Sarnali Basak s1527144
Paridhi Mishra s1549106

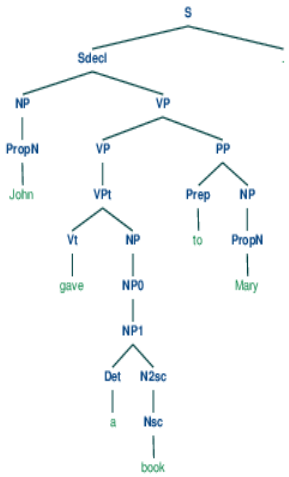
Tasks

Q.1. - Solution

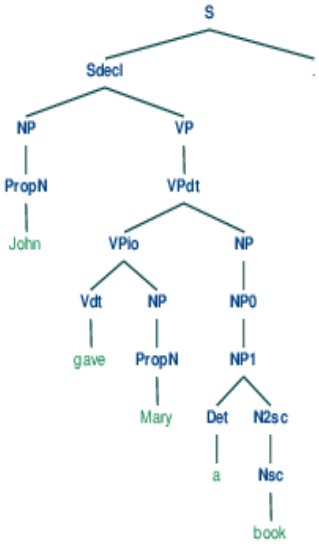
The comments are added in **ass2.py** file in the beginning which explains the three parts of the below mentioned regex pattern inside re.findall() function to implement the tokenize function as required.

```
re.findall(r"[\w]+'s|[\w]+|[\w'?]+|[~!@$%^&*(){}[].?]+",tokenstring)
```

Q.2. - Solution
Sentence 1 " John gave a book to Mary."
This sentence has two distinct complete parses.



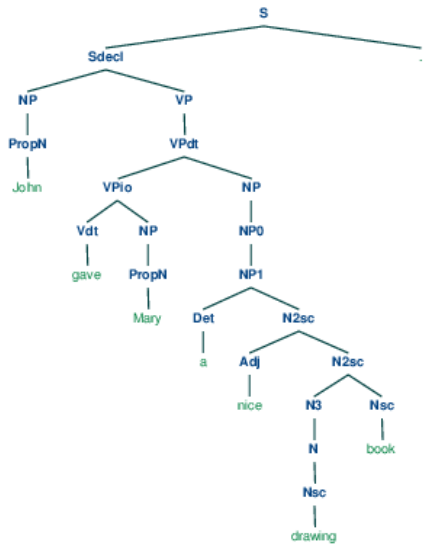
Parse Tree for Sentence 1



Parse Tree for Sentence 2

Sentence 2 " John gave Mary a book. "
This sentence has only one complete parse as given above.

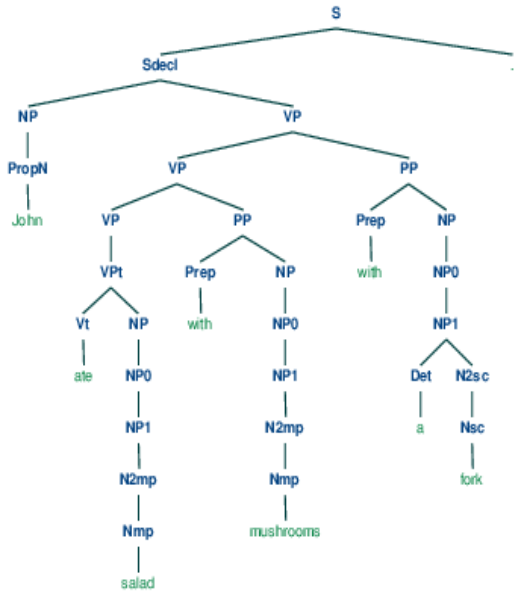
Sentence 3 " John gave Mary a nice drawing book."
There are two complete parses for this sentence.



