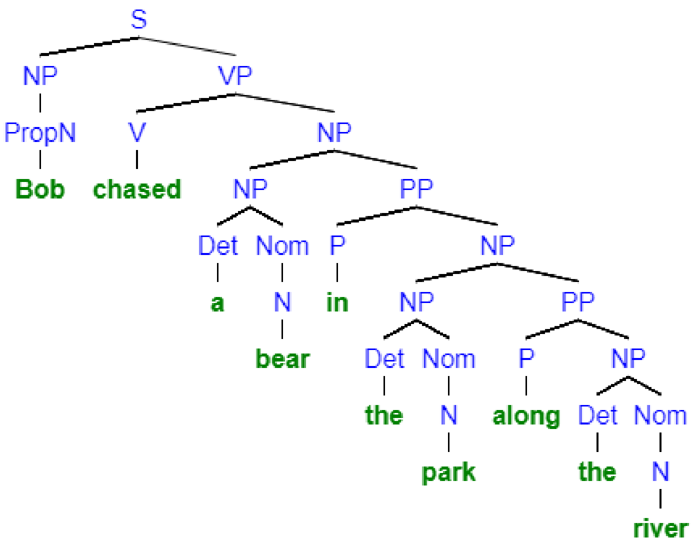


Sentence 2:

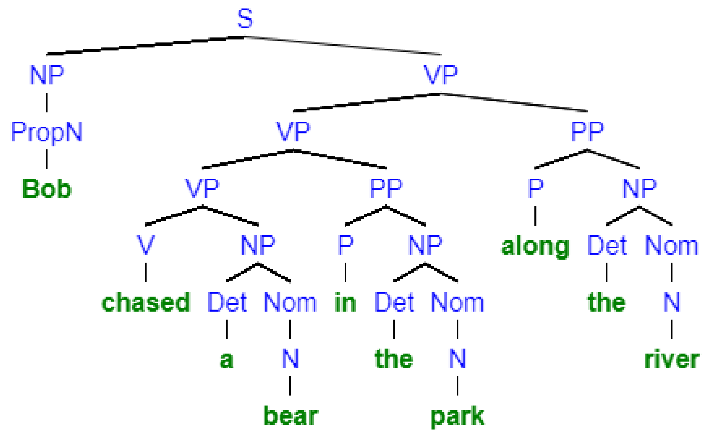
```
Sentence 2
(S
  (NP (PropN Bob))
  (VP
    (V chased)
    (NP
      (NP
        (NP (Det a) (Nom (N bear)))
        (PP (P in) (NP (Det the) (Nom (N park)))))
      (PP (P along) (NP (Det the) (Nom (N river)))))))
```



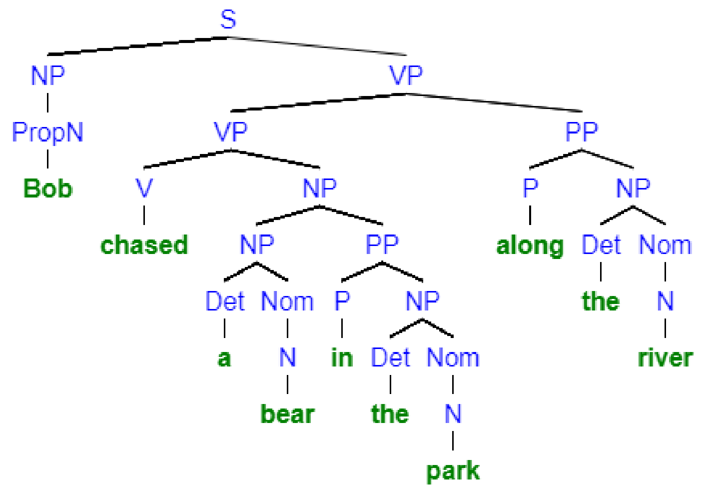

```
(S
  (NP (PropN Bob))
  (VP
    (V chased)
    (NP
      (NP (Det a) (Nom (N bear)))
      (PP
        (P in)
        (NP
          (NP (Det the) (Nom (N park)))
          (PP (P along) (NP (Det the) (Nom (N river))))))))
```



