Vidyavardhaka Sangha®, Mysore
VIDYAVARDHAKA COLLEGE OF ENGINEERING
Autonomous Institute, Affiliated to Visvesvaraya Technological University, Belagavi
(Approved by AICTE, New Delhi & Government of Karnataka)
Accredited by NBA | NAAC with 'A' Grade
Department of Computer Science & Engineering
Phone: +91 821-4276230, Email: hodcs@vvce.ac.in
Web: http://www.vvce.ac.in
@vvceofficial

# AUTOMATA THEORY
# (20CS53)
# ACTIVITY BASED ASSESMENT

## TOPIC: CYK (COCKE-YOUNGER-KASAMI) ALGORITHM

**BY,**

**AMITH A (4VV20CS008)**
**ARJUN ARUN(4VV20CS014)**
**ANIRUDHA D (4VV20CS011)**

# CYK ALGORITHM

- It is a Membership Algorithm
- To check whether the given string belongs to a particular language or not.
- It is a parsing algorithm for context free grammar (CFG).
- In order to apply CYK algorithm to a grammar, it must be in Chomsky Normal Form.
- It uses a dynamic programming algorithm to tell whether a string is in the language of a grammar.
- It is universal, i.e. It is applicable for all CNF.
- Time Complexity → $O(n^3)$
- Space Complexity → $O(n^2)$

## Algorithm:

Let w be the n length string to be parsed. And G represent the set of rules in our grammar with start state S.

1. Construct a table DP for size n × n.

2. If w = e (empty string) and S -> e is a rule in G then we accept the string else we reject.

3. For i = 1 to n:
    For each variable A:
        We check if A -> b is a rule and b = wi for some i:
            If so, we place A in cell (i, i) of our table

4. For l = 2 to n:
    For i = 1 to n-l+1:
        j = i+l-1
        For k = i to j-1:
            For each rule A -> BC:
We check if (i, k) cell contains B and (k + 1, j) cell contains C:
        If so, we put A in cell (i, j) of our table.

5. We check if S is in (1, n):
        If so, we accept the string
        Else, we reject.

## **Example Problem:**

Let our context free grammar G be:

S -> AB | BC

A -> BA | a

B -> CC | b

C -> AB | a

We check if “baaba” is in L(G):

SOLUTION:

1.We first insert single length rules into our table.

| | b | a | a | b | a |
|---|---|---|---|---|---|
| b | {B} | | | | |
| a | | {A,C} | | | |
| a | | | {A,C} | | |
| b | | | | {B} | |
| a | | | | | {A,C} |

2.We then fill the remaining cells of our table.

| | 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} |

3.We observe that S is in the cell (1, 5), Hence, the string “baaba” belongs to L(G).