Parse Tree for Sentence 3

Sarnali Basak s1527144
Paridhi Mishra s1549106

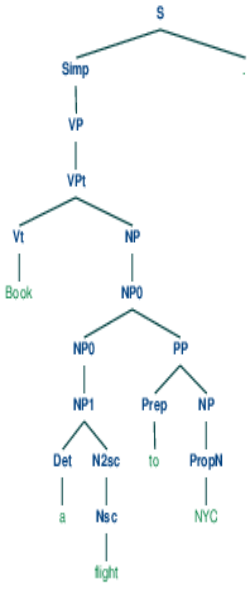
Sentence 4 "John ate salad with mushrooms with a fork. "
This sentence as four distinct complete parses. Following one is the most appropriate.



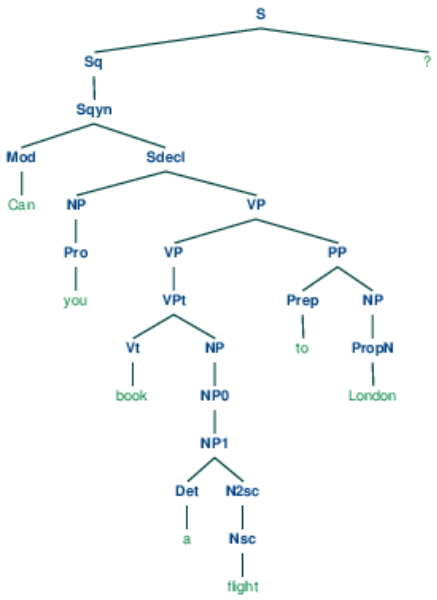
Parse Tree for Sentence 4

Sentence 5 " Book a flight to NYC."

This sentence has only one complete parse as given below.



Parse Tree for Sentence 5



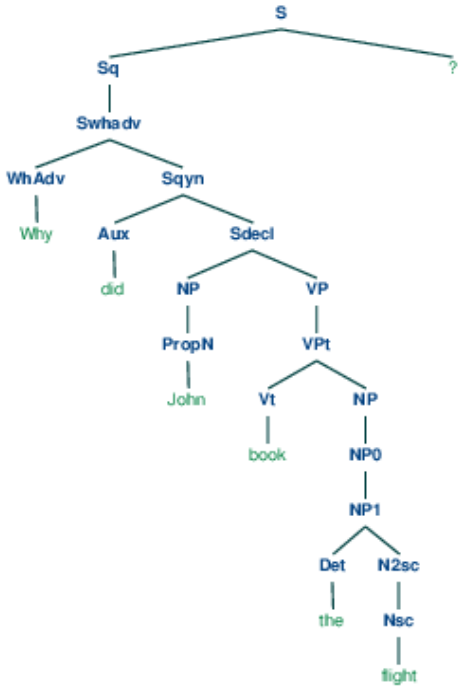
Parse Tree for Sentence 6

Sentence 6 " Can you book a flight to London? "

This sentence has two distinct complete parses. The most appropriate tree is given above.

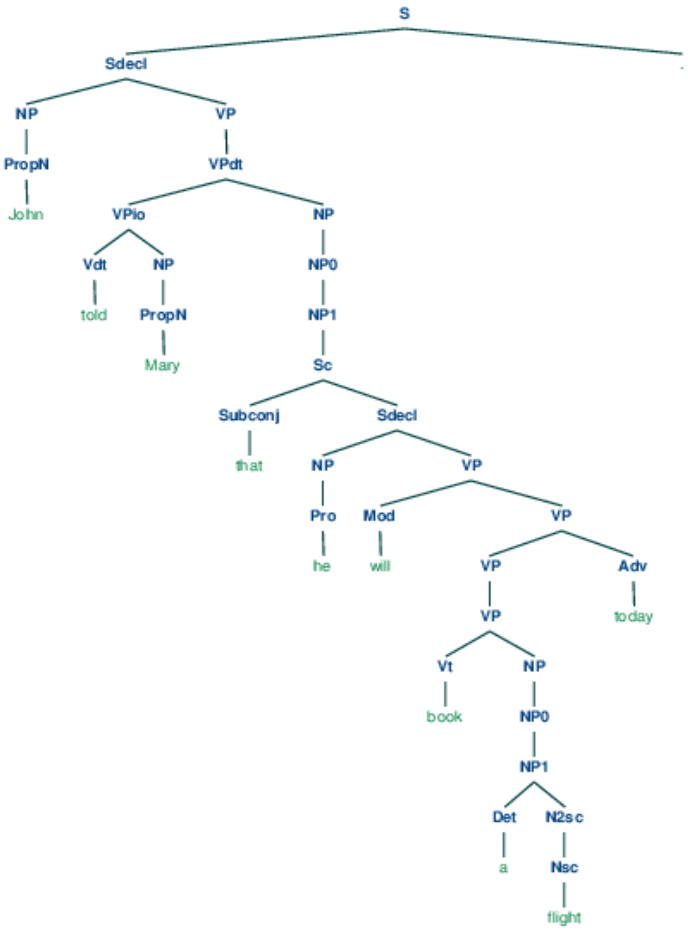
Sentence 7 " Why did John book the flight?"
This sentence has only one complete parse as given below in the left figure in the next page.

