Introduction

The problem in this assignment is to create a software that takes a Context-free Grammar (CFG) as an input and convert it into Chomsky Normal Form (CNF).

Context free grammar is a formal grammar which is used to generate all possible strings in a given formal language. Chomsky Normal Form is a different way to express the CFG.

The input is given as a text file with the format below:

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

S -> 00S | 11F
F -> 00F | epsilon

To represent the CFG (right side), a text file (left side) is given. Software shall read the content of the file and write CNF with the same format in the console.

Method

Let's represent the CFG as G = (V, T, R, S) where;

- V: Variables (Nonterminals)
- T: Terminals
- R: Production rules
- S: Start state

We want to create the Chomsky Normal Form of this CFG. In Chomsky Normal Form, there are three allowed rule types:

1. A->BC
2. A->b
3. S->epsilon is allowed while A->epsilon is not

The workflow of the algorithm that converts CFG to CNF is;

1. Add new start variable (S0)
2. Connect S to S0
3. Eliminate the rules of the form A->e (e represents epsilon)
4. Eliminate the rules of the form A->B
5. Create new rules as substitutions to provide the first two allowed rule types of the CNF

Implementation

Java is used in the project's implementation part. To keep the CFG in a structured way in Java, "Map" data structure is used with a String as key, and a list of Strings as value. The key represents the left hand side of the production rule of the CFG, and the value represents the right hand side.

Figure 1: CFG Example

For the given CFG example in *Figure 1*, the Map structure is:

CFG = {"S": ["00S", "11F"],

"F": ["00F", "e"]

}

Since the ε (epsilon) sign is represented with "e" letter in the input file, and this is specially not used for any other terminal, I left it as "e" in the Map structure.

The program has a single class with the following methods:

- getCFGRules(): Reads the given text file and creates the Map structure of the CFG.
- setParameters(): Sets the rule parameters of the class.
- CFGtoCNF(): Adds new start variable (S0) to Map, and calls the necessary methods by order to complete the CNF transformation on the Map.
- eliminator(): Calls either the "removeSingleVariable()" or the "removeThreeTerminal()" method depending on its input. This method provides the first two rule of the CNF mentioned in the Method section.
- splitEnter(): Splits the given input string by lines.
- stringtoMap(): It uses "splitEnter()" method to split strings and put to Map as CNF rule.
- removeEpsilon(): It removes the epsilon rules from the Map to provide 3rd step of the transformation mentioned in the Method section. Since I got Concurrent Modification Exception during the development, I used iterators to avoid this errors.

- removeSingleVariable(): It removes the rules consisting of single variables to provide the 4th step of the transformation. To achieve this, it puts the rule of that single variable instead the variable itself to the base variable's rule set.
- checkDuplicateInRuleList(): In CNF, there cannot be duplicate rules for in a variable's rule set. This method removes the duplicates and make sure that all rules are unique.
- onlyTwoTerminalandOneVariable(): Makes sure that there are only "one variable" or "two terminals" in the rule sets. This method also provides the first two rules of the CNF given in the Method section. To do this, it creates substitution variables to decrease the length of the rule if it is more than two variables, or more than one terminal.
- removeThreeTerminal(): If there are three terminals, it changes two of them with the substited variable created by "onlyTwoTerminalandOneVariable()" method to decrease the length of the rule from three to two to provide A->BC feature.
- printResult(): It prints the CNF with style of the input text.

The program first calls the "getCFGRules()" method to read the CFG, and then calls "setParameters()" method to get the class ready for the transformation. After that, it converts the given CFG to CNF by calling "CFGtoCNG()" method, and then prints the result by calling "printResult()" method.

Results

Below shows the input G1.txt (left side) content and the output CNF content (right side) created by the software.

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

```
NON-TERMINAL
S0
S
F
G
H
I
J
K
TERMINAL
1
0
RULES
S0:KG
S0:IH
S0:II
S:KG
S:IH
S:II
F:KJ
F:KK
G:KS
H:IF
I:1
J:KF
K:0
START
S0
```

Below shows the input G1.txt (left side) content and the output CNF content (right side) created by the software.

```
NON-TERMINAL
S
A
B
TERMINAL
a
b
RULES
S:a
S:aA
S:B
A:aBB
A:e
B:Aa
B:b
START
S
```

```
NON-TERMINAL
S0
A
B
S
G
H
I
TERMINAL
a
b
RULES
S0:G
S0:GA
S0:AG
S0:H
A:aBB
B:AG
B:H
B:G
S:G
S:GA
S:AG
S:H
G:a
H:b
I:BB
START
S0
```

Atahan Çaldır

S018194