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

Laboratory work 5:
Course: Formal Languages and Finite Automata
Topic: Chomsky Normal Form

Elaborated:
st. gr. FAF-221 Chichioi Iuliana

Verified:
asist. univ. Cretu Dumitru

Chişinău - 2023

TABLE OF CONTENTS

THEORY.....3
OBJECTIVES.....5
IMPLEMENTATION.....5
RESULTS.....9

THEORY

Chomsky Normal Form.

A grammar where every production is either of the form A → BC or A → c (where A, B, C are arbitrary variables and c an arbitrary symbol).

Example:

S → AS | a

A → SA | b

(If language contains ε, then we allow S → ε where S is the start symbol, and forbid S on RHS.)

Why Chomsky Normal Form?

The key advantage is that in Chomsky Normal Form, every derivation of a string of n letters has exactly

2n − 1 steps. Thus: one can determine if a string is in the language by exhaustive search of all derivations.

Conversion

The conversion to Chomsky Normal Form has four main steps:

1. Get rid of all ε productions.
2. Get rid of all productions where RHS is one variable.
3. Replace every production that is too long by shorter productions.
4. Move all terminals to productions where RHS is one terminal.

1) Eliminate ε Productions

Determine the nullable variables (those that generate ε) (algorithm given earlier).
Go through all productions, and for each, omit every possible subset of nullable variables.
For example, if P → AxB with both A and B

nullable, add productions P → xB | Ax | x. After this, delete all productions with empty RHS.

2) Eliminate Variable Unit Productions

A unit production is where RHS has only one symbol.

Consider production A → B. Then for every production B → α, add the production A → α. Repeat until done (but don't recreate a unit production already deleted).

3) Replace Long Productions by Shorter Ones

For example, if have production A → BCD, then replace it with A → BE and E → CD.

(In theory this introduces many new variables, but one can re-use variables if careful.)

4) Move Terminals to Unit Productions

For every terminal on the right of a non-unit production, add a substitute variable.

For example, replace production A → bC with productions A → BC and B → b.

Example

Consider the CFG:

S → aXbX

X→aY |bY |ε

Y→X|c

The variable X is nullable; and so therefore is Y . After elimination of ε, we obtain:

S →aXbX |abX |aXb|ab X→aY |bY |a|b Y→X|c

Example: Step 2

After elimination of the unit production Y → X, we obtain:

S →aXbX |abX |aXb|ab

X→aY |bY |a|b

Y →aY |bY |a|b|c

Example: Steps 3 & 4

Now, break up the RHSs of S; and replace a by A, b by B and c by C wherever not units:

S → EF | AF | EB | AB

X → AY | BY | a | b

Y →AY |BY |a|b|c

E → AX

F → BX

A→a

B→b

C→c

There are special forms for CFGs such as Chomsky Normal Form, where every production has the form A→B C or A→c. The algorithm to convert to this form involves (1) determining all nullable variables and getting rid of all ε-productions, (2) getting rid of all variable unit productions, (3) breaking up long productions, and (4) moving terminals to unit productions.

OBJECTIVES

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

Assigned variant:

Variant 5

1.Eliminate ε productions.
2.Eliminate any renaming.
3.Eliminate inaccessible symbols.
4. Eliminate the non productive symbols.
5. Obtain the Chomsky Normal Form.

$G=(V_N, V_T, P, S)$ $V_N=\{ S, A, B, C, D\}$ $V_T=\{ a, b, d\}$

P={ 1. S→dB
2. S→A
3. A→d
4. A→dS
5. A→aBdB
6. B→a
7. B→aS
8. B→AC
9. D→AB
10. C→bC
11. C→ε }

The provided Python code consists of two files: Lab_5.py and Unit_test.py.

Lab_5.py:

This file defines a class Grammar encapsulating a CFG and methods to perform various transformations for achieving CNF.

