Ministerul Educaţiei și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică

Laboratory work nr. 5
Course: Formal languages and finite automata
Topic: Chomsky Normal Form

Elaborated:
st. gr. FAF-221 Revenco Victor

Verified:
asist. univ. Cretu Dumitru

Chişinău - 2024

Theory

Chomsky Normal Form (CNF) is a way of representing context-free grammars (CFGs) in a specific form, named after the renowned linguist and cognitive scientist Noam Chomsky. It has several important properties that make it useful for theoretical analysis and practical applications in areas such as natural language processing and parsing algorithms. Here's a breakdown of the key aspects of CNF:

1. Formal Definition: In Chomsky Normal Form, every production rule of the grammar is in one of two forms:
   - $A \rightarrow BC$
   - $A \rightarrow a$

   where A, B, and C are non-terminal symbols, and a is a terminal symbol. The production $A \rightarrow \varepsilon$ (where ε represents the empty string) is allowed only if the start symbol appears on the right-hand side of a production.
2. Non-terminal Usage: In CNF, each non-terminal symbol (except for the start symbol, which can produce ε) must derive at least one string of terminal symbols.
3. Simplification: The removal of useless symbols and unproductive rules is often done before converting a grammar to CNF. Useless symbols are those that cannot be reached from the start symbol, and unproductive rules are those that cannot derive any terminal string.
4. Advantages:
   - CNF simplifies the structure of the grammar, making it easier to analyze and process.
   - It facilitates certain parsing algorithms, such as the CYK (Cocke-Younger-Kasami) algorithm, which operates efficiently on grammars in CNF.
   - CNF helps in proving various properties of context-free languages, such as closure properties.
5. Conversion: Any context-free grammar can be converted to an equivalent grammar in Chomsky Normal Form. The process typically involves several steps, including:
   - Eliminating ε-productions
   - Eliminating unit productions (productions of the form $A \rightarrow B$)
   - Eliminating productions with more than two non-terminals on the right-hand side
   - Introducing new non-terminals as necessary

Objectives:

1. Learn about Chomsky Normal Form (CNF)
2. Get familiar with the approaches of normalizing a grammar.
3. Implement a method for normalizing an input grammar by the rules of CNF.
4. The implementation needs to be encapsulated in a method with an appropriate signature (also ideally in an appropriate class/type).
5. The implemented functionality needs executed and tested.
6. A BONUS point will be given for the student who will have unit tests that validate the functionality of the project.
7. 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

```
def RemoveEpsilon(self):
    # Find occurence of empty string
    nt_epsilon = []
    for key, value in self.P.items():
        s = key
        productions = value
        for p in productions:
            if p == 'eps':
                nt_epsilon.append(s)

    for key, value in self.P.items():
        for ep in nt_epsilon:
            for v in value:
                prod_copy = v
                if ep in prod_copy:
                    # Generate all possible combinations without epsilon
                    new_productions = [prod_copy.replace(c, '') for c in
prod_copy if c != ep]
                    value.extend(new_productions)

    # New copy with added production
    P1 = self.P.copy()
    # Remove empty string from copy
    for key, value in self.P.items():
        for v in value:
            if v == 'eps':
                P1[key].remove(v)

    P_final = {}
    for key, value in P1.items():
        if len(value) != 0:
            P_final[key] = value
        else:
```

```
            self.V_N.remove(key)

    print(f"1.No epsilon productions:\n{P_final}")
    self.P = P_final.copy()
    return P_final
```