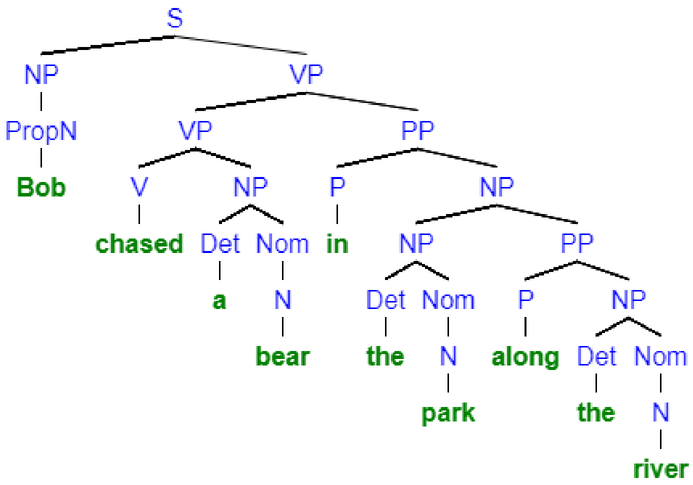

```
(S
  (NP (PropN Bob))
  (VP
    (VP
      (VP (V chased) (NP (Det a) (Nom (N bear))))
      (PP (P in) (NP (Det the) (Nom (N park)))))
    (PP (P along) (NP (Det the) (Nom (N river))))))
```




```
(S
  (NP (PropN Bob))
  (VP
    (VP
      (V chased)
      (NP
        (NP (Det a) (Nom (N bear)))
        (PP (P in) (NP (Det the) (Nom (N park))))))
    (PP (P along) (NP (Det the) (Nom (N river))))))
```

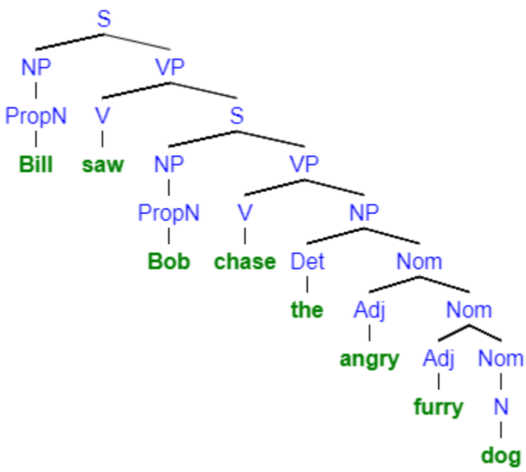



```
(S
  (NP (PropN Bob))
  (VP
    (VP (V chased) (NP (Det a) (Nom (N bear))))
    (PP
      (P in)
      (NP
        (NP (Det the) (Nom (N park)))
        (PP (P along) (NP (Det the) (Nom (N river))))))))
```



Sentence 3:

```
(S
  (NP (PropN Bill))
  (VP
    (V saw)
    (S
      (NP (PropN Bob))
      (VP
        (V chase)
        (NP
          (Det the)
          (Nom (Adj angry) (Nom (Adj furry) (Nom (N dog)))))))))
```



Q2. Parsing Quality

Consider the sentences:

S4. An bear eat an squirrel

S5. The dogs eats

a) Are these sentences correct? What are the grammatically correct equivalents for these sentences?

Solution:

No, these sentences are not grammatically correct.

In S4, 'An' cannot be grammatically used (while considering standard English grammar), because the word following 'An' must start with a vowel sound [2]. Both 'bear' and 'squirrel' do not start with a vowel sound. Also, 'eat' should have been in the third person singular form.

Grammatically correct equivalent: A bear eats a squirrel

In S5, 'eats' is a third person singular verb. Therefore it cannot follow 'dogs' as the word 'dogs' does not satisfy the conditions for a singular subject in third person [3].

Grammatically correct equivalent: The dogs eat

b) Run 2 parsers from NLTK on the two sentences. What is the output of the parsers? (Copy and paste only these sentences and their derivations). Explain why the parsers are correct or incorrect.

