**CONVERTER CFG TO CNF IN JAVA**

**Prepared by Musa Berkay Kocabaşoğlu 018431**

**1- Introduction**

A formal grammar called context free grammar is used to produce every conceivable string in a given formal language. Four tuples can be used to build context-free grammar as follows: G= (V, T, P, S)

* G describes the grammar
* T describes a finite set of terminal symbols
* V describes a finite set of non-terminal symbols
* P describes a set of production rules
* S is the start symbol

CNF stands for Chomsky normal form. A CFG(context free grammar) is in CNF(Chomsky normal form) if all production rules satisfy one of the following conditions:

* Start symbol generating ε. For example, S → ε
* A non-terminal generating two non-terminals. For example, A → BC
* A non-terminal generating a terminal. For example, B → a

Input files contains all the crucial details regarding CFG in a clear and concise manner. In order to complete this project, I read input files containing information about CFG, convert that information to the CNF, and then write the information from the CNF to the output console in a manner similar to the input files. I implement a Java program that converts CFGs to CNFs in my project.

**2- Methodology**

The steps of production rule that will be used in CNF is shown below. Epsilon, the null value, is represented by ε. Let’s convert with example rules:

* S → a | aA | B
* A → aBB | ε
* B → Aa | b

Step1: We creating a new production S1 → S, as the start symbol S appears on the RHS. The grammar will be:

* S1 → S
* S → a | aA | B
* A → aBB | ε
* B → Aa | b

Step2: We eliminating epsilon productions. As grammar contains A → ε null production, its removal from the grammar yields:

* S1 → S
* S → a | aA | B
* A → aBB
* B → Aa | b | a

Step3: We removing unit and useless productions which are S1 → S  and S → B productions.

* S0 → a | aA | Aa | b
* S → a | aA | Aa | b
* A → aBB
* B → Aa | b | a

Step4: If RHS productions with more than two elements, we need to split into two or more productions by adding non-terminal symbols.

* S0 → a | aA | Aa | b
* S → a | aA | Aa | b
* A → CB
* B → Aa | b | a
* C → aB

Step5: We replacing terminals with non-terminal symbols.

* S0 → D | DA | AD | E
* S → D | DA | AD | E
* A → DB
* B → AD | E | D
* C → DB
* D → a
* E → b

**3- Implementation**

In the beginning, I will import these built-in functions which are java.io.File, java.util.Scanner, java.io.FileNotFoundException for file loading, handling exceptions, and java.util.HashMap, java.util.ArrayList, java.util.Arrays, java.util.HashMap, java.util.Iterator, java.util.LinkedHashMap, java.utilList, java.util.Map, java.util.Set for necessary data structures. I create 3 different general method for loading the CFG grammar, converting CFG to CNF, and printing CNF to the console.

I construct these variables with these type for loading grammar:

* input: Data type is String. It stores the input obtained by reading the input file.
* lineCount: Data type is int. It stores the number of different production rules going out from non-terminal node.
* foundEpsilon: Data type is String. It stores the non-terminal which contains epsilon value.
* CNF\_map -> Data type is LinkedHashMap. Data type of key is String. Data type of value is String List. Key represents the non-terminals. Value represents the variables or terminals that are reachable from this key.
* CFG\_map -> Data type is HashMap. Key and value data are of the String data type. Values represent the variables that can be accessed from the non-terminals represented by the key.

Text

Description automatically generatedI read the file with load(String filename) function and I load the data step by step with while loop and scanner. I use condition for stopping while loop and pass loading a new variable. For doing this, I use String variable named current. In the RULES part I started to fill the CFG\_map variable. CFG\_map variable hold production rules like this in G1 text:

{ “S”: “→ 00S|11F“, “F”: “→ 00F|e” }

After loading the info my variables, I pass to the conversion function. In this function, I have 7 crucial part. Firstly, I’m creating new start symbol which name is S0 and add this to the CNF\_map variable. After that, I transfering to CNF\_map productions with input variable.

**Figure 1 – load Function**

Text

Description automatically generatedText

Description automatically generated

**Figure 2 – Adding new start symbol**

Text

Description automatically generatedEach terminal or non-terminal on the CNF\_map represents the right side, and each key on the CNF\_map represents the left. The right-hand side's epsilon values are removed using the removeEpsilon method with for loop. Then, duplicate keys are removed from the left. If there is only one variable or terminal at the right, the removeSingleVariable method eliminates the rules. onlyTwoNonTerminalOrOneTerminal method checks the right side and has only two non-terminals or terminal. Two non-terminals or one terminal value should be present on the right side. Otherwise, this approach sets up the rules in the appropriate way. The method removeThreeOrMoreTerminals searches for variables and terminals that have more than three terminals in order to remove them.

**Figure 3 – Initial Transfer to CNF**

Finally, I convert the rules of CFG with a converter. So, using the provided input text files, I use the print CNF() function to print our new nonterminals, terminals, start state, and rules in the same manner. I use a for loop to print variables, with the exception of start state, which only has one value. For printing terminals I look CNF\_map values which are one index symbols. I display all of the new rules using the CNF map variable since the conversion resulted in new non-terminal, rules and start state.

**Figure 4 – printCNF Function**

**4- Results**

**A picture containing text

Description automatically generatedA picture containing graphical user interface

Description automatically generated**The images below show the output produced after the run of the main method. This outputs show CNF1 and CNF2, the converted version of the CFGs in the G1 and G2 files.

**Figure 6 – CNF of G2.txt**

**Figure 5 – CNF of G1.txt**