Eliminating Epsilon Productions (elim_epsilon):

This method first identifies non-terminal symbols (nt_epsilon) that can produce epsilon.It then iterates through the grammar's productions and removes epsilon productions by appending modified productions where epsilon is removed. After processing, it removes non-terminal symbols that only produce epsilon and updates the grammar accordingly.

```
for key, value in self.P.items():
    if key in nt_epsilon and len(value) < 2:
        del P1[key]
    else:
        for v in value:
```

```
            if v == 'epsilon':
                P1[key].remove(v)
```

Eliminating Unit Productions (elim_unit_prod):

This method iterates through each production in the grammar.If a production contains a single non-terminal symbol, it replaces it with the productions of the corresponding non-terminal.This process continues until no unit productions are left in the grammar.

```
    for key, value in self.P.items():
        for v in value:
            if len(v) == 1 and v in self.V_N:
                P2[key].remove(v)
                for p in self.P[v]:
                    P2[key].append(p)
```

Eliminating Inaccessible Symbols (elim_inaccesible_symb):

It initializes a list accesible_symbols with all non-terminal symbols. Then, it traverses through the grammar's productions, removing symbols that are not reachable from the start symbol. Finally, it removes the inaccessible symbols from the grammar.

```
    for el in accesible_symbols:
        del P3[el]
```

Eliminating Unproductive Symbols (`elin_unnprod_symb`):

This method removes symbols that cannot generate any terminal string. It first copies the grammar's productions. Then, it iterates through each production, checking if it contains only terminals. If not, it removes the non-terminal symbol if it's not productive. Additionally, it removes productions containing undefined non-terminal symbols.

```
    for key, value in self.P.items():
        count = 0
        for v in value:
            if len(v) == 1 and v in self.V_T:
                count += 1
        if count == 0:
            del P4[key]
```

Transforming to Chomsky Normal Form (transf_to_cnf):

This method converts the grammar to Chomsky Normal Form (CNF). It initializes a dictionary temp to hold temporary symbols during the transformation. It iterates through each production, splitting non-unit productions into pairs of non-terminals. Temporary symbols are used to represent these pairs. It replaces original productions with the generated CNF productions. After the transformation, it returns the CNF grammar.

```
    for key, value in self.P.items():

        for v in value:

            if (len(v) == 1 and v in self.V_T) or (len(v) == 2 and v.isupper()):

                continue

            else:

                # Split non-unit productions into pairs of non-terminals

                left = v[:len(v) // 2]

                right = v[len(v) // 2:]
```

Each method modifies the grammar according to the respective transformation rules. These transformations are essential for simplifying and standardizing context-free grammars, making them easier to analyze and process in various computational applications.

Unit_test.py:

This file contains unit tests designed to thoroughly validate the functionality of the Grammar class's methods for transforming a CFG into CNF. It utilizes Python's built-in unittest framework to systematically test each transformation step and ensure the resulting CNF adheres to the defined standards.

Setup (setUp method):

The setUp method is executed before each test case. It creates an instance of the Grammar class with a predefined CFG for testing purposes. This CFG typically includes a mix of productions and symbols representing various CFG structures and complexities.

```
def setUp(self):

    self.g = Grammar()

    self.P1, self.P2, self.P3, self.P4, self.P5 = self.g.ReturnProductions()
```

Test Cases (test_... methods):

Each test case method within the TestGrammar class targets a specific transformation step or aspect of the CNF conversion process. For example, test_elim_epsilon verifies the correct elimination of epsilon productions, ensuring productions that derive an empty string ('epsilon') are properly removed from the grammar.

```
def test_elim_epsilon(self):

    # Define the expected result after eliminating epsilon productions

    expected_result = {'S': ['dB', 'd', 'dS', 'aBdB'],

                       'A': ['d', 'dS', 'aBdB'],

                       'B': ['a', 'aS', 'AC', 'd', 'dS', 'aBdB'],

                       'D': ['AB'],

                       'C': ['bC', 'b']

                       }

    # Assert that the actual result matches the expected result

    self.assertEqual(self.P1, expected_result)
```

Assertions:

Each test case typically includes one or more assertions using methods provided by the unittest framework, such as assertEqual, assertTrue, or assertIn. These assertions compare the actual output of the transformation method (e.g., the modified grammar) with the expected output based on the predefined CFG and the rules of CNF.

Execution (if __name__ == '__main__':):

The if __name__ == '__main__': block at the end of the file ensures that the unit tests are executed when the script is run as the main program. Running the file executes all defined unit tests, providing detailed feedback on the success or failure of each test case.