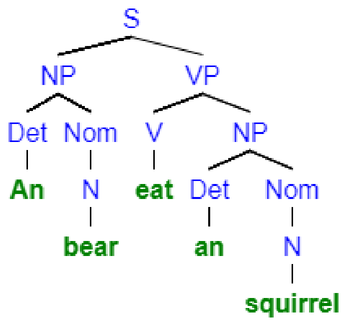
Solution:

We ran both the sentences using the Chart Parser and the Recursive Descent Parser. The outputs are as follows:

Sample output for Chart Parser:

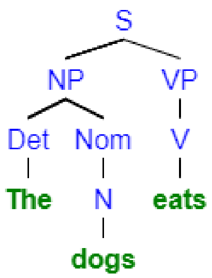
S4. An bear eat an squirrel

```
(S
  (NP (Det An) (Nom (N bear)))
  (VP (V eat) (NP (Det an) (Nom (N squirrel)))))
```



S5. The dogs eats

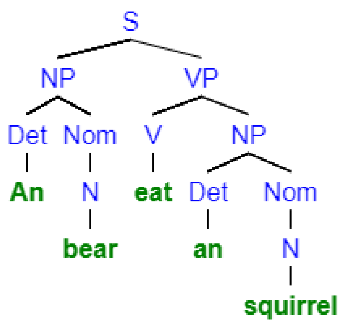
```
(S (NP (Det The) (Nom (N dogs))) (VP (V eats)))
```



<u>Sample output for Recursive Descent parser:</u>

S4. An bear eat an squirrel

```
(S
  (NP (Det An) (Nom (N bear)))
  (VP (V eat) (NP (Det an) (Nom (N squirrel)))))
```



S5. The dogs eats

```
RecursionError: maximum recursion depth exceeded while calling a Python object
```

<u>Explanation:</u>

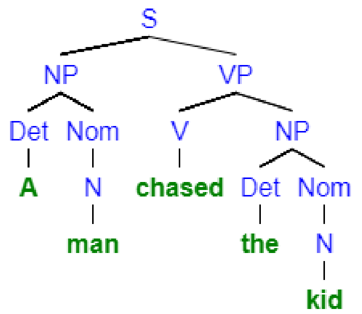
Within the given grammar, the parsers are able to allocate the suitable sentence tree to each of these sentences. Even though these sentences may seem incorrect to us in terms of the Standard English grammar, the proper trees for them given by the program is justified by the fact that these sentences are not ungrammatical as per the grammar defined by us. Hence, it produces proper trees. But in case of Recursive Descent (RD) Parser, it can be seen that instead of returning a parse tree it returns an error message saying that the maximum recursion depth has exceeded. This can be due to the fact that RD Parser is a top-down technique that recursively parses the input to make a parse tree [4]. This property of recursion leads this to an infinite loop while trying to parse a tree for S5.

c) Generate 2 other correct and 2 other incorrect sentences with this grammar. How would you have to change this grammar to prevent these sentences from being parsed? You can write your own rules to extend the grammar and ensure correct agreement.

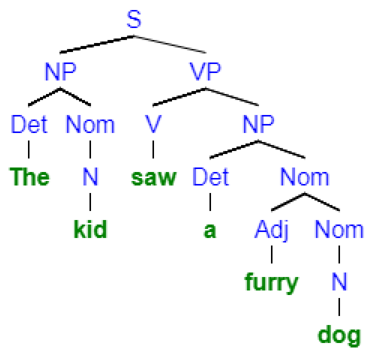
<u>Solution:</u>

Two sentences that are correct according to this grammar and Standard English grammar:

1. An man chased the kid.

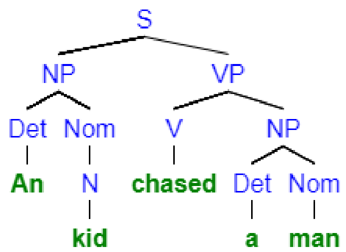


2. The kid saw a furry dog.

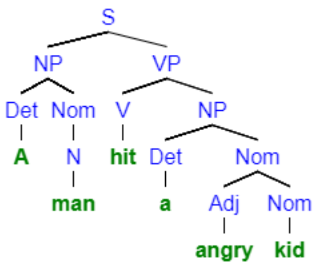


Two incorrect sentences which are correct according to this grammar but not by Standard English grammar:

