Name:	Pranay Singhvi
UID:	2021300126

Experiment 8

Aim:	To perform Constituency parsing
Problem Statement:	1. Constituency Parse manually a.Define the grammar (grammar of non terminals is fixed, grammar/lexicon for terminals will vary) b.Identify constituency using defined grammar and the POS tags of words which can be retrieved using POS tagger. c. If the new word is not in the grammar, add to the grammar 2. Constituency parsing using library 3. If the sentence is grammatically incorrect, give error in case 1 & 2.

Theory:

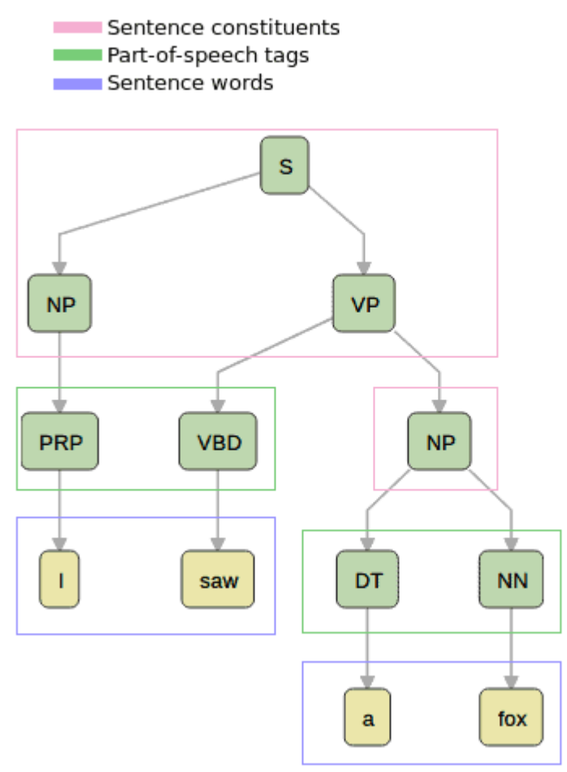
Constituency Parsing

The constituency parse tree is based on the formalism of context-free grammars. In this type of tree, the sentence is divided into constituents, that is, sub-phrases that belong to a specific category in the grammar.
In English, for example, the phrases “a dog”, “a computer on the table” and “the nice sunset” are all noun phrases, while “eat a pizza” and “go to the beach” are verb phrases.

The grammar provides a specification of how to build valid sentences, using a set of rules. As an example, the rule VP \rightarrow V NP means that we can form a verb phrase (VP) using a verb (V) and then a noun phrase (NP).

While we can use these rules to generate valid sentences, we can also apply them the other way around, in order to extract the syntactical structure of a given sentence according to the grammar.

Let’s dive straight into an example of a constituency parse tree for the simple sentence, “I saw a fox”:



A constituency parse tree always contains the words of the sentence as its terminal nodes. Usually, each word has a parent node containing its part-of-speech tag (noun, adjective, verb, etc…), although this may be omitted in other graphical representations.
All the other non-terminal nodes represent the constituents of the sentence and are usually one of verb phrase, noun phrase, or prepositional phrase (PP).

In this example, at the first level below the root, our sentence has been split into a noun phrase, made up of the single word “I”, and a verb phrase, “saw a fox”. This means that the grammar contains a rule like S \rightarrow NP VP, meaning that a sentence can be created with the concatenation of a noun phrase and a verb phrase.

Similarly, the verb phrase is divided into a verb and another noun phrase. As we can imagine, this also maps to another rule in the grammar.

