	Bansilal Ramnath Agarwal Charitable Trust's Vishwakarma Institute of Information Technology **Department of** **Artificial Intelligence and Data Science**	
Name: **Siddhesh Dilip Khairnar**		
Class: **TY**	Division: **B**	Roll No: **372028**
Semester: **6th**	Academic Year: **2023-24**	
Subject Name & Code: **Natural Language and Processing & ADUA32203**		
Title of Assignment: **Study Recursive Descent and Shift Reduce Parser and use them for parsing a sentence.**		
Date of Performance: **17-02-2024**	Date of Submission: **24-02-2024**	

ASSIGNMENT NO: - 6

Aim: Implement CKY algorithm for deep parsing.

Steps:

1) **Consider any 5 sentences of your choice. (Try to maintain variety of phrases in sentences)**
2) **Print output in tree format.**
3) **Analyze the output.**

❖ **THEORY:**

- CKY and Earley are two styles of parsing for context-free grammar (CFGs).
- They ensure that, given a set of grammar rules and an input sentence, all possible parses of that sentence are extracted.
- The results are the possible parse trees for the complete sentence, and all the subtrees that were found during the extraction of this parse tree.
  CKY stands for Cocke-Kasami-Younger, and involves parsing substrings of length 1, then length 2, and so on until the entire string has been parsed.
- The reason why this is useful is because the shorter substrings from previous iterations can be used when applying grammar rules to parse the longer substrings. In this way, CKY implements bottom-up parsing, without forgetting substring groupings that had been done previously.
- The Earley parsing algorithm is similar in that it tries to use grammar rules to build non-terminals from substrings. However, instead of applying grammar rules for substrings of increasing length, it reads in
  words one at a time and applies any grammar rules that are allowed by these input words. To do this, the parser stores a list of partially completed grammar rules.

	b	a	a	b	a
b	{B}	{S,A}	Φ	Φ	{S,A,C}
a		{A,C}	{B}	{B}	{S,A,C}
a			{A,C}	{S,C}	{B}
b				{B}	{S,A}
a					{A,C}

Code:

In [1]:
```
import matplotlib.pyplot as plt
import nltk
```

In [2]:
```
nltk.download('maxent_treebank_pos_tagger')
nltk.download('treebank')
```

```
[nltk_data] Downloading package maxent_treebank_pos_tagger to
[nltk_data]     C:\Users\sejbp\AppData\Roaming\nltk_data...
[nltk_data]   Package maxent_treebank_pos_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package treebank to
[nltk_data]     C:\Users\sejbp\AppData\Roaming\nltk_data...
[nltk_data]   Package treebank is already up-to-date!
```

Out[2]: True

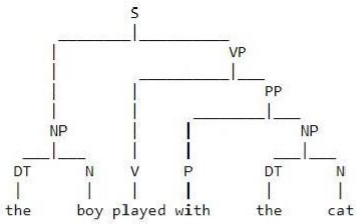
In [3]:
```
grammar = nltk.CFG.fromstring('''
    S -> NP VP | NP VP PP
    NP -> DT N | DT NN | DT JJ N
    VP -> V NP | V PP | V ADVP
    PP -> P NP
    ADVP -> RB
    DT -> 'a' | 'the'
    N -> 'dog' | 'cat' | 'mat' | 'boy' | 'ball'
    V -> 'chased' | 'sat' | 'played'
    P -> 'on' | 'with'
    RB -> 'quickly'
''')
```

In [5]:
```
text1 = "the dog chased the cat"

parser = nltk.ChartParser(grammar)
tokens = nltk.word_tokenize(text1)  # Tokenize the input string
trees = parser.parse(tokens)

for tree in trees:
    tree.pretty_print()
    tree.draw()
    #plt.show()
```

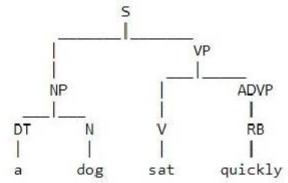
In [6]:
```
text2 = "a cat sat on the mat"

parser = nltk.ChartParser(grammar)
tokens = nltk.word_tokenize(text2)  # Tokenize the input string
trees = parser.parse(tokens)

for tree in trees:
    tree.pretty_print()
    tree.draw()
    #plt.show()
```

In [7]:
```
text3 = "the boy played with the cat"

parser = nltk.ChartParser(grammar)
```

```
    tree.pretty_print()
    tree.draw()
    #plt.show()
```

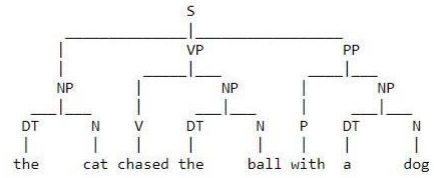


In [8]:
```
text4 = "a dog sat quickly"

parser = nltk.ChartParser(grammar)
tokens = nltk.word_tokenize(text4)  # Tokenize the input string
trees = parser.parse(tokens)

for tree in trees:
    tree.pretty_print()
    tree.draw()
    #plt.show()
```



In [9]:
```
text5 = "the cat chased the ball with a dog"
```

In [9]:
```
text5 = "the cat chased the ball with a dog"

parser = nltk.ChartParser(grammar)
tokens = nltk.word_tokenize(text5)  # Tokenize the input string
trees = parser.parse(tokens)

for tree in trees:
    tree.pretty_print()
    tree.draw()
    #plt.show()
```



In [ ]:
```
```

Conclusion:

Implementing the CKY algorithm for deep parsing allows us to analyze sentences by generating syntactic parse trees. This algorithm is efficient for parsing with context-free grammar and provides insights into the hierarchical structure of sentences.