1. An kid chased a man.



2. A man hit a angry kid.



To prevent these sentences from being parsed, the grammar needs to be changed to be able to separately identify the determiners ‘a’ and ‘an’. And also it should be able to identify separately words that begin with a consonant sound and those which begin with a vowel sound. It can be done as follows for example:

ConsN -> “man” | “kid”
VowN -> “entity” | “omlette”
ConsAdj -> “blue” | “green”
VowAdj -> “angry”
AnDet -> “An”
ADet -> “A”

Q3. Parsing Ambiguity

S6. He eats pasta with some anchovies in the restaurant

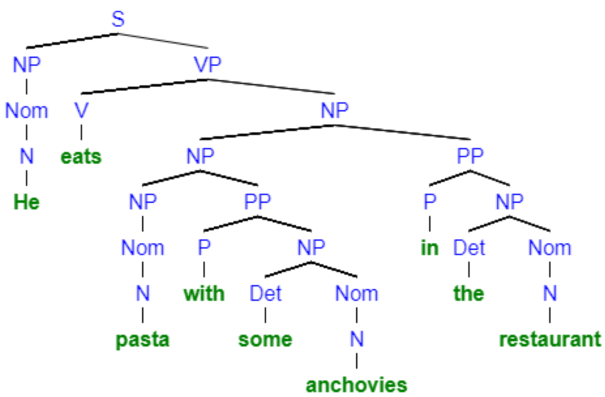
S7. He eats pasta with a fork in the restaurant

a)Do S6 and S7 have more than one interpretation? If so, draw all derivations and briefly describe each of the interpretations.

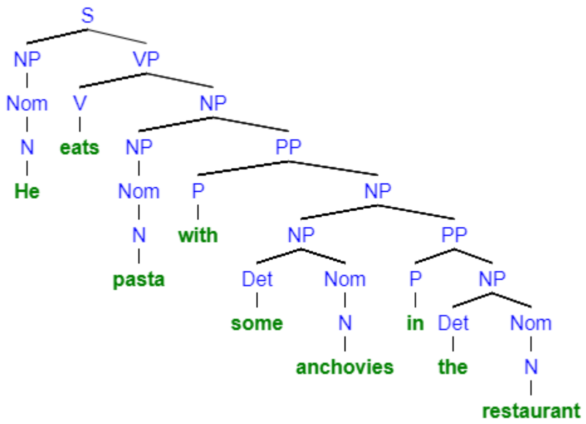
Solution:

Yes, these sentences have more than one interpretation.

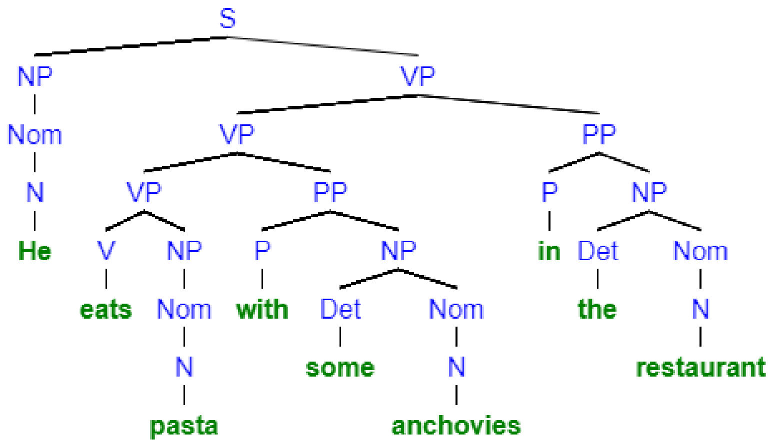
S6. He eats pasta with some anchovies in the restaurant



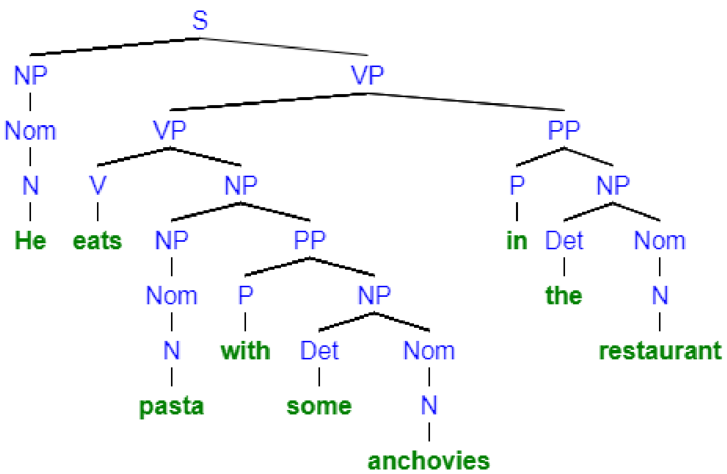
This interpretation treats [pasta with some anchovies] as a single noun phrase.