Sarnali Basak s1527144
Paridhi Mishra s1549106



Parse Tree for Sentence 7

Sentence 8 "John told Mary that he will book a flight today. "
There are two complete parses we found for this sentence.



Parse Tree for Sentence 8

Q3 - Solution

The purpose of a grammar is to mimic a natural language and approximate its behavior. This grammar is fairly descriptive and a little restrictive while handling some sentences. By scanning the grammar, it can be seen that there are a lot of unary rules in this grammar. After exploring the constructed parse trees, a pattern can be seen in which there are a lot of nodes with a single child in succession. That means that there is chaining of unary rules happening which can be replaced by removing the intermediate rules in case of **noun phrase handling** (*NP->NP0->NP1->N2mp->Nmp*). This factor also contributes to decreasing the efficiency of the parser since for every unary rule, the parser makes one new call to unary_update() function. Also the storage capacity of parser is limited and if the matrix is storing all the ambiguous subtrees at a time, it can lead to significant degradation in the performance of the parser. Similarly the **handling of the verb phrases** can be improved by removing some redundancy in the binary rules like *(VP0->Vdt NP)* and *(VPio->Vdt NP)* in which both the non-terminals *(VP0 and VPio)* lead to same set of non terminals . This may create difficulties while parsing the trees.

The grammar is weak in **handling conjoinability of sentences**. As the chunks of attachments increases in a sentence, the grammar creates multiple ambiguous parses for the same sentence. For example a sentence like "*The assignment can be written with latex using MacTex on a MacPro in the Drill Hall".* that already has attachment ambiguities will create multiple parses using this grammar. It can be handled by recognizing the conjuncts and adding more specific rules for these conjuncts like *"with", "using" and "on".*

Another aspect of this grammar which might be improved is it's **handling of terminals and non-terminals**. Take for example the binary rule *S->Sq"?".* This rule handles the terminal '?' and non-terminal 'Sq' at the same time, hence mixing the terminals and non-terminals in a binary rule which is not a good idea. It can be improved by replacing it with another unary rule e.g. *Q->'?'*

Other minor observations are :

To handle different types of sentences – declarative, imperative, interrogative and embedded sentences, the grammar can be simplified by introducing the following rule
S-> Sdecl | *Swhq* | *Synq* | *Simp*

It might also be helpful if this grammar also included words *"shall"/"do"* along with the current *"can"/"will"* which it handles.

When there are two consecutive adjectives in a sentence for e.g. "*nice*" and "*drawing*", the grammar produces ungrammatical strings.

Given a sentence like "He books the flight", the grammar does take "book" as a verb (along with being a noun), but if it accepts sentences with "book" verb, it should also accept sentences with "books" as a verb for third person. It can be improved by including the rules for all the variations (all the tenses) of the verbs it is already handling. So that a change in a sentence from passive to active or from first person to third person does not exclude the changed sentence from being accepted by this grammar.

