CNF Converter

Parfene Daniel, FAF-222

April 22, 2024

1 Introduction

The Chomsky Normal Form (CNF) algorithm is a crucial transformation in formal language theory, particularly for parsing context-free grammars. In this report, we discuss the implementation and functionality of a Python script that converts a given context-free grammar into Chomsky Normal Form.

2 Implementation Overview

The CNF algorithm is implemented as a Python class named `ChomskyNormalForm`. This class takes a context-free grammar as input and applies a series of transformations to convert it into Chomsky Normal Form. The implemented transformations include:

1. Removal of epsilon productions
2. Removal of unit productions
3. Removal of inaccessible symbols
4. Removal of non-productive symbols
5. Conversion to Chomsky Normal Form

3 Algorithm Details

3.1 Removal of Epsilon Productions

Epsilon productions are productions that derive the empty string (''). The algorithm identifies epsilon-producing non-terminals and iteratively removes them from productions until no more changes can be made. Additionally, it adjusts productions to account for epsilon removal.

```
1
2 ### FINDING EPSILON PRODUCTION AND STOCKING THEM
3         for non_terminal, productions in self.P.items():
4             if '' in productions:
5                 epsilon_producing_non_terminals.add(non_terminal)
6
7 ### ITERATING THROUGH THE PRODUCTIONS AND SUBSTITUTE EPSILONS
8 changes_made = True
9         while changes_made:
10            changes_made = False
11            for non_terminal, productions in self.P.items():
12                for production in productions:
13                    if all(c in epsilon_producing_non_terminals or
      c == '' for c in production):
14                        if epsilon_producing_non_terminals.add(
      non_terminal):
15                            changes_made = True
```

3.2 Removal of Unit Productions

Unit productions are productions where a non-terminal directly produces another non-terminal. The algorithm identifies and removes such unit productions iteratively until no more changes can be made.

```
1
2 ### Separate unit and other productions
3         for non_terminal, productions in self.P.items():
4             for production in productions:
5                 if len(production) == 1 and production in self.Vn:
6                     direct_unit_productions.setdefault(non_terminal
      , []).append(production)
7                 else:
8                     result_productions.setdefault(non_terminal, [])
      .append(production)
9
10 ### Expand unit productions to their respective non-terminals
11                    for prod in current_productions:
12                        if len(prod) == 1 and prod in self.Vn and
      prod not in visited:
13                            to_visit.append(prod)
14                        else:
15                            result_productions.setdefault(
      non_terminal, []).append(prod)
```

3.3 Removal of Inaccessible Symbols

Inaccessible symbols are non-terminals that cannot be reached from the start symbol. The algorithm identifies and removes such symbols and their productions from the grammar.

```
1
2 ### ITERATE THROUGH ALL PRODUCTIONS AND ITENTIFY WHICH WE CAN
      ACCESS
3 ### THE ONES LEFT ARE INACCESSIBLE
4
5     if symbol in self.Vn and symbol not in accessible_symbols:
6         accessible_symbols.add(symbol)
7         changed = True
```

3.4 Removal of Non-Productive Symbols

Non-productive symbols are non-terminals that cannot derive any string of terminals. The algorithm identifies and removes such symbols and their productions from the grammar.

```
1
2 ### ITERATE THROUGH ALL PRODUCTIONS AND ITENTIFY WHICH
3 ### OF THEM DON'T COME TO A TERMINAL
4
5     if non_terminal not in productive_symbols:
6         for production in productions:
7             production_is_productive = all(sym in self.Vt or sym in
       productive_symbols for sym in production)
8                 if production_is_productive:
9                     productive_symbols.add(non_terminal)
```

3.5 Conversion to Chomsky Normal Form

Finally, the algorithm converts the modified grammar into Chomsky Normal Form. This involves replacing productions with more than two non-terminals or terminals with new non-terminals.

4 Code Usage

To utilize the Chomsky Normal Form algorithm, users need to instantiate the `ChomskyNormalForm` class with a context-free grammar provided as a dictionary. The grammar should include sets of non-terminals (`VN`), terminals (`VT`), and productions (`P`). After instantiation, the transformed grammar can be accessed through the `P` attribute of the object.

5 Conclusion

The Chomsky Normal Form algorithm is a fundamental tool in formal language theory, aiding in the analysis and parsing of context-free grammars. The implemented Python script provides a practical implementation of this algorithm, allowing for the conversion of arbitrary context-free grammars into Chomsky Normal Form, which facilitates further processing and analysis.