PES UNIVERSITY

Department of Computer Science and Engineering

PES University, Bangalore, India

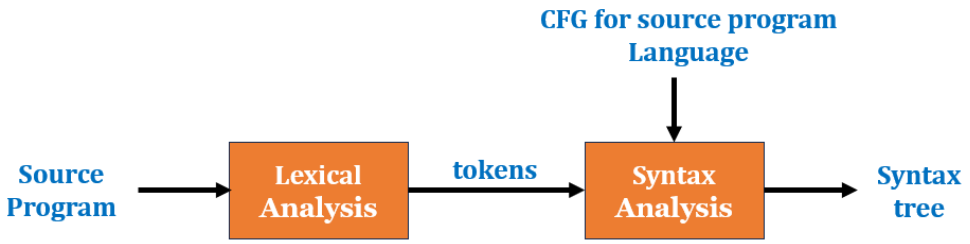
UE23CS243A: Automata Formal Language and Logic

Prakash C O, Associate Professor, Department of CSE

Chomsky Normal Form (CNF)

- A Context-Free Grammar (CFG) is especially useful in the field of both computer science and linguistics for describing the structure of programming languages and natural languages.
- Context-Free Grammars (CFGs) are powerful enough to express the syntax of most programming languages.

- **Syntax analyzer** (also known as Parser) receives the source code in the form of tokens from the lexical analyzer and determines whether the input token stream is syntactically well formed (i.e., by constructing a parse tree) with respect to a given grammar and creates an output a tree-like intermediate representation (i.e., syntax tree).



- Depending upon how the parse tree is built, parsing techniques are classified into three general categories:
  1. Universal,
  2. Top-down, and
  3. Bottom-up.
- Universal parsing methods such as the Cocke-Younger-Kasami (CYK) algorithm and Earley's algorithm can parse any grammar. These general methods are, however, too inefficient to use in production compilers.

- In computer science, the Cocke–Younger–Kasami algorithm (alternatively called CYK, or CKY) is a parsing algorithm for context-free grammars published by Itiroo Sakai in 1961.

- **The standard version of CYK operates only on context-free grammars (CFGs) given in Chomsky normal form (CNF).**

- The problem is that context-free grammars can be very complicated, with long and complex rules. It is therefore desirable to be able to describe the original CFG in a simplified ("normal") form.

- If G is a CFG in CNF, then for any string $w \in L(G)$ of length $n \geq 1$, exactly $2n-1$ steps are required for any derivation of w.

Chomsky Normal Form (CNF)

The first such simplified form was produced by Noam Chomsky, the famous linguist and the author of many concepts actively used in programming languages.

CNF Definition:

Noam Chomsky showed that every CFG can be transformed into a simplified form, in which only three types of rules are allowed:

- **A → BC**, where A, B, and C are variables.
- **A → a**, where A is a variable and a is a terminal symbol; and
- **S → ε**, where S is the start symbol of the grammar, and ε means an empty string. This rule can only appear if ε is in $L(G)$.

A context-free grammar, G, is said to be in Chomsky normal form if all of its production rules are of the above forms.

What is the application of Chomsky's Normal Form?

- CNF is used to make the grammar more structured and easier to analyze.
- A CFG in CNF is used in **natural language parsing (NLP)**, **compiler design, grammar optimization** etc.

Why Chomsky Normal Form?

- **The key advantage is that in CNF, 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.**
- Normal forms make many concepts much easier to handle because you can assume a simple structure for them.
- CNF enables a polynomial time algorithm to decide whether a string can be generated by the grammar.

CFG to CNF conversion steps:

Step 1: Eliminate lambda (or epsilon) productions from non-starting variables

Determine the nullable variables (those that generate empty string (ε), i.e., X is nullable, if there is a derivation if $X \overset{*}{\Rightarrow} \varepsilon$). Go through all productions, and for each, omit every possible subset of nullable variables.