Q.4. - Solution
The comments have been added to **cky.py** file under the CKY.build Indices method explaining line by line how it works and what the function does.

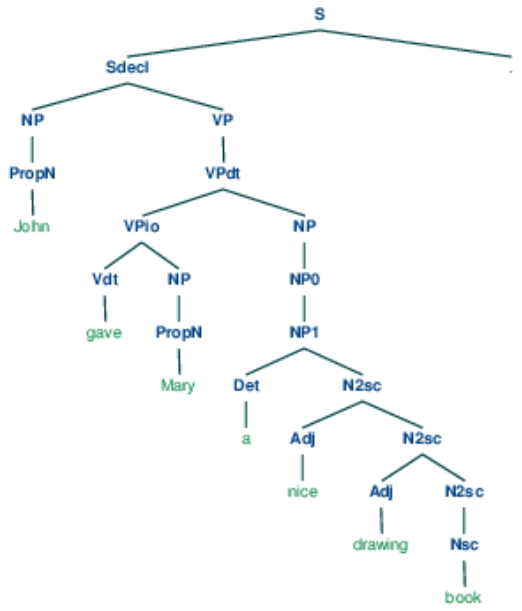
Q.5. - Solution
The comments have been added to **cky.py** file under the CKY.unary_fill and Cell.unary_update methods that explains what the function is doing. By line-by-line comments, it is also explained how the function is working.

Q.6. - Solution
The comments have been added to **cky.py** file to the CKY.parse method and its helper method maybe_build.

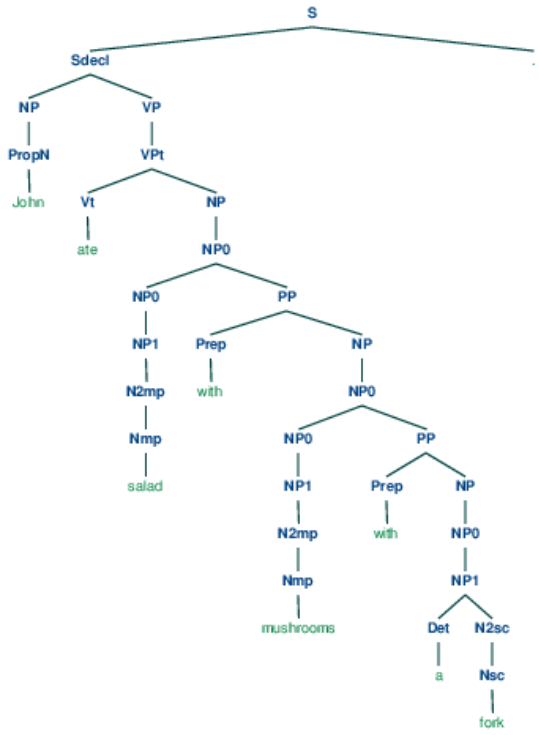
Q.7. - Solution
Sentence 1 " John gave a book to Mary."
This sentence has two distinct complete parse trees. Using reference (c) (see end of report) , it can be argued that the cause to be located of this statement is "gave" and "a book" is more closely associated with the cause "gave" than to "Mary".
The parse tree given in answer for Q2 emphasizes that "John gave a book" represents a better intuitive syntactic structure. *"To Mary"* part is not the primary subject. If it had been, the better representation for the same sentence will be *"John gave Mary a book*" instead of *"John gave a book to Mary".*

Sentence 2 " John gave Mary a book. "
This sentence has only one distinct complete parse. In this sentence, *"John gave Mary a book",* The *"Mary"* is an indirect dative where the verb *"gave"* is ditransitive with no preposition "*to"*. Besides, the meaning specifies an abstract *"have"* where John caused Mary to have the book. Though they did not have global ambiguities, it can be insufficient to parse by having the local ambiguities problem, like "Vdt/Vt-->gave" and "Nsc/Vt-->book".