# CYK ALGORITHM IMPLEMENTATION USING JAVA PROGRAM

```java
import java.io.*;
import java.util.*;

public class CYK{

    public static String word;
    public static String startingSymbol;
    public static boolean isTokenWord = false;
    public static ArrayList<String> terminals = new ArrayList<String>();
    public static ArrayList<String> nonTerminals = new ArrayList<String>();
    public static TreeMap<String,ArrayList<String>> grammar = new TreeMap<>();

    public static void main(String[] args){
        if(args.length < 2){
            System.out.println("Usage: java CYK <File> <Word>.");
            System.exit(1);
        }else if (args.length > 2){
            isTokenWord = true;
        }
        doSteps(args);
    }

    public static void doSteps(String[] args){
        parseGrammar(args);
        String[][] cykTable = createCYKTable();
        printResult(doCyk(cykTable));
    }

    public static void parseGrammar(String[] args){
        Scanner input = openFile(args[0]);
        ArrayList<String> tmp = new ArrayList<>();
        int line = 2;

        word = getWord(args);
        startingSymbol = input.next();
        input.nextLine();

        while(input.hasNextLine() && line <= 3){
            tmp.addAll(Arrays.<String>asList(toArray(input.nextLine())));
            if(line == 2) { terminals.addAll(tmp); }
            if(line == 3) { nonTerminals.addAll(tmp); }
            tmp.clear();
            line++;
        }

        while(input.hasNextLine()){
            tmp.addAll(Arrays.<String>asList(toArray(input.nextLine())));
            String leftSide = tmp.get(0);
```

```java
public static String getWord(String[] args){
    if(!isTokenWord) { return args[1]; }
    String[] argsWithoutFile = new String[args.length - 1];
    for(int i = 1; i < args.length; i++){
        argsWithoutFile[i-1] = args[i];
    }
    return toString(argsWithoutFile);
}

public static void printResult (String[][] cykTable){
    System.out.println("Word: " + word);
    System.out.println("\nG = (" + terminals.toString().replace("[", "{").replace("]", "}")
                    + ", " + nonTerminals.toString().replace("[", "{").replace("]", "}")
                    + ", P, " + startingSymbol + ")\n\nWith Productions P as:");
    for(String s: grammar.keySet()){
        System.out.println(s + " -> " + grammar.get(s).toString().replaceAll("[\\[\\]\\,]", "").replaceAll("\\s", " | "));
    }
    System.out.println("\nApplying CYK-Algorithm:\n");
    drawTable(cykTable);
}

public static void drawTable(String[][] cykTable){
    int l = findLongestString(cykTable) + 2;
    String formatString = "| %-" + l + "s ";
    String s = "";
    StringBuilder sb = new StringBuilder();
//Building Table Structure Modules
    sb.append("+");
    for(int x = 0; x <= l + 2; x++){
        if(x == l + 2){
            sb.append("+");
        }else{
            sb.append("-");
        }
    }
    String low = sb.toString();
    sb.delete(0, 1);
    String lowRight = sb.toString();
//Print Table
    for(int i = 0; i < cykTable.length; i++){
        for(int j = 0; j <= cykTable[i].length; j++){
            System.out.print((j == 0) ? low : (i <= 1 && j == cykTable[i].length - 1) ? "" : lowRight);
        }
        System.out.println();
        for(int j = 0; j < cykTable[i].length; j++){
            s = (cykTable[i][j].isEmpty()) ? "-" : cykTable[i][j];
            System.out.format(formatString, s.replaceAll("\\s", ","));
            if(j == cykTable[i].length - 1) { System.out.print("|"); }
        }
```

```
        }
        System.out.println(low+"\n");
    //Step 4: Evaluate success.
        if(cykTable[cykTable.length-1][cykTable[cykTable.length-1].length-1].contains(startingSymbol)){
            System.out.println("The word \"" + word + "\" is an element of the CFG G and can be derived from it.");
        }else{
            System.out.println("The word \"" + word + "\" is not an element of the CFG G and can not be derived from it.");
        }
    }

    public static int findLongestString(String[][] cykTable){
        int x = 0;
        for(String[] s : cykTable){
            for(String d : s){
                if(d.length() > x){ x = d.length(); }
            }
        }
        return x;
    }

//Jagged Array for the Algorithm
    public static String[][] createCYKTable (){
        int length = isTokenWord ? toArray(word).length : word.length();
        String[][] cykTable = new String[length + 1][];
        cykTable[0] = new String[length];
        for(int i = 1; i < cykTable.length; i++){
            cykTable[i] = new String[length - (i - 1)];
        }
        for(int i = 1; i < cykTable.length; i++){
            for(int j = 0; j < cykTable[i].length; j++){
                cykTable[i][j] = "";
            }
        }
        return cykTable;
    }

    public static String[][] doCyk(String[][] cykTable){
    //Step 1: Fill header row
        for(int i = 0; i < cykTable[0].length; i++){
            cykTable[0][i] = manageWord(word, i);
        }
    //Step 2: Get productions for terminals
        for(int i = 0; i < cykTable[1].length; i++){
            String[] validCombinations = checkIfProduces(new String[] {cykTable[0][i]});
            cykTable[1][i] = toString(validCombinations);
        }
        if(word.length() <= 1) { return cykTable; }
    //Step 3: Get productions for subwords with the length of 2
        for(int i = 0; i < cykTable[2].length; i++){
```

```
        return toArray(word)[position];
    }

    public static String[] checkIfProduces(String[] toCheck){
        ArrayList<String> storage = new ArrayList<>();
        for(String s : grammar.keySet()){
            for(String current : toCheck){
                if(grammar.get(s).contains(current)){
                    storage.add(s);
                }
            }
        }
        if(storage.size() == 0) { return new String[] {}; }
        return storage.toArray(new String[storage.size()]);
    }

    public static String[] getAllCombinations(String[] from, String[] to){
        int length = from.length * to.length;
        int counter = 0;
        String[] combinations = new String[length];
        if(length == 0){ return combinations; };
        for(int i = 0; i < from.length; i++){
            for(int j = 0; j < to.length; j++){
                combinations[counter] = from[i] + to[j];
                counter++;
            }
        }
        return combinations;
    }

    public static String toString(String[] input){
        return Arrays.toString(input).replaceAll("[\\[\\]\\,]", "");
    }

    public static String[] toArray(String input){
        return input.split("\\s");
    }

    public static Scanner openFile(String file){
        try{
            return new Scanner(new File(file));
        }catch(FileNotFoundException e){
            System.out.println("Error: Can't find or open the file: " + file + ".");
            System.exit(1);
            return null;
        }
    }
}
```