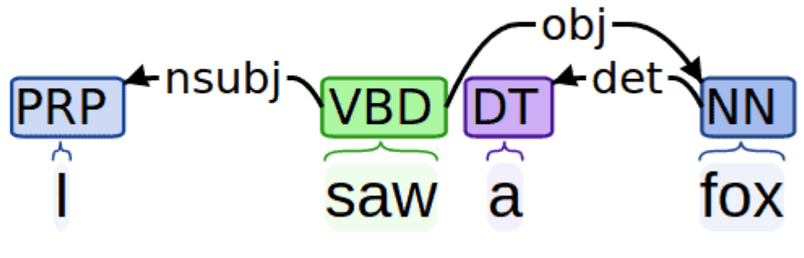
To sum things up, constituency parsing creates trees containing a syntactical representation of a sentence, according to a context-free grammar. This representation is highly hierarchical and divides the sentences into its single phrasal constituents.

Dependency Parsing

As opposed to constituency parsing, dependency parsing doesn’t make use of phrasal constituents or sub-phrases. Instead, the syntax of the sentence is expressed in terms of dependencies between words — that is, directed, typed edges between words in a graph.
More formally, a dependency parse tree is a graph G = (V, E) where the set of vertices V contains the words in the sentence, and each edge in E connects two words. The graph must satisfy three conditions:

1. There has to be a single root node with no incoming edges.
2. For each node v in V, there must be a path from the root R to v.
3. Each node except the root must have exactly 1 incoming edge. Additionally, each edge in E has a type, which defines the grammatical relation that occurs between the two words.

Let’s see what the previous example looks like if we perform dependency parsing:



As we can see, the result is completely different. With this approach, the root of the tree is the verb of the sentence, and edges between words describe their relationships. For example, the word “saw” has an outgoing edge of type nsubj to the word “I”, meaning that “I” is the nominal subject of the verb “saw”. In this case, we say that “I” depends on “saw”.

Importing Libraries

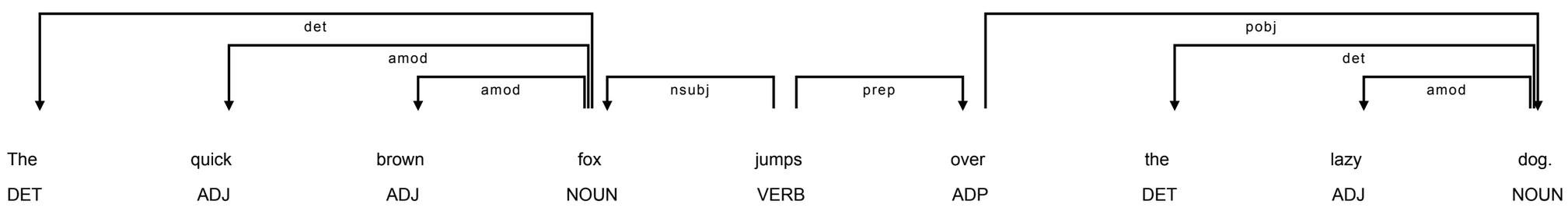
In [ ]:
```
import os
import nltk
import prettytable
from nltk import CFG
from nltk.tree import Tree
import matplotlib.pyplot as plt
```

```
from nltk.parse import CoreNLPParser
from nltk.parse.corenlp import CoreNLPServer
import spacy
from spacy import displacy
from nltk.tree import Tree as NLTKTree
```

Dependency Parsing

In [ ]:
```
import spacy
from spacy import displacy

# Load the English language model from spaCy
nlp = spacy.load("en_core_web_sm")

# Define a function to parse text and display the dependency parse tree graphically
def parse_and_display(text, port=None):
    # Process the text using spaCy
    doc = nlp(text)

    # Generate a dependency parse tree
    displacy.serve(doc, style="dep", port=port, options={'compact': True})

# Example text
text = "The quick brown fox jumps over the lazy dog."

# Parse and display the example text
parse_and_display(text, port=5001)  # Change the port number if needed
```

```
/Users/pranaysinghvi/Library/CloudStorage/OneDrive-Personal/SPIT College/3)Class/Semester 6/3)BAP/1)Experiments/ven-bap/lib/python3.12/site-packages/spacy/displacy/__init__.py:106:
UserWarning: [W011] It looks like you're calling displacy.serve from within a Jupyter notebook or a similar environment. This likely means you're already running a local web server,
so there's no need to make displaCy start another one. Instead, you should be able to replace displacy.serve with displacy.render to show the visualization.
  warnings.warn(Warnings.W011)
```




```
Using the 'dep' visualizer
Serving on http://0.0.0.0:5001 ...
```

```
127.0.0.1 - - [17/Apr/2024 10:55:14] "GET / HTTP/1.1" 200 7535
127.0.0.1 - - [17/Apr/2024 10:55:14] "GET /favicon.ico HTTP/1.1" 200 7535
Shutting down server on port 5001.
```