Sentence 3 " John gave Mary a nice drawing book."
This sentence has two distinct complete parses. The grammar shows structural ambiguities when the grammar assigns the word two possible non-terminals to word "*drawing*". The tree that assigns noun to word "*drawing"* is presented in answer to Q2. Comparing that with the tree given below, which assigns adjective to the word *"drawing"* as well as to the word "*nice*", it can be argued that that it is unlikely that "*drawing*" is an adjective. So the tree in Q2 is more appropriate because the word "*drawing*" is placed after the word "*nice*", so drawing should be associated with a Nsc head and not with an Adj head.



Sentence 4 "John ate salad with mushrooms with a fork. "
This sentence has four distinct parses. Evidence of this can be found in matrix in cell(2,3) and (4,5) as it can be seen that the multiple noun phrases for "*salad*" and "*mushroom*". As these number of noun phrases increase, the chances of attachment ambiguities also increase, since there are more and more ways of attaching this chunk to the main sentence. The tree chosen in answer for Q2 is more meaningful because the sentence can be seen as two different chunks (though it has ambiguities), first being *"ate salad with mushrooms"* and second being "with a fork". In this parse, it clearly distinguishes John did something (ate something (*salad with mushrooms*)) using something (*with a fork*).
This sentence shows syntactic ambiguity with phrase attachments. *"With a fork"* can be attached with "*mushrooms"* or "*salads*" or "*ate*". But "*with a fork"* is better matched with "ate" than the other two. The tree shown below shows this ambiguity when "*with a fork*" associates with "*mushrooms*".



Sentence 5 " Book a flight to NYC."

There is one parse for this sentence but this sentence can have two meanings. "*Book"* can either used as noun (the object book) or a verb (the act of booking which indicates imperative sentence), thereby showing lexical ambiguity in this sentence.

Sentence 6 " Can you book a flight to London? "

This sentence has two distinct parses. It does not state clearly whether the flight that has to be booked is going to London or not. So it is specifying any flight that will go to London and not any particular flight. If instead of det 'a', det 'the' is used, then it will have a different meaning. Another way to look at this is *"Can you book a flight that is going to London*" will be less ambiguous because the destination now is more closely linked with the flight using relative pronoun "that".

Sentence 7 " Why did John book the flight?"

Even though this sentence has only one complete parse, this sentence can be emphasized in three ways depending on what is being focused on when we ask this question.

*Why did **John** book the flight?* (Why not someone else book the flight?)
*Why did John **book** the flight?* (Why did he book and not cancel the flight?)
*Why did John book the **flight**?* (Why did John book the flight and not the train?)

So this sentence can have implications and create ambiguities by emphasizing in different ways (emphatic stress – reference (d))

Sentence 8 "John told Mary that he will book a flight today. "

This sentence has two complete parses. When deciding between the full parses, the tree mentioned in answer to Q2 is relatively more meaningful because it clearly attaches the adverb "*today*" with verb phrase "*book a flight",* whereas the other parse tree is attaching the verb phrase (the act of John) separately from the adverb "*today*" implying that what John did was done today instead of booking the flight today.
Considering whether today is being referred to the act of telling (when did John tell Mary?) or act of booking (when the flight will be booked?), the tree in answer to Q2 is the most appropriate among the four parsed trees found. Another aspect of its ambiguity is *will John book the flight for Mary or will he book the flight for himself*? This question is not answered clearly by the sentence.

Q8 - Solution

From the chapter **CKY Recognition** (13.4.1 Jurafsky & Martin (2nd edition)), the CKY algorithm and the code (in given cky.py) that implements this algorithm is a "**recognizer**".

The CKY parser implemented in cky.py simply finds the labels belonging to the correct cell of the matrix and adds them if not already present by doing a look-up in 2 default dictionaries for unary and binary rules of the grammar. It does not store the information if one wants to print a complete parsed tree starting from top right of the matrix since one cannot say from which cell the symbol S was derived when it was added to this cell. So climbing down the parsed tree is not possible as is. By extending the features of this **CKY parser** by adding additional information, one can (after a successful run of CKY parse algorithm and filling of the matrix) navigate to the parsed tree starting from the node. This extension will allow to build functions like create_tree() and first_tree(). The code design and implementation is as follows:

Design:

The Label class design is modified to add 4 extra parameters

1) cell row position
2) cell column position
3) another label object corresponding to the left subtree of the current label
4) another label object corresponding to the right subtree of the current label
5)

The subtrees will later be constructed during first_tree() fn call. But the information for both is stored in the label class so this kind of backtracking will be possible later.

