Timothy MacNary
Alexandra Warlen
Final Project - Theory of Computation
CFG -> CNF Converter

Running the Program

To compile this program, download and unzip the zip file. Import the project to your favorite Java IDE. Compile the program and run it. When prompted, first enter the number of lines in your CFG. Then proceed by entering each nonterminal, terminal or combination in that line of the CFG in the order that it appears in the line. Follow each entry by pressing the 'enter' key. At the end of each line, type 'done' to signify that you are done entering that line of your CFG. After you have completed the number of lines signified by your first answer, the code will run and print the CFG that you have entered, followed by either a message letting you know that your code is already in proper CNF format, or it will print out the new CNF version of the CFG. Also keep in mind that the '$' represents the epsilon character for this program. So if you would like to add an epsilon, replace its existence with '$' when entering it into your CFG.

Project Design

Overall, the design of the project stayed mostly consistent with the initial proposal, however we did have to add some additional helper methods to complete each task more efficiently as well as a few extra global variables to keep track of some data.

The most noticeable change was the use of mostly ArrayLists rather than regular arrays. We chose to switch to this data structure specifically for its built in functions and the ease of adding more elements dynamically. This changed the main data structure of our program which went from a 2 dimensional array of String in the proposal to a 2 dimensional ArrayList of String in the final product.

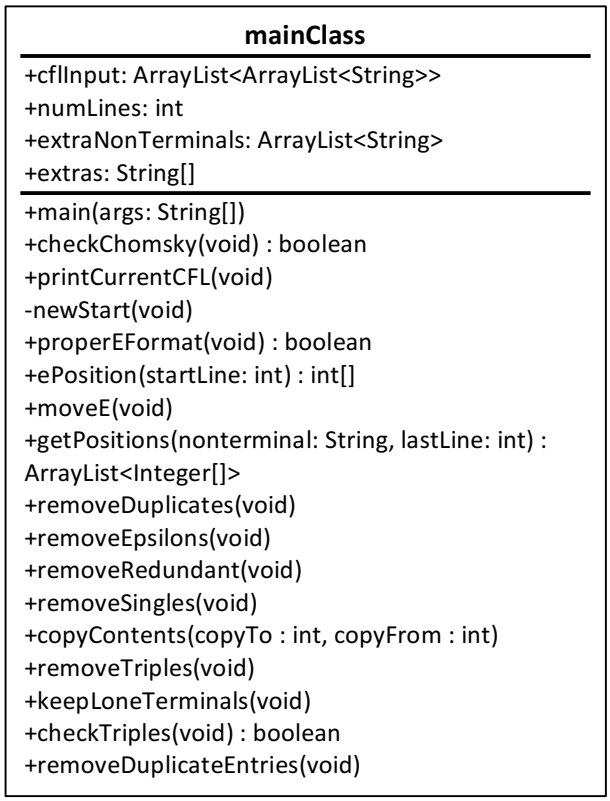
This data structure can be visualized using the following example:
CFG:
A -> bA | B
B -> a | b | epsilon ($)

This CFG would look like this when stored in our program:

	0	1	2	3
0	**A**	**bA**	**B**	
1	**B**	**a**	**b**	**$**

Timothy MacNary
Alexandra Warlen
Final Project - Theory of Computation
CFG -> CNF Converter

The new organization of the program can be observed by this UML diagram of the setup of the project. Some of the names of the original methods were changed and many helper methods were added to make the code easier to understand and more simplified.



Results of Testing

This program does have some limitations, the sample output file shows working examples as this code does work properly for most test cases. However, in order for the program to work properly, the new non-terminals created must be single uppercase letters, so we have a list of these letters that we will use from P to Z that are the new non-terminals. This means that these letters should not be used for the initial non-terminals as it may cause some confusion with the result.

Evidence of Testing and Test Cases

To view some test cases of this program's output, please navigate to the three screenshots of the demo_output png files located in the zip file of the project. These outputs of the program first show the given CFG, then below, the output of the CNF in proper format is printed after the program creates it. You may also view these test cases in action by following this link: https://uportland.mediaspace.kaltura.com/media/CFG+to+CNF+Demo/1_iazf279a

Timothy MacNary
Alexandra Warlen
Final Project - Theory of Computation
CFG -> CNF Converter

Teammate Contributions

As this project was very involved and each step of the CFG manipulation was very precise, it was necessary to code the entire project with pair programming. Although each method was unique and could have been coded separately, keeping the entire project organized as a whole and keeping track of what needed to happen next was extremely important and quite difficult. Due to the complicated nature of the program with many very detailed steps and many parts relying on one another, we decided that it would be most successful if we worked together to code every portion of the program. We switched turns typing, but each idea and implementation was decided on as a team and each portion of the code was created with an equal contribution from both team members.