DOKUZ EYLUL UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

CME 3203 – THEORY OF COMPUTATION ASSIGNMENT REPORT

CONVERTING CFG to CHOMSKY NORMAL FORM

by

Ramazan Hakan Cankul

Ali Şiyar Arslan

December, 2022

İZMİR

Page

CONTENTS

CHAPTER ONE
PROJECT DESCRIPTION

In this assignment, it is requested to convert “CFG.txt” in Context free grammer form to Chomsky normal form.

A context free grammar (CFG) is in Chomsky Normal Form (CNF) if all production rules satisfy one of the following conditions:

- A non-terminal generating a terminal (e.g.; X->x)
- A non-terminal generating two non-terminals (e.g.; X->YZ)
- Start symbol generating ε. (e.g.; S-> ε)

So, to convert a context-free grammar (CFG) to Chomsky normal form, you can follow these steps:

Step 1: Eliminate null, unit and useless productions. If CFG contains null, unit or useless production rules, eliminate them.

Step 2: Eliminate terminals from RHS if they exist with other terminals or non-terminals. e.g,; production rule X->xY can be decomposed as:

X->ZY, Z->x

Step 3: Eliminate RHS with more than two non-terminals. e.g,; production rule X->XYZ can be decomposed as:

X->XP, P->YZ

CHAPTER TWO

PSEUDO CODE

```
❖ printTable()
        foreach (traverse lines)
            for (i traverse length of each lines)
                print element

❖ deleteElement take parameter as elementToDelete
        foreach (traverse all lines)
            foreach (traverse elements of one line)
                if(element = elementToDelete))
                    element.remove();
                    return lineName;
        return null;

❖ isContainElement take parameter as elementToFind
        for (traverse all lines)
            foreach(traverse element of one line)
                if(element = elementToFind))
                    return true;
        return false;

❖ addEpsilonAndCreateVariation take parameter as elementToCheck
        foreach (traverse for all lines)
            for (k traverse length of each line)
                if(element = elementToCheck))
                    add "€" to enf of the line
                if(element contains elementToCheck))
                    let n integer variable for number of nullable value inside element
                    for (i traverse length of element)
                        if(element.charAt(i) = elementToCheck )
                            n++
                    let flag boolean variable which check in how many steps the value to
                    be added to an array will change
```

```
let cnt integer variable for count of  how many steps the value to be
added to an array will change ex: last digit every step, second to last
every two steps

let numberOfFromTheEnd variable for count of nullable value from
the end

let allCombinations that is string array which length 2^n

for (i traverse length of allCombinations)

    allCombinations[i] = ""

for (i traverse length of element)

    cnt=0

    for (j traverse length of allCombinations)

        if( (flag = true) and
        element.charAt(element.length-i-1) =
        elementToCheck))

            allCombinations[j]+=
            element.charAt(element.length-i-1)

            cnt++

            if(cnt == 2^numberOfFromTheEnd)

                flag=!flag

                cnt=0

        else if((flag=false) &&
        (element.charAt(element.length-i-1) =
        elementToCheck)))

            allCombinations[j]+=""

            cnt++

            if(cnt == 2^numberOfFromTheEnd))

                flag=!flag;

                cnt=0;

        else

            allCombinations[j]+=
            element.charAt(element.length-i-1)))

    if(element.charAt(element.length-i-1) = elementToCheck))

        numberOfFromTheEnd++;

for (i traverse length of allCombinations)

    let str string variable which is equal to allCombinations[i]

    let nstr string variable which is equal "" -> (null)

    for (j traverse length of str)

  char ch = str.charAt(j)

  nstr = ch + nstr;

    if(i is not equal to 0)
        line.addElement(nstr);
```

❖ **removeUnitProduction**

```
    foreach (traverse for all lines)
        create a list for elements at the line (name as lineElements)
        for (k traverse lineElements size)
            let element variable as string equal to lineElements(k)
            if(line.getName = element))
                remove element from the lineElements
            else if(length of element = 1) and (present alphabet doesn't contain element))
                remove element from the lineElements
                let temp_k is temporary variable
                for (i traverse length of line)
                    let lineName2 string variable which is equal to ith element
                    of the present lines list
                    if(lineName2 = element)
                        let lineSize integer variable which is equal to length
                        of the ith element of the present lines list
                        for (j traverse in lineSize)
                            add element to lineElement
                            temp_k++
```