The Cell class design is modified to store a list of labels storing the complete information for all label instances that are assigned to this cell during the matrix building. It was previously storing only the string representations of the symbols.

There is an additional check which is introduced in the binary_scan() fn to store only one copy (i.e. the first copy) of a symbol (a terminal or a non-terminal) in cases where there are local and structural ambiguities present in the grammar for a particular the sentence.

There are many more code changes to various functions in the cky8.py file to accommodate the above changes. Please see the comments in the cky8.py file for all the changes at a particular line of the code.

Implementation:

To implement the extended CKY parsing, the code is changed at all the places where the new label is added to the list of labels for a particular cell. The printing of the completed parse tree is implemented with two functions first_tree(), which internally calls another recursive function create_tree() multiple times to generate the parse tree. Please note that this parse tree is traversed in a PreOrder fashion and outputs the tree string accordingly. This parenthesized tree string is then passed as a parameter to nltk.tree.Tree class, which generates a Tree instance representing the complete parse tree for the given sentence.

It is worth noting that if there are multiple parsed trees (for example for sentence 4(4 trees) and sentence 8(2 trees)), the first_tree() only outputs the first tree, which gets generated.

The design mentioned above is implemented in cky8.py file and this file is imported in ass2.py file.

The CKY chart implemented in cky.py now starts behaving as a parser instead of the original recognizer. Using backtracking and the information stored during the construction of this chart, the parse tree can be successfully created starting from right and top most corner of chart and following the start symbol S.

Q.9. - Solution

Note: Please uncomment the last 11 lines of ass2.pky and import cky9.py instead of cky8.py for this implementation.

Extending the features of the CKY parser from Q8, the file cky9.py implements another function all_trees(). This function is only possible when the check for multiple labels is removed from the binary_scan() function. The all_trees() returns a list of nltk.tree.Tree instances each of them representing a complete parse tree. It is interesting to note how sentence 4 and 8 have multiple parse trees generated using this function.

There is noticeable difference in the running time of the modified parser in Q8 as compared to original parser and again when compared to Q9. The modified parser is running slow, sometimes taking 3-5 seconds when parsing grammar 2. As we store more labels in the cells, the parser experiences

degradation in performance since for each cell the parser has to do all the operation multiple times for each copy of the same label.symbol.

References:

a) *https://en.wikipedia.org/wiki/Punctuation*
b) *http://www.bu.edu/linguistics/UG/course/lx522-f05/lx522/archives/85.html*
c) *http://ling-blogs.bu.edu/lx500f10/files/2010/09/lx500univf10-05b-have-handout.pdf*
d) *https://books.google.co.uk/books?id=s9ksGSs5ClkC&pg=PA155&lpg=PA155&dq=interrogative+sentences+examples+and+ambiguities&source=bl&ots=Uebdw6rgnf&sig=ZaDqkwVLivCYuwhr7zUJMQK5aNM&hl=en&sa=X&ved=0CCYQ6AEwAWoVChMIlJGs9OSKyQIVBbQaCh2SNQ3a#v=onepage&q=interrogative%20sentences%20examples%20and%20ambiguities&f=false*
e) *https://www.google.co.uk/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&cad=rja&uact=8&ved=0CCoQFjABahUKEwi41NDY9YrJAhUFfhoKHdfNCOw&url=http%3A%2F%2Fwww.mit.edu%2F~6.863%2Fspring2011%2Fjmnew%2F12.pdf&usg=AFQjCNEJ20-dDuaKeWfPRwdj6cf4Zbs6lw&sig2=dXsPAfDP_pUxBu4Y3CvHaA*