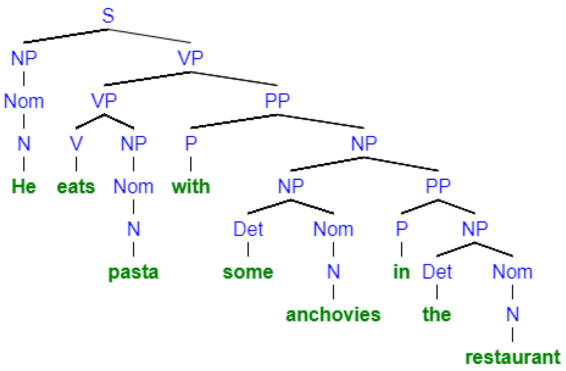
Here [some anchovies in the restaurant] is a single noun phrase.



Here more importance is given to the verb ‘eats’ and the rest of the sentence excluding ‘He’ is treated as a single verb phrase.

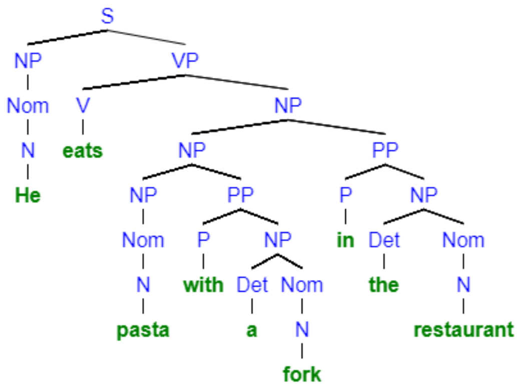


A similar approach to the previous interpretation, but we have a separate noun phrase and preposition phrase within the main verb phrase.

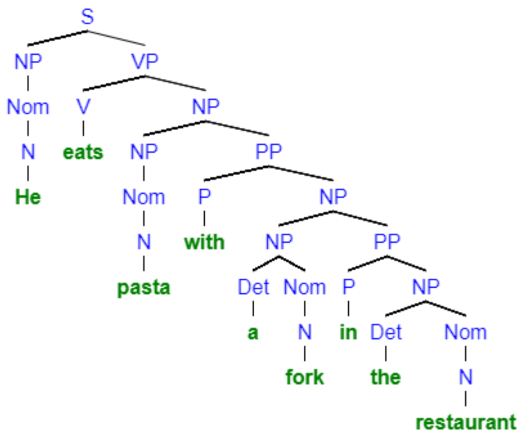


Here, [eats pasta] is given as a sub verb phrase and everything other than that in the main VP is treated as a single preposition phrase.

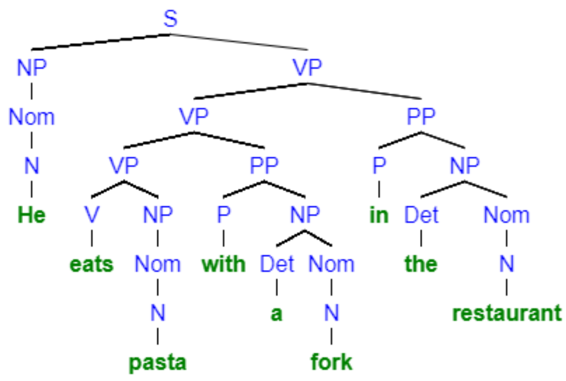
<u>S7. He eats pasta with a fork in the restaurant</u>