Constituency Parsing Using Recursive Descent Parser

In [ ]:
```
# Step 2: Define a function to get the POS tags of a sentence
def get_pos_tags(sentence):
    text = nltk.word_tokenize(sentence)
    return nltk.pos_tag(text)
# Step 3: Define the grammar rules
grammar_rules = """
    S -> NP VP
    NP -> Det N
    VP -> V NP
    Det -> 'the'
    N -> 'cat' | 'mouse'
    V -> 'chased'
"""
# Step 4: Define a function to parse the sentence using the grammar rules
def parse_sentence(sentence, grammar):
    rd_parser = nltk.RecursiveDescentParser(grammar)
    words = nltk.word_tokenize(sentence)
    for tree in rd_parser.parse(words):
        return tree
# Step 5: Define a function to display the parse tree
def display_tree(tree):
    tree.pretty_print()
# Step 6: Test the parser with a sentence
sentence = "the cat chased the mouse"
grammar = CFG.fromstring(grammar_rules)
tree = parse_sentence(sentence, grammar)
display_tree(tree)
```

```
              S
      ________|_____
     |              VP
     |         _____|___
     NP       |         NP
  ___|___     |      ___|____
Det      N    V    Det       N
 |       |    |     |        |
the     cat chased the     mouse
```

Constituency Parsing manually

In [ ]:
```
class Tree:
    def __init__(self, label, children=None):
        self.label = label
        self.children = children if children is not None else []

    def __repr__(self):
        if self.children:
            children_str = " ".join(map(str, self.children))
            return f"({self.label} {children_str})"
        else:
```

```
            return self.label


def tokenize(sentence):
    return sentence.split()


def parse(tokens, grammar, symbol, start=0):
    if symbol not in grammar:  # terminal symbol
        return (start < len(tokens) and tokens[start] == symbol, start + 1, symbol)
    else:  # non-terminal symbol
        for rule in grammar[symbol]:
            pos = start
            children = []
            for token in rule:
                success, pos, child = parse(tokens, grammar, token, pos)
                if not success:
                    break
                children.append(child)
            else:
                return (True, pos, Tree(symbol, children))
        return (False, start, None)


grammar_rules = {
    "S": [["NP", "VP"]],
    "NP": [["Det", "Adj", "N"], ["Det", "N"]],
    "VP": [["V", "Det", "Adj", "N"]],
    "Det": [["The"], ["the"]],
    "N": [["child"], ["forest"]],
    "V": [["explored"]],
    "Adj": [["curious"], ["mysterious"]],
}

def display_tree(tree):
    tree_str = str(tree)
    nltk_tree = NLTKTree.fromstring(tree_str)
    nltk_tree.pretty_print()


sentence = "The curious child explored the mysterious forest"
tokens = tokenize(sentence)
success, pos, tree = parse(tokens, grammar_rules, "S")
if success and pos == len(tokens):
    display_tree(tree)  # make sure the entire input is consumed
```

```
                    S
       _____________|_____
      NP                   VP
  ____|______        ______|_____________
Det   Adj     N      V     Det    Adj       N
 |     |      |      |      |      |        |
The curious child explored the mysterious forest
```

Conclusion

In this experiment, we learned about Constituency Parsing and Dependency Parsing. We also learned how to perform Constituency Parsing using Recursive Descent Parser and manually.