Name : Sarvagya Singh **SAPID : 60009200030** **BATCH : K1**

Experiment No 5

Aim: - Implement CKY algorithm for Parsing the given string and Generate recursive set of sentences using Context Free Grammar

Theory:

Context Free Grammar

CFGS the backbone of many formal models of the syntax of natural language (and, for that matter, of computer languages). Context Free Grammar play a role in many computational applications, including grammar checking, semantic interpretation, dialogue understanding, and machine translation. It express sophisticated relations among the words in a sentence, yet computationally tractable enough that efficient algorithms exist for parsing sentences with them

Syntactic constituency

Syntactic constituency is the idea that **groups of words can behave as single units**, or constituents. For example, group of words come together to form a Noun Phrase. Part of developing a grammar involves building an inventory of the constituents in the language. Consider noun phrase the noun phrase, a sequence of words surrounding at least one noun. Here are some examples of noun phrases.

Harry the Horse	a high-class spot such as Mindy's
the Broadway coppers	the reason he comes into the Hot Box
they	three parties from Brooklyn

Context Free Grammar

It is formal system for modeling constituent structure in English and other natural languages is the Context-Free Grammar, or CFG.It is also called Phrase-Structure Grammars. A context-free grammar consists of a set of rules or productions. Which expresses the ways that symbols of the language can be grouped and ordered together, and a lexicon of words and symbols. Following is the Example of production rules of - CFG

$$NP \rightarrow Det\ Nominal$$
$$NP \rightarrow ProperNoun$$
$$Nominal \rightarrow Noun \mid Nominal\ Noun$$

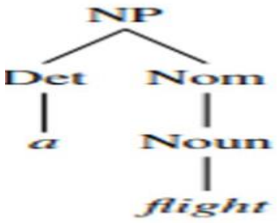
Context-free rules can be hierarchically embedded, so we can combine the previous rules with others, like the following, that express facts about the lexicon:

$$Det \rightarrow a$$
$$Det \rightarrow the$$
$$Noun \rightarrow flight$$

CFG as a generator

A CFG can be used to generate a set of strings. This sequence of rule expansions is called a derivation of the string of words. It is common to represent a derivation by a parse tree

(commonly shown inverted with the root at the top). Figure below "A parse tree for "a flight"" shows the tree representation of this derivation



Formal Definition of Context-Free Grammar

A context-free grammar G is defined by four parameters: N, Σ, R, S (technically this is a "4-tuple").

N	a set of **non-terminal symbols** (or **variables**)
Σ	a set of **terminal symbols** (disjoint from N)
R	a set of **rules** or productions, each of the form $A \rightarrow \beta$,
	where A is a non-terminal,
	β is a string of symbols from the infinite set of strings $(\Sigma \cup N)^*$
S	a designated **start symbol** and a member of N

Constituency parsing

The problem of mapping from a string of words to its parse tree is called syntactic parsing; Syntactic parsing is the task of assigning a syntactic structure to a sentence. Parse trees can be used in applications such as grammar checking sentence that cannot be parsed may have grammatical errors (or at least be hard to read). Parse trees can be an intermediate stage of representation for semantic analysis .

The grammar for L0, with example phrases for each rule

Grammar Rules			Examples
S	→	*NP VP*	I + want a morning flight
NP	→	*Pronoun*	I
	|	*Proper-Noun*	Los Angeles
	|	*Det Nominal*	a + flight
Nominal	→	*Nominal Noun*	morning + flight
	|	*Noun*	flights
VP	→	*Verb*	do
	|	*Verb NP*	want + a flight
	|	*Verb NP PP*	leave + Boston + in the morning
	|	*Verb PP*	leaving + on Thursday
PP	→	*Preposition NP*	from + Los Angeles

Example of lexicons

Noun →	*flights* | *breeze* | *trip* | *morning*
Verb →	*is* | *prefer* | *like* | *need* | *want* | *fly*
Adjective →	*cheapest* | *non-stop* | *first* | *latest* | *other* | *direct*
Pronoun →	*me* | *I* | *you* | *it*
Proper-Noun →	*Alaska* | *Baltimore* | *Los Angeles* | *Chicago* | *United* | *American*
Determiner →	*the* | *a* | *an* | *this* | *these* | *that*
Preposition →	*from* | *to* | *on* | *near*
Conjunction →	*and* | *or* | *but*

Cocke-Younger-Kasami Algorithm

It is used to solves the membership problem using a dynamic programming approach. The algorithm is based on the principle that the solution to problem [i, j] can constructed from solution to sub problem [i, k] and solution to sub problem [k, j]. The algorithm requires the Grammar G to be in Chomsky Normal Form (CNF). Note that any Context-Free Grammar can be systematically converted to CNF. This restriction is employed so that each problem can only be divided into two sub problems and not more – to bound the time complexity.