Fig1. Remove epsilon method

This method removes epsilon (empty string) productions from the grammar. It first identifies non-terminals that derive epsilon and then removes epsilon from all productions involving these non-terminals. For example, if there is a production **A -> eps**, **A** is added to the list of non-terminals deriving epsilon (**nt_epsilon**). Then, for each production involving **A**, the method generates all possible combinations without epsilon and adds them to the production set. Finally, it removes the epsilon production and updates the grammar.

```
def TransformToCNF(self):
    temp = {}
    new_symbol_counter = 0

    # Generate new symbols for CNF productions
    def generate_new_symbol():
        nonlocal new_symbol_counter
        new_symbol_counter += 1
        return f'X{new_symbol_counter}'

    for key, value in self.P.items():
        new_productions = []
        for production in value:
            while len(production) > 2:
                new_symbol = generate_new_symbol()
                temp[new_symbol] = production[:2]
                new_productions.append(new_symbol)
                production = production[2:]
            new_productions.append(production)
        self.P[key] = new_productions

    # Remove epsilon productions
    for key, value in self.P.items():
        self.P[key] = [v for v in value if v != '']

    # Add new symbols to the grammar
    self.P.update(temp)

    print(f"5. Final Grammar:\n{self.P}")
```

Fig2. Tranform to CNF method

This method transforms the grammar into Chomsky Normal Form (CNF), where each production is either a terminal, a non-terminal followed by two non-terminals, or a non-terminal that produces a single terminal. It iterates through each production and if a production has more than two symbols on the right-hand side, it introduces new non-terminals and splits the production into multiple productions, each with two symbols on the right-hand side.

Overall, these methods work together to transform a context-free grammar into Chomsky Normal Form by removing epsilon productions, unit productions, inaccessible symbols, and unproductive symbols, and then converting the resulting grammar into CNF.

Screenshots

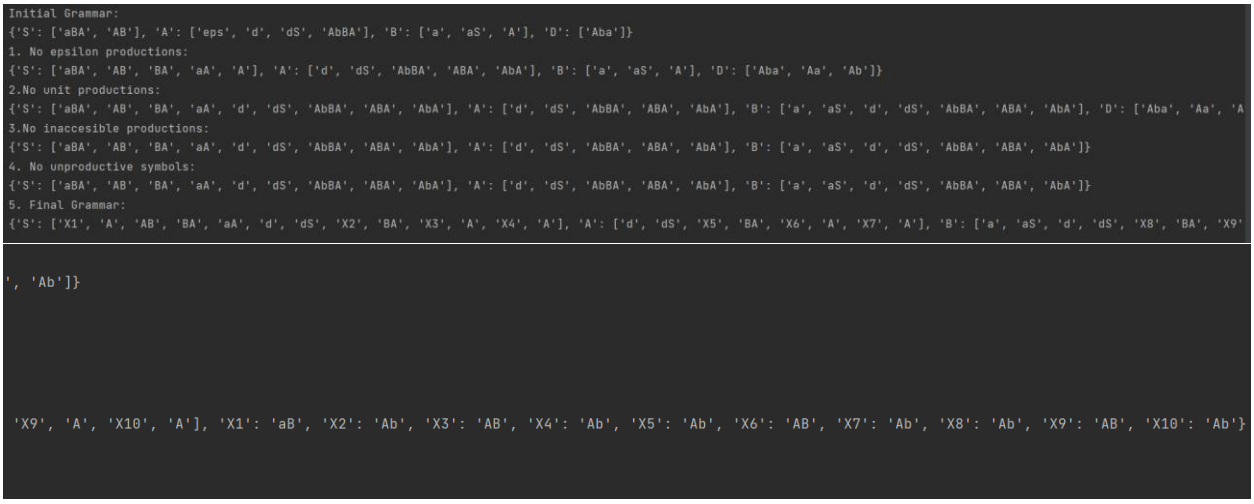

Fig3. Output

Conclusions

This laboratory work involves implementing several key transformations on a context-free grammar (CFG) in Python to convert it into Chomsky Normal Form (CNF). The implemented transformations include removing epsilon productions, eliminating unit productions, removing inaccessible symbols, removing unproductive symbols, and finally, transforming the grammar into CNF.

Each transformation method operates on the CFG's production rules and non-terminal symbols to gradually simplify the grammar and ensure it adheres to the rules of CNF. The TransformToCNF method is particularly crucial, as it standardizes the CFG into CNF by converting productions into pairs of non-terminals or terminals.

By executing these methods in sequence, the initial CFG undergoes a series of transformations, resulting in a CFG that is equivalent in language but structured according to CNF. This work demonstrates the systematic approach required to convert a CFG into CNF, which is a key concept in formal language theory and computational linguistics.