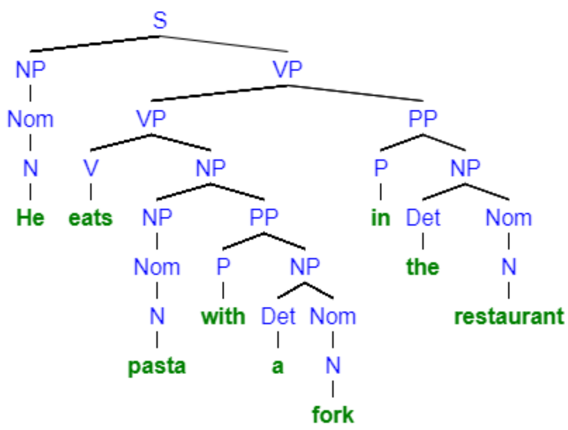
[in the restaurant] is treated as a single preposition phrase.



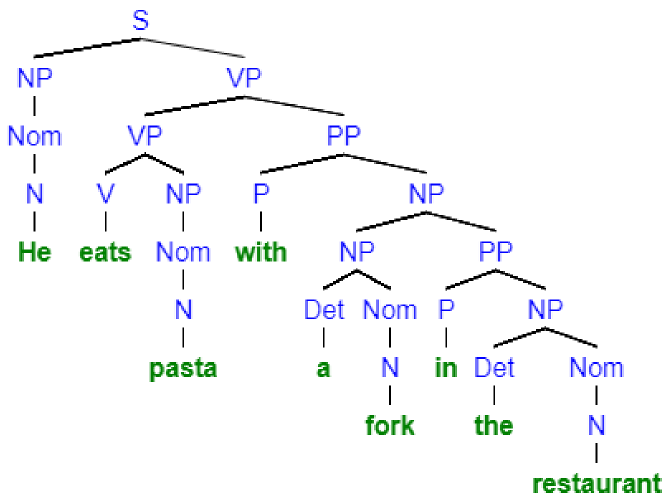
[a fork in the restaurant] is treated as a single noun phrase.



[eats pasta], [with a fork] and [in the restaurant] are treated separately here.



Here [pasta with a fork] and [in the restaurant] are treated separately.



Here ‘with’ is used as a connecting word between the surrounding noun phrase and verb phrase.

b) Run the Shift Reduce Parser and the Earley Chart Parser from NLTK on these sentences. Which of the parsers detects the ambiguity for S6 and S7? Copy each interpretation generated by each parser

<u>Solution:</u>

The Earley Parser gives the following outputs:

<u>S6. He eats pasta with some anchovies in the restaurant</u>

```
(S
  (NP (Nom (N He)))
  (VP
    (V eats)
    (NP
      (NP
        (NP (Nom (N pasta)))
        (PP (P with) (NP (Det some) (Nom (N anchovies)))))
      (PP (P in) (NP (Det the) (Nom (N restaurant)))))))

(S
  (NP (Nom (N He)))
  (VP
    (V eats)
    (NP
      (NP (Nom (N pasta)))
      (PP
        (P with)
        (NP
          (NP (Det some) (Nom (N anchovies)))
          (PP (P in) (NP (Det the) (Nom (N restaurant)))))))))

(S
  (NP (Nom (N He)))
  (VP
    (VP
      (VP (V eats) (NP (Nom (N pasta))))
      (PP (P with) (NP (Det some) (Nom (N anchovies)))))
    (PP (P in) (NP (Det the) (Nom (N restaurant))))))

(S
  (NP (Nom (N He)))
  (VP
    (VP
      (V eats)
      (NP
        (NP (Nom (N pasta)))
        (PP (P with) (NP (Det some) (Nom (N anchovies))))))
    (PP (P in) (NP (Det the) (Nom (N restaurant))))))
```

```
(S
  (NP (Nom (N He)))
  (VP
    (VP (V eats) (NP (Nom (N pasta))))
    (PP
      (P with)
      (NP
        (NP (Det some) (Nom (N anchovies)))
        (PP (P in) (NP (Det the) (Nom (N restaurant))))))))
```