How does the CYK Algorithm work?

For a string of length **N**, construct a table **T** of size **N x N**. Each cell in the table **T[i, j]** is the set of all constituents that can produce the substring spanning from position i to j. The process involves filling the table with the solutions to the sub problems encountered in the bottom-up parsing process. Therefore, cells will be filled from left to right and bottom to top.

	1	2	3	4	5
1	[1, 1]	[1, 2]	[1, 3]	[1, 4]	[1, 5]
2		[2, 2]	[2, 3]	[2, 4]	[2, 5]
3			[3, 3]	[3, 4]	[3, 5]
4				[4, 4]	[4, 5]
5					[5, 5]

In T[i, j], the row number i denotes the start index and the column number j denotes the end index. The algorithm considers every possible subsequence of letters and adds **K** to T[i, j] if the sequence of letters starting from i to j can be generated from the non-terminal K. For subsequence of length 2 and greater, it considers every possible partition of the subsequence into two parts, and checks if there is a rule of the form A -> BC in the grammar where B and C can generate the two parts respectively, based on already existing entries in T. The sentence can be produced by the grammar only if the entire string is matched by the start symbol, i.e, if S is a member of T[1, n].

Example: -
Consider a sample grammar in Chomsky Normal Form

NP --> Det | Nom
Nom --> AP | Nom
AP --> Adv | A
Det --> a | an
Adv --> very | extremely
AP --> heavy | orange | tall
A --> heavy | orange | tall | muscular
Nom --> book | orange | man

Now consider the phrase, "a very heavy orange book":

a(1) very(2) heavy (3) orange(4) book(5)

Let us start filling up the table from left to right and bottom to top, according to the rules described above:

	1 a	2 very	3 heavy	4 orange	5 book
1 a	Det	–	–	NP	NP
2 very		Adv	AP	Nom	Nom
3 heavy			A, AP	Nom	Nom
4 orange				Nom, A, AP	Nom
5 book					Nom

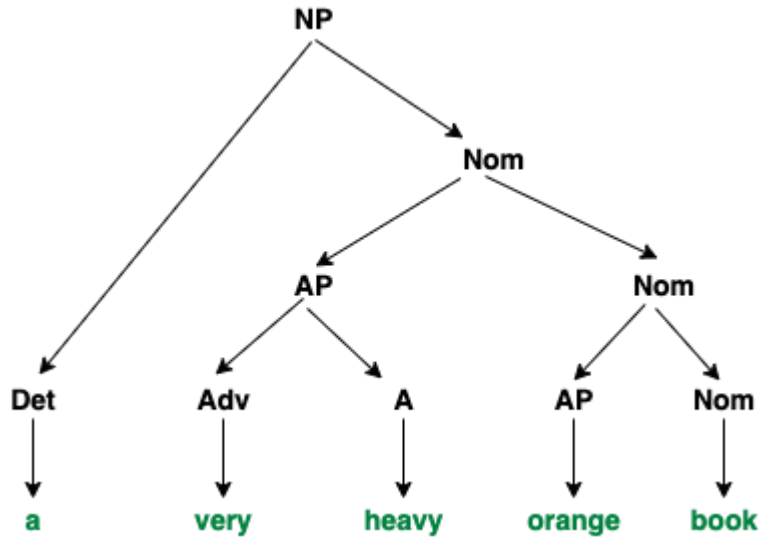
The table is filled in the following manner:

1. T [1, 1] = {Det} as Det –> a is one of the rules of the grammar.
2. T [2, 2] = {Adv} as Adv –> very is one of the rules of the grammar.
3. T [1, 2] = {} as no matching rule is observed.
4. T [3, 3] = {A, AP} as A –> very and AP –> very are rules of the grammar.
5. T [2, 3] = {AP} as AP –> Adv (T [2, 2]) A (T [3, 3]) is a rule of the grammar.
6. T [1, 3] = {} as no matching rule is observed.
7. T [4, 4] = {Nom, A, AP} as Nom –> orange and A –> orange and AP –> orange are rules of the grammar.
8. T [3, 4] = {Nom} as Nom –> AP (T [3, 3]) Nom (T [3, 4]) is a rule of the grammar.
9. T [2, 4] = {Nom} as Nom –> AP (T [2, 3]) Nom (T [4, 4]) is a rule of the grammar.
10. T [1, 4] = {NP} as NP –> Det (T [1, 1]) Nom (T [2, 4]) is a rule of the grammar.
11. T [5, 5] = {Nom} as Nom –> book is a rule of the grammar.
12. T [4, 5] = {Nom} as Nom –> AP (T [4, 4]) Nom (T [5, 5]) is a rule of the grammar.

13. T [3, 5] = {Nom} as Nom –> AP (T [3, 3]) Nom (T [4, 5]) is a rule of the grammar.
14. T [2, 5] = {Nom} as Nom –> AP (T [2, 3]) Nom (T [4, 5]) is a rule of the grammar.
15. T [1, 5] = {NP} as NP –> Det (T [1, 1]) Nom (T [2, 5]) is a rule of the grammar.

We see that T [1][5] has **NP**, the start symbol, which means that this phrase is a member of the language of the grammar *G*.

The parse tree of this phrase would look like this:



Lab Experiment to be performed in this Session: -

Write a program to generate sentences using Context free grammar.
Write a program to implement CKY algorithm to parse a given string.

Sarvagya Singh

60009200030 - K1

Context Free Grammar

In [4]:

```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

Out[4]:

```
True
```

In [5]:

```
import random

grammar = {
    "<start>": ["<noun_phrase> <verb_phrase>"],
    "<noun_phrase>": ["<determiner> <noun>", "<determiner> <adjective> <noun>"],
    "<verb_phrase>": ["<verb>", "<verb> <noun_phrase>"],
    "<determiner>": ["the", "a"],
    "<adjective>": ["big", "small", "red", "green", "blue"],
    "<noun>": ["cat", "dog", "house", "car", "tree"],
    "<verb>": ["runs", "jumps", "barks", "drives", "sleeps"]
}

def generate_sentence(grammar, symbol):
    if symbol not in grammar:
        return symbol
    else:
        expansion = random.choice(grammar[symbol])
        return " ".join(generate_sentence(grammar, s) for s in expansion.split())

sentence = generate_sentence(grammar, "<start>")
print(sentence)
```

```
a blue dog barks a small tree
```

In [6]:

```
import nltk

def parse_cky(sentence, grammar):
    words = nltk.word_tokenize(sentence)
    n = len(words)

    # Initialize the table
    table = [[set() for j in range(n-i)] for i in range(n)]

    # Fill in the diagonal with the words
    for i, word in enumerate(words):
        table[i][0] = set(grammar.productions(rhs=word))

    # Fill in the rest of the table
    for j in range(1, n):
        for i in range(j-1, -1, -1):
            for k in range(i+1, j+1):
                for prod in grammar.productions():
                    if len(prod.rhs()) == 2:
                        A, B = prod.rhs()
                        if B in table[k-1][j-i-k] and A in table[i][k-1]:
```

```
                            table[i][j-i].add(prod)

    # Return the final result
    if nltk.grammar.Nonterminal('S') in table[0][n-1]:
        return True
    else:
        return False
```

In [7]:

```
grammar = nltk.CFG.fromstring("""
    S -> NP VP
    NP -> Det N | N
    VP -> V NP | V
    Det -> 'the' | 'a'
    N -> 'cat' | 'dog' | 'tree' | 'house'
    V -> 'runs' | 'jumps' | 'barks' | 'sleeps'
""")

sentence = "the cat jumps"
result = parse_cky(sentence, grammar)

if result:
    print(f"The sentence '{sentence}' is grammatically correct.")
else:
    print(f"The sentence '{sentence}' is not grammatically correct.")
```

```
The sentence 'the cat jumps' is not grammatically correct.
```

In [ ]:

```
!jupyter nbconvert --to html ''
```