**Chomsky Normal Form**

**Course: Formal Languages & Finite Automata**

**Author: Vladlen Mîțu**

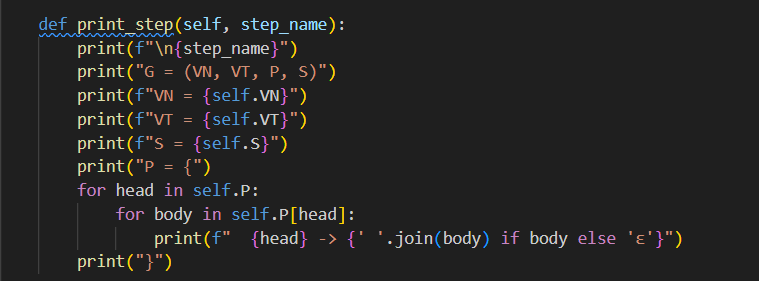
**Theory:**

**Chomsky Normal Form (CNF)** is a way of simplifying context-free grammars. In CNF, each rule must be either of the form A → BC (two non-terminals) or A → a (a single terminal). The only exception is the start symbol, which can go to ε (empty string). CNF is useful for parsing and algorithmic analysis, especially in computer science. To convert a grammar to CNF, we eliminate ε-productions, unit rules, and unreachable or useless symbols, then rewrite rules to fit the CNF structure.

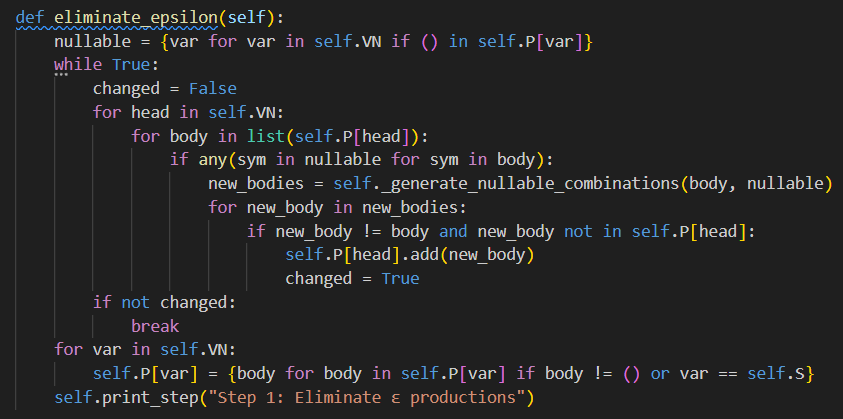
**Objectives:**

1. Learn about Chomsky Normal Form (CNF) [1].
2. Get familiar with the approaches of normalizing a grammar.
3. Implement a method for normalizing an input grammar by the rules of CNF.
   1. The implementation needs to be encapsulated in a method with an appropriate signature (also ideally in an appropriate class/type).
   2. The implemented functionality needs executed and tested.
   3. Also, another **BONUS point** would be given if the student will make the aforementioned function to accept any grammar, not only the one from the student's variant.

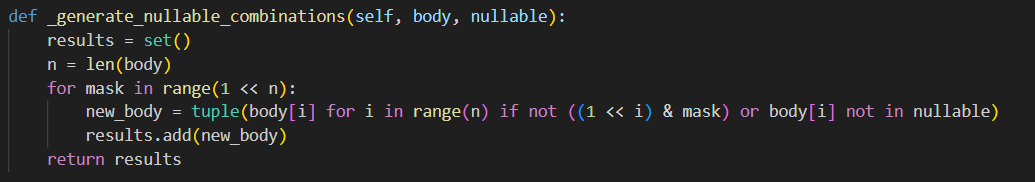
**Implementation description:**



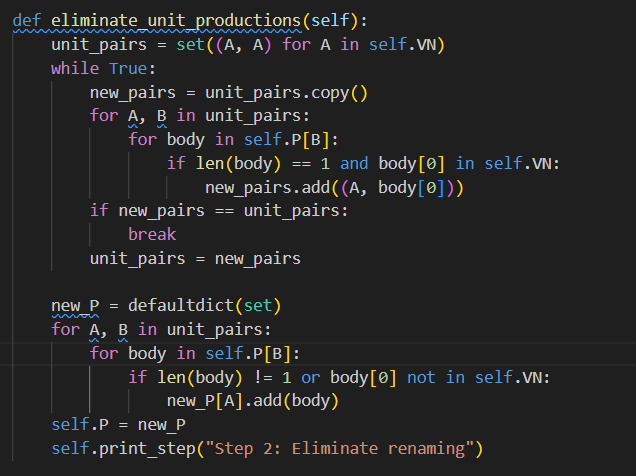
The print\_step method is used to display the current state of the grammar at each step of the conversion process. It prints the grammar components (VN, VT, P, S) in a readable format, including all productions. This helps track changes made during each transformation step.



