CS 410 – Project 2 Report

Context Free Grammar to Chomsky Normal Form converter

By: Esrah Zahid (S020289)

Table of Contents

I. Introduction:

This report explains my design and implementation of a program code that converts a Context Free Grammar (CFG) to its equivalent Chomsky Normal Form (CNF). In Automata theory, a Context Free Grammar is a formal grammar which is used to generate all possible strings in each formal language called Context Free Languages.

We convert a CFG to a simplified form to reduce ambiguity and for convenience purposes and one of the simplest and most useful forms of doing this is converting to its equivalent CNF.

In this project, I will attempt to convert CFG to CNF using the Java programming language. The input file will always have the same format which is:

```
NON-TERMINAL
S
F
TERMINAL
0
1
RULES
S:00S
S:11F
F:00F
F:e
START
S
```

File: G1.txt

Every file consists of “NON-TERMINAL”, “TERMINAL”, “RULES” and “START”.

The rules imply the following:
S:00S
S:11F
This means, means S->00S|11F

II. Method:

I will begin by first reading the input file which is stored in the same directory as *CFGtoCNF.java*. The format of the text file will remain the same. I will use *java.util.Scanner* import to read the input text files and find the indexes of “TERMINAL”, “RULES” and “START”. This will help me later jump to a particular index if I need to using *java.nio.file.*

Next, I created 8 of the following methods to handle each step in converting a CFG to CNF:

```
public void convertCFGtoCNF() throws IOException {
    insertNewStartSymbol();          //Step 1. Eliminate start symbol from RHS.
    convertStringToMap();
    eliminateNullProductions();      //Step 2a. Eliminate null productions.
    removeDuplicateKeyValue();
    eliminateUnitProductions();      //Step 2b. Eliminate Unit productions
    onlyTwoTerminalandOneVariable(); //Step 3. Assign new variable for two non-terminal or one terminal
    eliminateThreeTerminal();        //Step 4. Eliminate RHS with more than two non-terminals.
    outputResult();                  //Print the output to console
}
```

I will be explaining the functions in detail in the next section. Each method is a step-by-step process for converting to Chomsky normal form.

III. Implementation:

I will begin by storing the Strings of the CFG to a List after reading the input file

```
List<String> str = new ArrayList<>();
```

Example of what the contents of the List will look like after reading them from the file:
[S:a|aA|B, A:aBB|e, B:Aa|b]

To store the rule contents of the input file I will be making use of a Linked HashMap data structure.

```
final Map<String, List<String>> variableToProductionMap = new LinkedHashMap<>();
```

I chose this option because it will easily help me map variable with production after I read them from the List. (Eg: S->a|aA|B).

To store items in VariableToProductionMap I can easily use:

```
variableToProductionMap.put(variable, productionList);
```

Next, I will follow step by step procedure to convert from CFG to CNF:

1. Step 1) Insert New Start Symbol:
   In our case, the new start symbol will always be S0.
   ```
   private void insertNewStartSymbol() {
       String newStart = "S0";
       List<String> newProduction = new ArrayList<>();
       newProduction.add("S");
       variableToProductionMap.put(newStart, newProduction);
   }
   ```
2. Step 2) Eliminate any null or unit productions in our CFG.

In our program a null production (epsilon) is represented as e. We will iterate over the contents of the List to find any occurrences of e and replace it with empty string.

Unit Productions occur when one non-terminal gives another non-terminal. To remove them we will again be iterating and checking for any occurrences of unit productions.
For example:
A → aAS|aS|a
B → SbS| A|bb
Where B-> A is a unit production will become:
B → SbS|bb|aAS|aS|a

3. Step 3) In this step I will try to remove the terminals if they exist with other terminals or variables and replace it with a new "terminal"

```
int asciiBegin = 70; //F
```

The new terminal always starts with the "F" and will get incremented to the next letter if any of the previous terminal gets decomposed.
An example of such occurrence is:
A -> aB
Here my program will detect and decompose it to:
A -> FB
F -> a
This is done by the following way:

```
Map<String, List<String>> tempList = new LinkedHashMap<>();
```

```
if (found) {
    ArrayList<String> newVariable = new ArrayList<>();
    newVariable.add(newProduction);
    key = Character.toString((char) asciiBegin);
    tempList.put(key, newVariable);
    asciiBegin++;
}
```

4. Step 4) Using the same concept as above, I will try to remove any occurrence of more than two non-terminals and replace it with a newly generated terminal.
An example of such occurrence is:
A -> ABC
Here my program will detect and decompose it to:
A -> FC
F -> AB

5. Finally, I will output the result to console using printMap() function that I have created:

```
private void printMap() {
    for (Map.Entry<String, List<String>> stringListEntry : variableToProductionMap.entrySet()) {
        System.out.println(stringListEntry.getKey() + ":" + stringListEntry.getValue());
    }
}
```

IV. Results:

The resulting CNF will be printed to the console in the similar format as the input. There will be 4 headers, namely, “NON-TERMINAL”, “TERMINAL”, “RULES” and “START”

```
private void outputResult() throws IOException {
    System.out.println("NON-TERMINAL");
    for (Object key : variableToProductionMap.keySet()){
        System.out.println(key);
    }
    System.out.println("TERMINAL");
    Path path = Paths.get(filePath);
    for (int i = terminalIndex; i < ruleIndex - 1; i++){
        System.out.println(Files.readAllLines(path).get(i));
    }
    System.out.println("RULES");
    printMap();
    System.out.println("START");
    System.out.println("S");
}
```

For example. Running the program code on G1 and G2 will output the following, respectively:

```
NON-TERMINAL
S0
S
F
G
H
I
J
TERMINAL
0
1
RULES
S0:[JF, HG, HH]
S:[JF, HG, HH]
F:[JS]
G:[HF]
H:[1]
I:[JF]
J:[0]
START
S
```

```
NON-TERMINAL
S0
S
A
B
F
G
TERMINAL
a
b
RULES
S0:[a, FA, AF, b, a]
S:[a, FA, AF, b, a]
A:[aH]
B:[AF, b, a]
F:[a]
G:[H]
START
S
```

(*) Here the rule S0:[JF, HG, HH] means S0 -> JF|HG|HH