S7. He eats pasta with a fork in the restaurant

```
(S
  (NP (Nom (N He)))
  (VP
    (V eats)
    (NP
      (NP
        (NP (Nom (N pasta)))
        (PP (P with) (NP (Det a) (Nom (N fork)))))
      (PP (P in) (NP (Det the) (Nom (N restaurant)))))))
```

```
(S
  (NP (Nom (N He)))
  (VP
    (V eats)
    (NP
      (NP (Nom (N pasta)))
      (PP
        (P with)
        (NP
          (NP (Det a) (Nom (N fork)))
          (PP (P in) (NP (Det the) (Nom (N restaurant)))))))))
```

```
(S
  (NP (Nom (N He)))
  (VP
    (VP
      (VP (V eats) (NP (Nom (N pasta))))
      (PP (P with) (NP (Det a) (Nom (N fork)))))
    (PP (P in) (NP (Det the) (Nom (N restaurant))))))
```

```
(S
  (NP (Nom (N He)))
  (VP
    (VP
      (V eats)
      (NP
        (NP (Nom (N pasta)))
        (PP (P with) (NP (Det a) (Nom (N fork))))))
    (PP (P in) (NP (Det the) (Nom (N restaurant))))))
```

```
(S
  (NP (Nom (N He)))
  (VP
    (VP (V eats) (NP (Nom (N pasta))))
    (PP
      (P with)
      (NP
        (NP (Det a) (Nom (N fork)))
        (PP (P in) (NP (Det the) (Nom (N restaurant))))))))
```

The Shift Parsers doesn't give any results when used in the program. It results in empty lists rather than generating a parsed tree. The code snippet and the corresponding output is as follows:

```
parsersr= nltk.parse.ShiftReduceParser(grammar)
print('Sentence 6 \n')
for tree in parsersr.parse(s6):
    print(tree)
print('\n')

print('Sentence 7 \n')
for tree in parsersr.parse(s7):
    print(tree)
```

```
Warning: VP -> V NP will never be used
Warning: VP -> V S will never be used
Sentence 6



Sentence 7
```

Q4. Calculating similarity between words

Task1: Build a program to calculate word similarity in BioSim-100.txt using WordNet. For each word pair in BioSim-100.txt you will calculate the WordNet similarity between the pair, using the path similarity function implemented in NLTK, and print this into a file, along with the gold standard similarity.

Solution:

Sample Output:

Note: The data provided is 'SimLex999-100.txt' and not the one mentioned in the question ('BioSim-100.txt)

```
old     new     1.58    None
smart   intelligent     9.2     None
hard    difficult       8.77    1.0
happy   cheerful        9.55    None
hard    easy    0.95    None
fast    rapid   8.75    0.06666666666666667
happy   glad    9.17    None
short   long    1.23    None
stupid  dumb    9.58    None
weird   strange 8.93    None
wide    narrow  1.03    None
bad     awful   8.42    None
easy    difficult       0.58    None
bad     terrible        7.78    None
hard    simple  1.38    None
smart   dumb    0.55    None
insane  crazy   9.57    None
happy   mad     0.95    None
large   huge    9.47    None
hard    tough   8.05    None
new     fresh   6.83    None
sharp   dull    0.6     None
quick   rapid   9.7     0.125
dumb    foolish 6.67    None
wonderful       terrific        8.63    None
strange odd     9.02    None
happy   angry   1.28    None
narrow  broad   1.18    0.09090909090909091
simple  easy    9.4     None
old     fresh   0.87    None
apparent        obvious 8.47    None
inexpensive     cheap   8.72    1.0
nice    generous        5       None
weird   normal  0.72    0.1111111111111111
```

The data was printed into a file called “SimLex999-100-predicted.txt”. The columns represent word_1, word_2, Gold_Similarity_Value and Path_Similarity_Value.

Task2: Build a program to detect word similarity in other texts. You will need to pre-process the user specified input text, reading the file, performing sentence splitting, tokenization and lemmatization, and removing stopwords and punctuation. The resulting file should contain only content words, one word per line. For each word in the file you will calculate the WordNet path similarity between the pair, and print this into a file. Now apply your program to the file text1.txt.