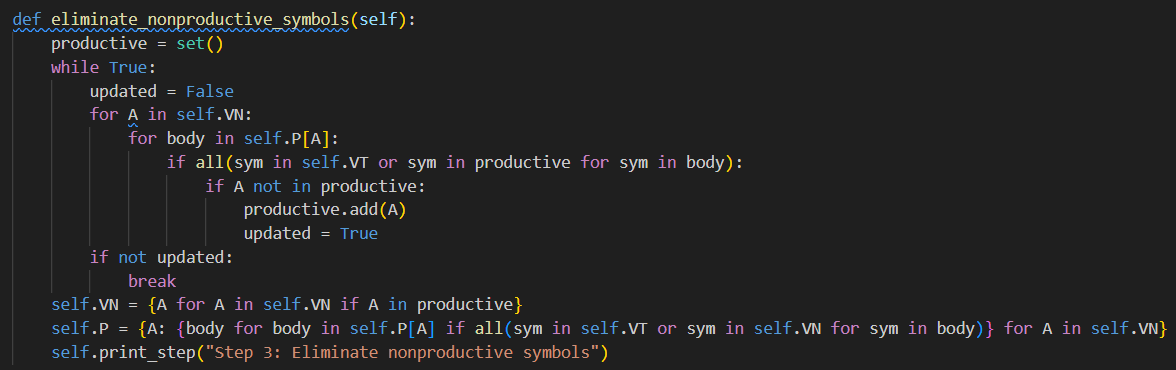
The eliminate\_epsilon method removes ε-productions (nullable productions) from the grammar while preserving its language. It identifies nullable variables, generates all possible combinations of productions without ε, and updates the grammar accordingly.



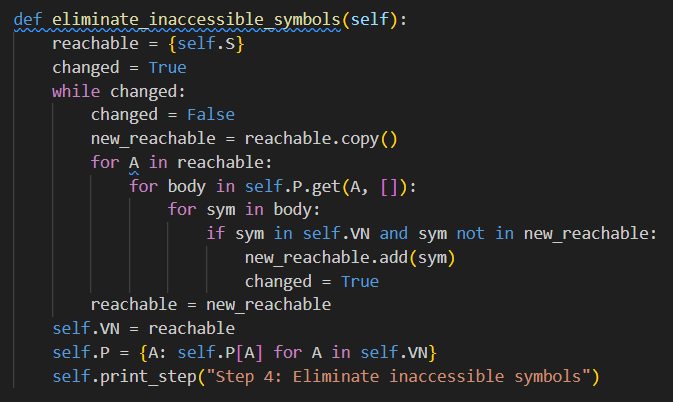
The generate\_nullable\_combinations method generates all possible combinations of a production body by removing nullable variables (variables that can produce ε). It uses bitmasking to systematically include or exclude nullable symbols, ensuring all valid combinations are created.



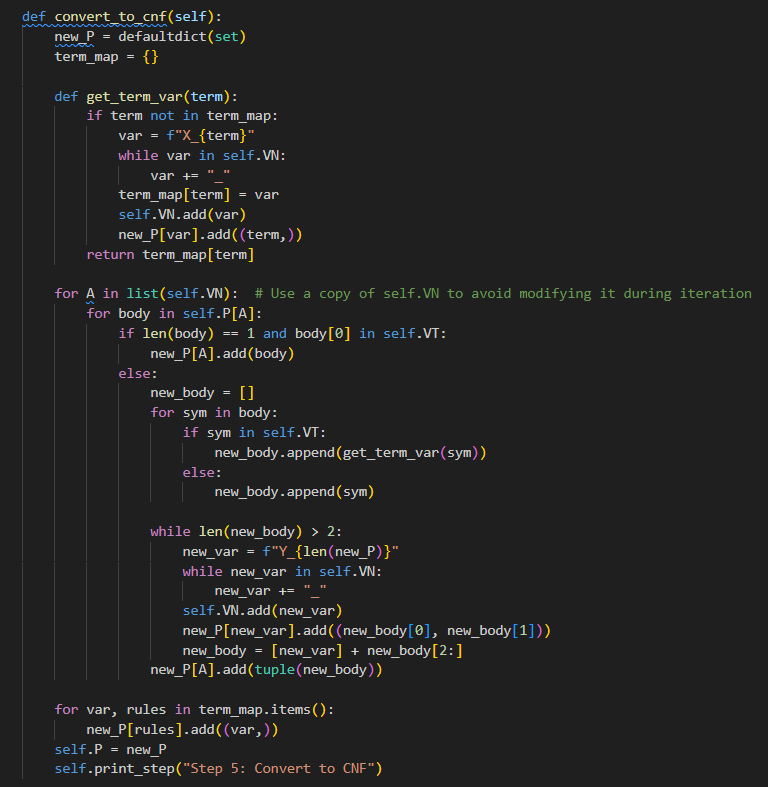
The eliminate\_unit\_productions method removes unit productions (e.g., A → B) from the grammar. It identifies all unit pairs (variables that can derive each other) and replaces unit productions with the non-unit productions of the corresponding variables, ensuring the grammar remains equivalent.



The eliminate\_nonproductive\_symbols method removes nonproductive variables (those that cannot derive terminal strings). It identifies productive variables iteratively and updates the grammar to include only productions involving these variables.



The eliminate\_inaccessible\_symbols method removes variables that cannot be reached from the start symbol. It iteratively identifies all reachable variables and updates the grammar to include only those variables and their associated productions.



The convert\_to\_cnf method transforms the grammar into Chomsky Normal Form (CNF). It replaces terminals in mixed productions with new variables, breaks long productions into binary productions, and ensures all rules conform to CNF requirements.

**Conclusion:**

This code implements a CFGtoCNFConverter class that systematically converts a context-free grammar (CFG) into Chomsky Normal Form (CNF). It performs key transformations, including eliminating ε-productions, unit productions, nonproductive symbols, and inaccessible symbols, and ensures all productions conform to CNF rules. The process is modular, with each step clearly defined and tested, making the implementation reusable for any grammar.

**References:**

1. Else Course FAF.LFA21.1
2. Automata Theory, Languages, and Computation: https://www-2.dc.uba.ar/staff/becher/Hopcroft-Motwani-Ullman-2001.pdf
3. Theory of Computation: https://www.geeksforgeeks.org/introduction-of-theory-of-computation/