# RESULT:

String is accepted by the given context free grammar

```
Word: abbbabaa

G = ({a, b}, {S, A, B, E, C, X, Y, Z}, P, S)

With Productions P as:
A -> a | YE | XC
B -> b | XE | YZ
C -> AA
E -> YB | XA
S -> YB | XA | *
X -> b
Y -> a
Z -> BB

Applying CYK-Algorithm:

+-------+-------+-------+-------+-------+-------+-------+-------+
| a     | b     | b     | b     | a     | b     | a     | a     |
+-------+-------+-------+-------+-------+-------+-------+-------+
| A,Y   | B,X   | B,X   | B,X   | A,Y   | B,X   | A,Y   | A,Y   |
+-------+-------+-------+-------+-------+-------+-------+-------+
| E,S   | Z     | Z     | E,S   | E,S   | E,S   | C     |
+-------+-------+-------+-------+-------+-------+-------+
| B     | -     | B     | B     | A     | A     |
+-------+-------+-------+-------+-------+-------+
| Z     | Z     | Z     | E,S   | C     |
+-------+-------+-------+-------+-------+
| B     | -     | B     | A     |
+-------+-------+-------+-------+
| Z     | Z     | E,S   |
+-------+-------+-------+
| B     | B     |
+-------+-------+
| E,S   |
+-------+

The word "abbbabaa" is an element of the CFG G and can be derived from it.
```

String is not accepted by the given context free grammar

```
Word: aabbaa

G = ({a, b}, {S, A, B, E, C, X, Y, Z}, P, S)

With Productions P as:
A -> a | YE | XC
B -> b | XE | YZ
C -> AA
E -> YB | XA
S -> YB | XA | *
X -> b
Y -> a
Z -> BB

Applying CYK-Algorithm:

+-------+-------+-------+-------+-------+-------+
| a     | a     | b     | b     | a     | a     |
+-------+-------+-------+-------+-------+-------+
| A,Y   | A,Y   | B,X   | B,X   | A,Y   | A,Y   |
+-------+-------+-------+-------+-------+-------+
| C     | E,S   | Z     | E,S   | C     |
+-------+-------+-------+-------+-------+
| A     | B     | B     | A     |
+-------+-------+-------+-------+
| E,S   | E,S   | E,S   |
+-------+-------+-------+
| A     | A     |
+-------+-------+
| C     |
+-------+

The word "aabbaa" is not an element of the CFG G and can not be derived from it.
```