Solution:

Sample output:

```
way     way     1.0
way     sometimes       None
way     flight  0.125
way     sews    None
way     excitement      0.125
way     finish  0.14285714285714285
way     hear    None
way     dog     0.07692307692307693
way     danger  0.16666666666666666
way     place   0.1
way     concerned       None
way     adventurous     None
way     hard    None
way     adventure       0.09090909090909091
way     time    0.1111111111111111
way     on      None
way     closet  0.07692307692307693
way     black   0.16666666666666666
way     surprised       None
way     eating  0.09090909090909091
way     safe    0.07142857142857142
way     next    None
way     gone    None
way     tomato  0.07692307692307693
way     southwestern    0.07692307692307693
way     duty    0.1111111111111111
way     by      None
way     over    0.09090909090909091
way     stick   0.08333333333333333
way     knew    None
way     desperate       0.1
way     reached None
way     wood    0.1
way     finally None
way     own     None
way     cat     0.05555555555555555
```

The data is saved into a file named "similarity_counts.txt". The columns represent word_1, word_2 and the path_similarity_value.

Task3: Replace each word by its hypernym and calculate the similarities between each word pair printing this additional information to the file original-pairs-hypernyms.txt.

<u>Solution:</u>

<u>Sample Output:</u>

```
old     new     1.58    past    None    0
smart   intelligent     9.2     pain    None    0
hard    difficult       8.77    None    None    0
happy   cheerful        9.55    None    None    0
hard    easy    0.95    None    None    0
fast    rapid   8.75    abstinence      waterway        0.07692307692307693
happy   glad    9.17    None    iridaceous_plant        0
short   long    1.23    tract   desire  None
stupid  dumb    9.58    simpleton       None    0
weird   strange 8.93    anglo-saxon_deity       None    0
wide    narrow  1.03    None    strait  0
bad     awful   8.42    quality None    0
easy    difficult       0.58    None    None    0
bad     terrible        7.78    quality None    0
hard    simple  1.38    None    herb    0
smart   dumb    0.55    pain    None    0
insane  crazy   9.57    None    lunatic 0
happy   mad     0.95    None    None    0
large   huge    9.47    size    None    0
hard    tough   8.05    None    combatant       0
new     fresh   6.83    None    None    0
sharp   dull    0.6     musical_notation        change  None
quick   rapid   9.7     area    waterway        0.16666666666666666
dumb    foolish 6.67    None    None    0
wonderful       terrific        8.63    None    None    0
strange odd     9.02    None    None    0
happy   angry   1.28    None    None    0
narrow  broad   1.18    strait  woman   0.1111111111111111
simple  easy    9.4     herb    None    0
old     fresh   0.87    past    None    0
apparent        obvious 8.47    None    None    0
inexpensive     cheap   8.72    None    None    0
nice    generous        5       None    None    0
weird   normal  0.72    anglo-saxon_deity       practice        0.14285714285714285
weird   odd     9.2     anglo-saxon_deity       None    0
bad     immoral 7.62    quality None    0
sad     funny   0.95    None    joke    0
wonderful       great   8.05    None    achiever        0
guilty  ashamed 6.38    None    None    0
beautiful       wonderful       6.5     None    None    0
```

The data is saved into a file named “original-pairs-hypernyms.txt”. The columns represent word_1, word_2, similarity_between_words, hypernym_1, hypernym_2, similarity_between_hypernyms.

Task4: What are the 10 most similar pairs that you found for text1.txt? Print them to the file top.txt.

<u>Solution:</u>

<u>Sample Output:</u>

Word Pair	Similarity
(whistle, whistling)	1.0
(sew, sewed)	1.0
(discovered, noticed)	1.0
(vanished, disappeared)	1.0
(touch, touching)	1.0
(sewed, sew)	1.0
(sewed, sews)	1.0
(touching, touch)	1.0
(got, acquired)	1.0
(guile, cunning)	1.0

The top ten most similar pairs in the file "text1.txt" has been given above. It is saved to a file called "top.txt".

REFERENCES

[1] Miles Shang, 2011, *Syntax Tree Generator*, <http://mshang.ca/syntree/>

[2] EnglishClub.com, *When to say a or an*, <https://www.englishclub.com/pronunciation/a-an.htm>

[3] Richard Nordquist, 2019, ThoughtCo., *Third-Person Singular Verb Endings in English*, < https://www.thoughtco.com/third-person-singular-verb-ending-1692468>

[4] TutorialsPoint, *Compiler Design – Top Down Parser*, <https://www.tutorialspoint.com/compiler_design/compiler_design_top_down_parser.htm>