❖ **eliminateMoreThanTwoNonTerminal**

```
    for (i traverse list of lines)
        for (j traverse length of line)
            let element string variable which is equal to jth element of the lineElements
            let countNonTerminalValues integer variable which is equal to 0
            let toChange string variable which is equal to "" (null)
            for (k traverse length of element)
                if(present alphabet contains kth character of element)
                    countNonTerminalValues++
                if(countNonTerminalValues >= 2)
                    add kth character of element to toChange
            if(length of toChange >= 2)
                let lineSize2 integer value which is equal to size of list of lines
                for (k traverse length of lineSize2)
                    create list name as lineElements2 which elements of kth
                    line
```

```
                        for (l traverse in lineElements2)
                            let element2 string variable which is equal to lth
                            element of lineElements2
                            if(element2 ends with toChange and element2 is
                            equal to toChange)
                                remove element2 from lineElements2
                                add element to lineElements2
                    add element to list of lines
```

❖ **eliminateTerminals**

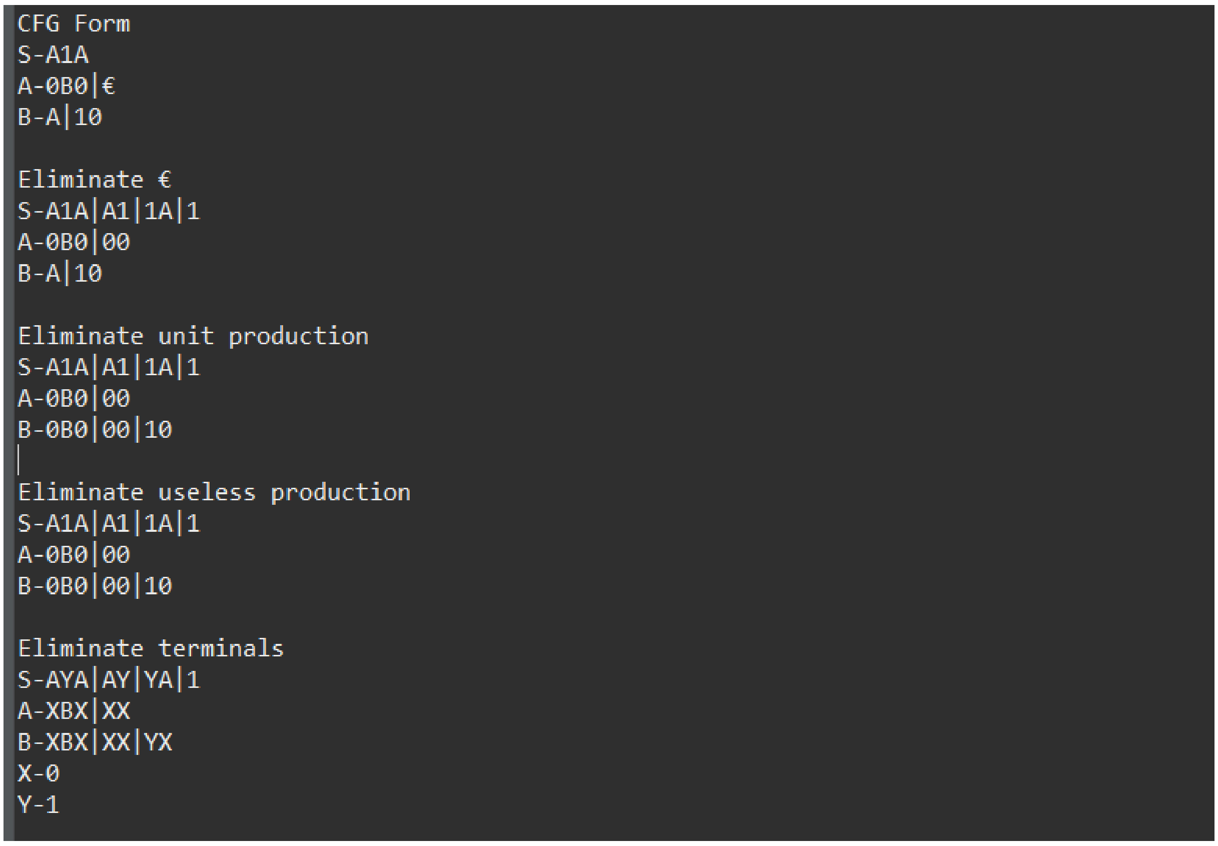
```
    foreach (traverse all alphabet)
        let isTerminalNearNonTerminal boolean variable which is false
        for (traverse all line)
            create list name as lineElements and fill with element which is in the line
            for (i traverse in lineElements)
                let element which is equal to ith element of lineElements
                if((element contains terminal) and (element is equal to terminal))
                    let newElement string variable which is replace element 0th
                    character of terminal and 0th character of 0th element of
                    newLineNames
                    remove ith element from lineElements
                    add element to lineElements
                    isTerminalNearNonTerminal=true
        if(isTerminalNearNonTerminal = true)
            add element to list of lines
```

