Name: Bodhisatya Ghosh
UID: 2021700026
Branch: CSE DS
Experiment: 8
Batch: A

Theory

Constituency parsing is an important step in natural language processing and is used in a wide range of applications, such as natural language understanding, machine translation, and text summarization.

Constituency parsing is different from dependency parsing, which aims to identify the syntactic relations between words in a sentence. Constituency parsing focuses on the hierarchical structure of the sentence, while dependency parsing focuses on the linear structure of the sentence. Both techniques have their own advantages and can be used together to better understand a sentence.

Some challenges in Constituency Parsing are long-distance dependencies, syntactic ambiguity, and the handling of idiomatic expressions, which makes the parsing process more complex.

Applications of Constituency Parsing Constituency parsing is a process of identifying the constituents (noun phrases, verbs, clauses, etc.) in a sentence and grouping them into a tree-like structure that represents the grammatical relationships among them.

The following are some of the applications of constituency parsing:

- Natural Language Processing (NLP) – It is used in various NLP tasks such as text summarization, machine translation, question answering, and text classification.
- Information Retrieval – It is used to extract information from large corpora and to index it for efficient retrieval.
- Text-to-Speech – It helps in generating human-like speech by understanding the grammar and structure of the text.
- Sentiment Analysis – It helps in determining the sentiment of a text by identifying positive, negative, or neutral sentiments in the constituents.
- Text-based Games and Chatbots – It helps in generating more human-like responses in text-based games and chatbots.

- Text Summarization – It is used to summarize large texts by identifying the most important constituents and representing them in a compact form.
- Text Classification – It is used to classify text into predefined categories by analyzing the constituent structure and relationships.

In [ ]:
```
import stanza

nlp = stanza.Pipeline(lang='en', processors='tokenize,pos,constituency')
doc = nlp('This is a test')
for sentence in doc.sentences:
    display(sentence.constituency)
```

```
2024-04-17 18:30:29 INFO: Checking for updates to resources.json in case models have been updated.  Note: this behavior can be turned off with download_method=None or download_method=DownloadMethod.REUSE_RESOURCES
Downloading https://raw.githubusercontent.com/stanfordnlp/stanza-resources/main/resources_1.8.0.json:   0%|   …
2024-04-17 18:30:29 INFO: Downloaded file to C:\Users\Rommel\stanza_resources\resources.json
2024-04-17 18:30:29 WARNING: Language en package default expects mwt, which has been added
2024-04-17 18:30:30 INFO: Loading these models for language: en (English):
=====================================
| Processor    | Package             |
-------------------------------------
| tokenize     | combined            |
| mwt          | combined            |
| pos          | combined_charlm     |
| constituency | ptb3-revised_charlm |
=====================================

2024-04-17 18:30:30 INFO: Using device: cpu
2024-04-17 18:30:30 INFO: Loading: tokenize
2024-04-17 18:30:30 INFO: Loading: mwt
2024-04-17 18:30:30 INFO: Loading: pos
2024-04-17 18:30:30 INFO: Loading: constituency
2024-04-17 18:30:31 INFO: Done loading processors!
(ROOT (S (NP (DT This)) (VP (VBZ is) (NP (DT a) (NN test)))))
```

In [ ]:
```
import nltk
grammar = nltk.CFG.fromstring("""
    S -> NP VP
    NP -> DT NN | NP PP
    VP -> VBD NP | VP PP
```

```
    PP -> P NP
    DT -> 'the' | 'a'
    NN -> 'dog' | 'cat' | 'ball' | 'bone' | 'park' | 'stick'
    VBD -> 'chased' | 'ate'
    P -> 'with' | 'in'
""")

sentences = [
    "the dog chased the cat",
    "a cat chased the ball",
    "the dog chased the ball with a stick",
]

for sentence in sentences:
    print("Sentence:", sentence)
    tokens = nltk.word_tokenize(sentence)
    parser = nltk.ChartParser(grammar)
    trees = list(parser.parse(tokens))
    for tree in trees:
        print(tree)
        print("\n")
```

```
Sentence: the dog chased the cat
(S (NP (DT the) (NN dog)) (VP (VBD chased) (NP (DT the) (NN cat))))


Sentence: a cat chased the ball
(S (NP (DT a) (NN cat)) (VP (VBD chased) (NP (DT the) (NN ball))))


Sentence: the dog chased the ball with a stick
(S
  (NP (DT the) (NN dog))
  (VP
    (VP (VBD chased) (NP (DT the) (NN ball)))
    (PP (P with) (NP (DT a) (NN stick)))))


(S
  (NP (DT the) (NN dog))
  (VP
    (VBD chased)
    (NP (NP (DT the) (NN ball)) (PP (P with) (NP (DT a) (NN stick))))))
```