- For example, if G is

  P → AxB
  A→ ε ⇨ **P → x**
  B→ ε

  (Here, both A and B is deriving only epsilon or lambda, then consider only without A and B in production body), After eliminating all epsilon or lambda productions, the CFG is **P → x**

- For example, if G is

  P → AxB ⇨ **P → Ax** ⇨ **P → Ax | x**
  A→ a | ε **A→ a | ε** **A→ a**
  B→ ε

  After eliminating lambda production **B→ ε** (Here, B is deriving only epsilon or lambda, then consider only without B in production body), the CFG is **P → Ax** and **A→ a | ε**

After eliminating lambda production, **A→ ε** (Here, A is deriving not only epsilon or lambda, but also derives a, then consider with and without A in production body),
the CFG is **P → Ax | x** and **A→ a**

- For example, if G is

P → AxB
A→ a | ε
B→ b | ε

⇒

P → AxB | xB
A→ a
B→ b | ε

⇒

P → AxB | Ax | xB | x
A→ a
B→ b

After eliminating epsilon or lambda production **A→ ε**, (Here, A is deriving not only epsilon or lambda, but also derives a, then consider with and without A in production body),
the CFG is **P → AxB | xB** and **A→ a**, **B→ b | ε**
After eliminating epsilon or lambda production **B→ ε**, (Here, B is deriving not only epsilon or lambda, but also derives b, then consider with and without B in production body),
the CFG is **P → AxB | Ax | xB | x** and **A→ a**, **B→ b**

Step2: Eliminate Unit productions of type Variable→Variable (i.e., of type A→B)

- Consider production A → B. Then for every production B → α, add the production A → α. Repeat until done (but don't re-create a unit production already deleted).
- For example, if G is

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

Step 3: Eliminate Useless Symbols (Eliminate non-generating and non-reachable symbols)

There are 2 aspects:

- **Derivability:** Each Variable must derive a string/must end with set of terminal(s).
  For example, If we drop production rules containing non-generating B from the following grammar

S → AB | a we obtain **S → a** (i.e., after dropping the production rule S → AB)
A → b **A → b** (Now A is unreachable and became useless)

- **Reachability:** Each Variable must be reachable from S.
  For example, if we drop production rules containing unreachable C and c from the following grammar

S → Aa | a we obtain **S → Aa | a**
A → b **A → b**
C → c

Step 4: Convert the CFG to CNF as per definition.

- **Move Terminals to Unit Productions of type Variable→terminal**
  That is, **for every terminal on the RHS of a non-unit production, add a substitute variable**.
  For example, replacing b by B in RHS and by introducing new Rule B → b in the following grammar

A → bC | bb | a | b we obtain **A → BC | BB | a | b**
C → c **C → c**
B → b

- **Replace Long Production bodies (i.e., length>2) by Shorter Ones**
  (i.e., Eliminate right-hand sides with more than 2 nonterminals)
  For example, in the following grammar
  A → BCD then after replacing long production bodies, we obtain

A → BE or **A → ED**
E → CD **E → BC**

Note-1:

1. **The CNF normalization is performed in order to standardize the grammar, because the RHS of grammar productions have no specific format and hence the parsing process is hard.**
2. **The CNF idea is to make every step in derivation useful.**
3. **For a given grammar, there can be more than one CNF.**
4. **CNF produces the same language as generated by CFG.**
5. **Chomsky Normal Form (CNF) turns every non-binary parse tree into binary parse tree.**
6. **The different order of CNF conversion steps (i.e., 2, 1, 3, 4 or 4, 2, 1, 3 …) may lead to an exponential blow up in the number of rules in some CFGs.**

Note-2: Useless symbols

Consider a grammar G = (N, T, P, S)

$X \in N \cup T$ is useful, if there is a derivation $S \overset{*}{\Rightarrow} \alpha X \beta \overset{*}{\Rightarrow} w$ for $w \in T^*$

$X \in N \cup T$ is useless, if X is not useful.

It is clear that eliminating production rules involving useless symbols have no impact on the language of the grammar.

Note-3: Generating and reachable symbols

Consider a grammar G = (N, T, P, S)

$X \in N \cup T$ is generating, if $X \overset{*}{\Rightarrow} w$ for some $w \in T^*$

$X \in N \cup T$ is reachable, if there is some derivation $S \overset{*}{\Rightarrow} \alpha X \beta$

Example 1: Convert the following CFG into an equivalent CFG in CNF.

S → aXbX
X → aY | bY | ε
Y → X | c

Solution:

Step 1: Eliminate lambda productions

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

Step2: Eliminate Unit productions of type Variable→Variable

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

Step 3: Eliminating Useless productions and Symbols

No useless productions and Symbols.

