**MINISTRY OF EDUCATION AND RESEARCH OF REPUBLIC OF MOLDOVA TECHNICAL UNIVERSITY OF MOLDOVA FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS**

**Laboratory work 5:**

# Topic: Chomsky Normal Form

**Course: Formal Languages & Finite Automata**

**Author:Ostafi Eugen, FAF-222**

**Chișinău, 2024**

**THEORY**

Chomsky Normal Form (CNF) is a standard form used to represent context-free grammars (CFGs), named after the linguist Noam Chomsky. In CNF, each production rule of the grammar is restricted to one of two specific forms: either a non-terminal symbol expanding into exactly two non-terminal symbols, or a non-terminal symbol expanding into a single terminal symbol. This structured format provides several advantages in formal language theory and computational linguistics.

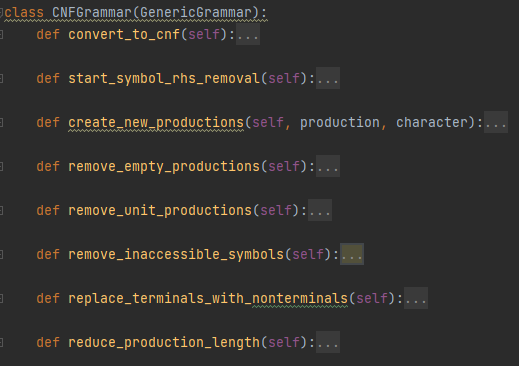
One of the primary benefits of CNF is its simplicity. By imposing strict constraints on the production rules, CNF simplifies the parsing process. Algorithms for parsing CNF grammars are often more efficient and straightforward compared to parsing unrestricted CFGs. This simplicity makes CNF particularly useful in applications where computational efficiency is crucial, such as compiler design and natural language processing.

**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. A **BONUS point** will be given for the student who will have unit tests that validate the functionality of the project.
   4. 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**

1. **context-free grammar to Chomsky Normal Form**

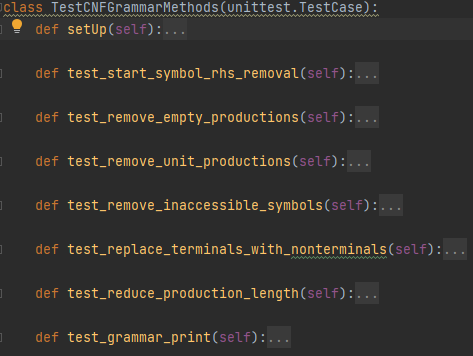


This class, **CNFGrammar**, is designed to operate on a context-free grammar (CFG) and convert it into Chomsky Normal Form (CNF). Here's a brief explanation of each method:

1. **convert\_to\_cnf():**
   * Calls a series of methods in a specific order to transform the given CFG into CNF.
2. **start\_symbol\_rhs\_removal():**
   * Ensures that the start symbol does not appear on the right-hand side of any production. If it does, it introduces a new start symbol and adds a production rule for it.
3. **create\_new\_productions(production, character):**
   * Generates new productions by removing occurrences of a specified character from a given production.
4. **remove\_empty\_productions():**
   * Removes ε (empty string) productions from the grammar, updating the productions accordingly to maintain language equivalence.
5. **remove\_unit\_productions():**
   * Removes unit productions, where a non-terminal symbol directly produces another non-terminal symbol. It iteratively eliminates unit productions until none are left.
6. **remove\_inaccessible\_symbols():**
   * Removes symbols (both non-terminals and their corresponding productions) that are inaccessible from the start symbol.
7. **replace\_terminals\_with\_nonterminals():**
   * Replaces terminal symbols with new non-terminal symbols, ensuring that each terminal symbol is only produced by a single non-terminal symbol.
8. **reduce\_production\_length():**
   * Reduces the length of productions to at most two symbols by introducing new non-terminal symbols for longer productions.

These methods work together to systematically transform the input CFG into CNF, ensuring that the resulting grammar meets the strict requirements of CNF while preserving the language generated by the original CFG.

1. **Testing**

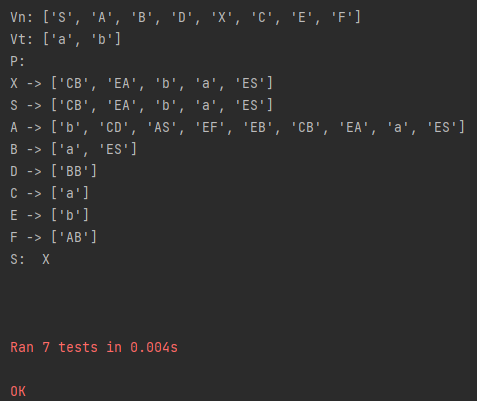


The class **TestCNFGrammarMethods** is a unit test class designed to test the methods of the **CNFGrammar** class. Here's a brief explanation of each test method:

1. **setUp(self):**
   * Initializes a **CNFGrammar** object with a predefined CFG for testing purposes.
2. **test\_start\_symbol\_rhs\_removal(self):**
   * Tests whether the **start\_symbol\_rhs\_removal()** method successfully removes the start symbol from the right-hand side of any production rule.
3. **test\_remove\_empty\_productions(self):**
   * Tests whether the **remove\_empty\_productions()** method correctly removes ε (empty string) productions from the grammar.
4. **test\_remove\_unit\_productions(self):**
   * Tests whether the **remove\_unit\_productions()** method effectively eliminates unit productions, ensuring that no non-terminal symbol directly produces another non-terminal symbol.
5. **test\_remove\_inaccessible\_symbols(self):**
   * Tests whether the **remove\_inaccessible\_symbols()** method correctly removes inaccessible symbols (both non-terminals and their corresponding productions) from the grammar.
6. **test\_replace\_terminals\_with\_nonterminals(self):**
   * Tests whether the **replace\_terminals\_with\_nonterminals()** method appropriately replaces terminal symbols with new non-terminal symbols in the grammar.
7. **test\_reduce\_production\_length(self):**
   * Tests whether the **reduce\_production\_length()** method successfully reduces the length of productions to at most two symbols.
8. **test\_grammar\_print(self):**
   * Tests whether the **convert\_to\_cnf()** method correctly converts the grammar to CNF and whether the **print\_grammar()** method properly prints the resulting CNF grammar.

These test methods ensure that the **CNFGrammar** class functions as expected and that each method behaves as intended when operating on a CFG.

**Output**



**Conclusion**

In summary, the **CNFGrammar** class transforms context-free grammars into Chomsky Normal Form (CNF), ensuring adherence to CNF requirements while preserving language equivalence. Its methods systematically modify the grammar structure to achieve this. The accompanying unit test class, **TestCNFGrammarMethods**, verifies the correctness of each conversion step. Together, they form a reliable framework for CNF conversion and testing, essential for language processing tasks requiring CNF grammars.