RESULTS

```
------------------------------------------------
Initial Grammar:
S -> dB | A
A -> d | dS | aBdB
B -> a | aS | AC
D -> AB
C -> bC | epsilon
------------------------------------------------
1. Eliminating epsilon productions:
S -> dB | A
A -> d | dS | aBdB
B -> a | aS | AC | A
D -> AB
C -> bC | b
------------------------------------------------
2. Eliminating unit productions:
S -> dB | d | dS | aBdB
A -> d | dS | aBdB
B -> a | aS | AC | d | dS | aBdB
D -> AB
C -> bC | b
------------------------------------------------
3. Eliminating inaccessible symbols:
S -> dB | d | dS | aBdB
A -> d | dS | aBdB
B -> a | aS | AC | d | dS | aBdB
C -> bC | b
------------------------------------------------
```

```
------------------------------------
4. Eliminating unproductive symbols:
S -> dB | d | dS | aBdB
A -> d | dS | aBdB
B -> a | aS | AC | d | dS | aBdB
C -> bC | b
------------------------------------
5. Obtain Chomsky Normal Form(CNF):
S -> DE | d | DF | GH
A -> d | DF | GH
B -> a | IF | AC | d | DF | GH
C -> JK | b
D -> d
E -> B
F -> S
G -> aB
H -> dB
I -> a
J -> b
K -> C
------------------------------------
```

Figure 1 Lab_5.py


```
------------------------------------------------
Initial Grammar:
S -> dB | A
A -> d | dS | aBdB
B -> a | aS | AC
D -> AB
C -> bC | epsilon
------------------------------------------------
1. Eliminating epsilon productions:
S -> dB | A
A -> d | dS | aBdB
B -> a | aS | AC | A
D -> AB
C -> bC | b
------------------------------------------------
2. Eliminating unit productions:
S -> dB | d | dS | aBdB
A -> d | dS | aBdB
B -> a | aS | AC | d | dS | aBdB
D -> AB
C -> bC | b
------------------------------------------------
3. Eliminating inaccessible symbols:
S -> dB | d | dS | aBdB
A -> d | dS | aBdB
B -> a | aS | AC | d | dS | aBdB
C -> bC | b
------------------------------------------------
```

```
------------------------------------
4. Eliminating unproductive symbols:
S -> dB | d | dS | aBdB
A -> d | dS | aBdB
B -> a | aS | AC | d | dS | aBdB
C -> bC | b
------------------------------------
5. Obtain Chomsky Normal Form(CNF):
S -> DE | d | DF | GH
A -> d | DF | GH
B -> a | IF | AC | d | DF | GH
C -> JK | b
D -> d
E -> B
F -> S
G -> aB
H -> dB
I -> a
J -> b
K -> C
------------------------------------
```

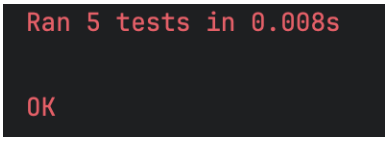

Figure 1 Unit_test.py

CONCLUSION

The completion of Laboratory Work 5 on Chomsky Normal Form (CNF) for the course "Formal Languages and Finite Automata" has provided a comprehensive understanding of CNF and its significance in formal language theory. Through the implementation of various transformation steps, including the elimination of epsilon productions, unit productions, inaccessible symbols, and unproductive symbols, as well as the conversion to CNF, valuable insights into the normalization process of context-free grammars were gained.

The objectives of the laboratory work, including learning about CNF, becoming familiar with grammar normalization approaches, and implementing a method for normalizing grammars according to CNF rules, were successfully achieved. The implementation, encapsulated within the Grammar class in Lab_5.py, demonstrated the application of theoretical concepts into practical coding solutions.

Furthermore, the inclusion of unit tests in Unit_test.py provided an essential mechanism for validating the correctness and robustness of the implemented methods. By systematically testing each transformation step, the unit tests ensured that the resulting CNF grammar adhered to the specified standards and maintained the integrity of the original grammar's structure and rules.

In conclusion, the laboratory work on Chomsky Normal Form has contributed significantly to the understanding of formal languages and finite automata, providing valuable hands-on experience in grammar normalization techniques and reinforcing theoretical knowledge through practical implementation and testing.

REFERENCES

https://github.com/Chiuliana/DSL_Laboratory_Works/tree/main/Labs/Lab_5_Chomsky_Normal_Form