Step 4: Convert the CFG to CNF as per definition.

Move Terminals to Unit Productions of type **Variable→terminal**
(i.e., For every terminal on the RHS of a non-unit production, add a substitute variable.)
Replace a by A, and b by B, by introducing new Rules `A → a and B → b`

```
S → AXBX | ABX | AXB| AB
X → AY | BY | a | b
Y → AY | BY | a | b | c
A → a
B → b
```

Replace Long Production bodies (i.e., length>2) by Shorter Ones

```
S → EF | AF | EB | AB
X → AY | B Y | a | b
Y → AY | BY | a | b | c
E → AX
F → BX
A → a
B → b
```

Example 2: Convert the following CFG into an equivalent CFG in CNF.

S → AbA
A → Aa | ε

Solution:

Step 1: Eliminate lambda productions

```
S → AbA                          S → AbA | bA | Ab | b
A → Aa | ε        ⇨              A → Aa | a
```

Step2: Eliminate Unit productions of type Variable→Variable

No unit productions

Step 3: Eliminating Useless productions and Symbols

No useless productions and Symbols.

Step 4: Convert the CFG to CNF as per definition.

Move Terminals to Unit Productions of type **Variable→terminal**

```
S → AbA | bA | Ab | b            S → ABA | BA | AB | b
A → Aa | a              ⇨        A → AC | a
                                 B → b
                                 C → a
```

Replace Long Productions by Shorter Ones

```
S → ABA | BA | AB | b            S → TA | BA | AB | b
A → AC | a                       T → AB
B → b                   ⇨        A → AC | a
C → a                            B → b
                                 C → a
```

Note: If G is a CFG in CNF, then for any string w∈L(G) of length n≥1, exactly 2n−1 steps are required for any derivation of w.

Example: Consider the string 'aba' and the above CFG in CNF.

LMD: S ⇒ TA
⇒ ABA
⇒ aBA
⇒ abA
⇒ **aba** The string 'aba' is derived in 2n-1=2*3-1=5 steps.

Example 3: Convert the following CFG into an equivalent CFG in CNF.

S → AB
A → aAA | ε
B → bBB | ε

Solution:

Step 1: Eliminate lambda productions

The variables A and B are both nullable; and so therefore is S.

S → AB
A → aAA | ε
B → bBB | ε

⇨

S → AB | ε | A | B
A → aAA | a | aA
B → bBB | b | bB

Step2: Eliminate Unit productions of type Variable→Variable

S → AB | ε | A | B
A → aAA | a | aA
B → bBB | b | bB

⇨

S → AB | ε | aAA | a | aA | bBB | b | bB
A → aAA | a | aA
B → bBB | b | bB

Step 3: Eliminating Useless productions and Symbols

No useless productions and Symbols.

Step 4: Convert the CFG to CNF as per definition.

Move Terminals to Unit Productions of type **Variable→terminal**

S → AB|ε|aAA|a|aA|bBB|b|bB
A → aAA | a | aA
B → bBB | b | bB

⇨

S → AB|ε|CAA|a|CA|DBB|b|DB
A → CAA | a | CA
B → DBB | b | BB
C → a
D → b

Replace Long Productions by Shorter Ones

S → AB|ε|CAA|a|CA|DBB|b|DB
A → CAA | a | CA
B → DBB | b | BB
C → a
D → b

⇨

S → AB|ε|EA|a|CA|FB|b|DB
A → EA | a | CA
B → FB | b | BB
C → a
D → b
E → CA
F → DB

Example 4: Convert the following CFG into an equivalent CFG in CNF.

S → ASA | A | ε
A → 00 | ε

Solution:

Step 1: Eliminate lambda productions

S → ASA | A | ε
A → 00 | ε

⇨

S → ASA | S | SA | AS | A | ε
A → 00

Step2: Eliminate Unit productions of type Variable→Variable

Eliminate first, the unit production **S → S** because it's useless.

S → ASA | S | SA | AS | A | ε
A → 00

⇨

S → ASA | SA | AS | A | ε
A → 00

Eliminate next, the unit production $\mathbf{S \rightarrow A}$

S → ASA | SA | AS | A | ε
A → 00

⇨

S → ASA | SA | AS | 00 | ε
A → 00

Step 3: Eliminating Useless productions and Symbols

No useless productions and Symbols.