❖ **removeUselessProduction**

```
    for (i traverse all lines)
        let Line line variable which is equal to ith element of list of line
        let lineName string variable which is equal to name of line
        let isReachable boolean variable which is false
        foreach (traverse list of lines)
            create a list of elements in a row
            foreach (traverse in lineElements)
                if(element contains lineName)
                    isReachable = true
                    break
            if(isReachable = true)
                break;
        if ((isReachable = false) and lineName is not equal to "S")
            remove line from list of lines
```

CHAPTER THREE

SCREENSHOTS OF THE PROGRAM



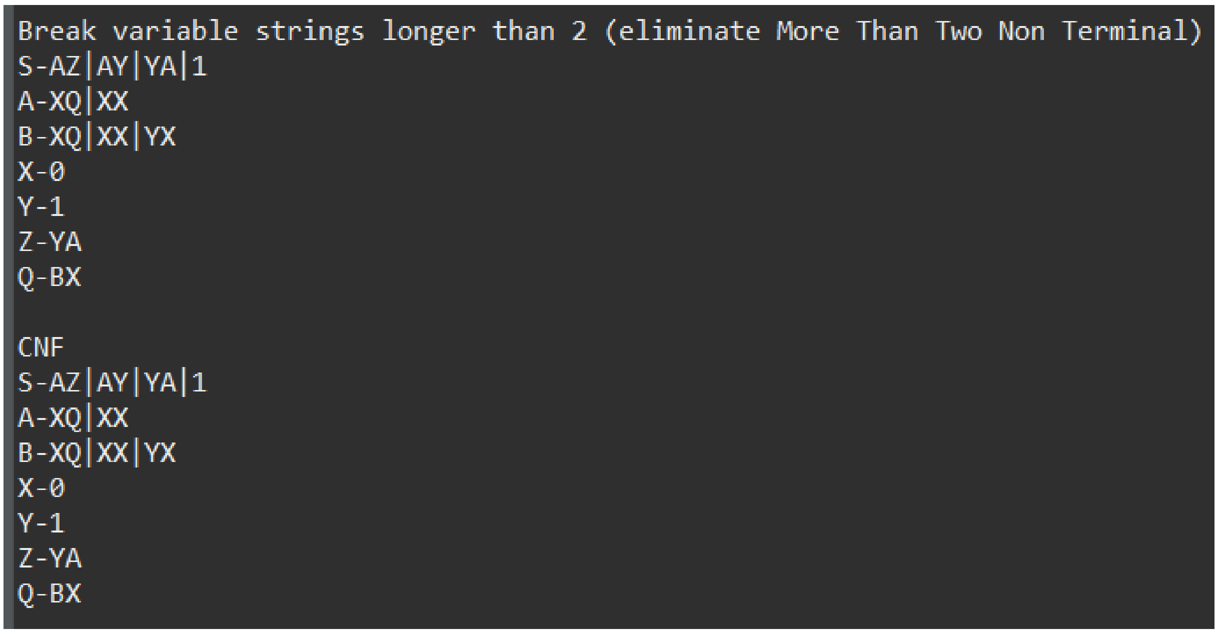
```
CFG Form
S-A1A
A-0B0|€
B-A|10

Eliminate €
S-A1A|A1|1A|1
A-0B0|00
B-A|10

Eliminate unit production
S-A1A|A1|1A|1
A-0B0|00
B-0B0|00|10

Eliminate useless production
S-A1A|A1|1A|1
A-0B0|00
B-0B0|00|10

Eliminate terminals
S-AYA|AY|YA|1
A-XBX|XX
B-XBX|XX|YX
X-0
Y-1
```



```
Break variable strings longer than 2 (eliminate More Than Two Non Terminal)
S-AZ|AY|YA|1
A-XQ|XX
B-XQ|XX|YX
X-0
Y-1
Z-YA
Q-BX

CNF
S-AZ|AY|YA|1
A-XQ|XX
B-XQ|XX|YX
X-0
Y-1
Z-YA
Q-BX
```

REFERENCES

1. Chomsky, Noam (1959). "On Certain Formal Properties of Grammars". Information and Control. 2 (2): 137–167. doi:10.1016/S0019-9958(59)90362-6. Here: Sect.6, p.152ff.
2. https://www.javatpoint.com/automata-chomskys-normal-form
3. https://www.tutorialspoint.com/convert-the-given-context-free-grammar-to-cnf
4. https://www.geeksforgeeks.org/converting-context-free-grammar-chomsky-normal-form/