Step 4: Convert the CFG to CNF as per definition.

Move Terminals to Unit Productions of type **Variable→terminal**

S → ASA | SA | AS | 00 | ε
A → 00

⇨

S → ASA | SA | AS | BB | ε
A → BB
B → 0

Replace Long Productions by Shorter Ones

S → ASA | SA | AS | BB | ε
A → BB
B → 0

⇨

S → CA | SA | AS | BB | ε
C → AS
A → BB
B → 0

Example 5: Convert the following CFG into an equivalent CFG in CNF.

S → ABA
A → aA | ε
B → bBc | ε

Note: nullable(G) = {A, B, S}

Example 6: Convert the following CFG into an equivalent CFG in CNF.

E → E + T | T
T → T * F | F
F → num | id

Exercises:

1) Convert the following grammar to Chomsky Normal Form (CNF).

S → X | XYa | XbX
X → Xa |λ
Y → Yb | YZ
Z → ZY | ZX | bY

2) If G is the grammar in CNF, then fill the following table

w	n=|w|	Length of Derivation (2n-1)	Max-depth of the parse tree	Min-depth of the parse tree
λ	0	1		
a_1	1	1		
$a_1\ a_2$	2	3		
$a_1\ a_2\ a_3$	3	5		
$a_1\ a_2\ a_3\ a_4$	4	7		
$a_1\ a_2\ a_3\ a_4\ a_5$	5	9		
$a_1\ a_2\ a_3\ a_4\ a_5\ a_6$	6	11		

Note:

- **If G is a CFG in CNF, then for any string $w \in L(G)$ of length $n \geq 1$, exactly $2n-1$ steps are required for any derivation of w.**

- In the tree, the total number of edges from the root node to the leaf node in the longest path is known as "Depth of Tree".

3) Convert the following CFG to CNF.

```
S → aAa | bBb | BB
A → C
B → S | A
C → S | λ
```

4) Design a CNF grammar for the set of strings of balanced parentheses.

Advantages of Chomsky Normal Form (CNF)

Chomsky Normal Form (CNF) has many advantages in formal language theory and algorithm parsing. Some of these advantages include:

- **Simple and Standardized grammar**
- **Efficient Parsing Algorithms**
- **Less Grammar Size**
- **Elimination of Empty String Rules**

Questions

1) What is Chomsky's normal form in TOC?

- In TOC **Chomsky Normal Form** is a special form of context-free grammar (CFG).
- The production rules of **Chomsky Normal Form are limited to only single terminal** and **two non-terminal symbols.** If empty string (ε) is in $L(G)$, then **S → ε**, is allowed, where S is the start symbol of the grammar.

2) What is Chomsky's theory called?

- **Chomsky's theory,** formed by **Noam Chomsky** is called **Linguistic Theory** or **Chomskyan Generative Grammar.**

3) What are the key elements of Chomsky's theory?

- The key elements of **Chomsky's** theory include **Universal grammar** (noun, pronoun, adverb etc.), **transformational rules** between grammars**, surface** and **deep** structure language acquisition and most importantly a person's **innate ability** to learn grammar and language.

4) What is the application of Chomsky's Normal Form?

- Chomsky's Normal Form is a type of context free grammar which is used to make the grammar more structured and easier to analyze.
- CNF is used in **natural language processing**, **algorithm parsing**, **compiler design, grammar optimization** etc.

Additional Info:

Chomsky reduced form

Another way to define the Chomsky normal form is:
A formal grammar is in **Chomsky reduced form** if all of its production rules are of the form:

- **A → BC**, where A, B, and C are variables.

- **A → a**, where A is a variable and a is a terminal symbol;

Only those context-free grammars which do not generate the empty string can be transformed into Chomsky reduced form.

Chomsky's normal form (Approach-2)

The following steps are used to convert context-free grammar into Chomsky's normal form.

1. Introduce a new start variable that produces the old start variable (i.e., The start variable should not appear on the right-hand side of any rule)
2. Remove the null productions from non-starting variables.
3. Remove the unit productions.
4. Eliminate Useless Symbols (i.e., Eliminate non-generating and non-reachable symbols)
5. Convert all rules such that the right-hand side has only one terminal or two variables. New variables can be introduced to convert the rules.