
THEORY OF COMPUTATION

Chapter 4 CONTEXT FREE LANGUAGES

Complete Study Notes for Engineering Students

Theory · Solved Examples · Derivation Trees · 15 MCQs with
Answer Key · 10 Broad Questions

Topics Covered

- 4.1 Context free grammars · Examples · Leftmost and rightmost derivation · Derivation tree
- 4.2 Ambiguity in context free grammar and parse tree · Removal of ambiguity
- 4.3 Simplification of CFG · Useless symbols · Unit productions · ϵ -productions
- 4.4 Chomsky normal form and Greibach normal form

Contents

4.1 Context Free Grammars	3
4.1.1 Grammar and the Chomsky Hierarchy	3
4.1.2 Definition of a Context Free Grammar	4
4.1.3 Examples of Context Free Languages and Grammars	5
4.1.4 Leftmost and Rightmost Derivations	5
4.1.5 Derivation Tree (Parse Tree)	6
4.2 Ambiguity in Context Free Grammar	8
4.2.1 Ambiguous Grammar and Parse Tree	8
4.2.2 Removal of Ambiguity	9
4.3 Simplification of Context Free Grammars	10
4.3.1 Removal of Useless Symbols	10
4.3.2 Removal of Unit Productions	11
4.3.3 Removal of ϵ -Productions	11
4.4 Normal Forms of Context Free Grammars	13
4.4.1 Chomsky Normal Form (CNF)	13
4.4.2 Greibach Normal Form (GNF)	14
4.4.3 Comparison of CNF and GNF	16
Chapter Summary	17
Multiple Choice Questions (15)	18
Answer Key	20
Broad Questions (10)	21

Context Free Languages

Regular expressions and finite automata cannot describe languages such as $\{a^n b^n\}$ or the set of balanced brackets, because a finite automaton has no memory for counting. **Context free grammars** (CFGs) are a more powerful notation: they describe nested, recursive structure and are the basis of the syntax of almost every programming language. Compilers use them to build *parse trees* of programs.

After studying this chapter you should be able to:

- define a CFG and write grammars for common languages;
- write leftmost and rightmost derivations and draw derivation (parse) trees;
- decide whether a grammar is ambiguous and remove the ambiguity;
- remove useless symbols, unit productions and ϵ -productions to simplify a grammar;
- convert a grammar into **Chomsky normal form** and **Greibach normal form**.

Note: in this chapter the empty string is written ϵ (the same as Λ used in Chapter 3).

4.1 Context Free Grammars

4.1.1 Grammar and the Chomsky Hierarchy

A **grammar** is a set of rules for generating the strings of a language. Formally, a grammar is a 4-tuple $G = (V, T, P, S)$:

Symbol	Meaning	Example
V	Finite set of variables (non-terminals, syntactic categories).	$V = \{S, A\}$
T	Finite set of terminals (the alphabet of the language); $V \cap T = \emptyset$.	$T = \{a, b\}$
P	Finite set of productions (rules) of the form $\alpha \rightarrow \beta$.	$S \rightarrow aSb$
S	The start symbol , $S \in V$.	S

Table 4.1: Components of a grammar

Noam Chomsky classified grammars into four types according to the form of their productions. Each type generates a class of languages recognised by a particular machine.

Type	Grammar	Production form	Language class	Machine
0	Unrestricted	$\alpha \rightarrow \beta$ (α contains a variable)	Recursively enumerable	Turing machine
1	Context sensitive	$ \alpha \leq \beta $	Context sensitive	Linear bounded automaton
2	Context free	$A \rightarrow \alpha$ (A a single variable)	Context free	Pushdown automaton
3	Regular	$A \rightarrow aB$ or $A \rightarrow a$	Regular	Finite automaton

Table 4.2: Chomsky hierarchy

Every regular language is context free, every context free language is context sensitive, and so on; each class properly contains the next.

4.1.2 Definition of a Context Free Grammar

A grammar $G = (V, T, P, S)$ is **context free** if every production has the form

$$A \rightarrow \alpha, \text{ where } A \in V \text{ and } \alpha \in (V \cup T)^*.$$

The left side is *always a single variable*, so A can be replaced by α *regardless of the context* (the symbols around A) - hence the name. Productions with the same left side are combined using "|": $A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_k$.

Key terms

Term	Meaning
Derivation step $\Rightarrow$	$\alpha A \beta \Rightarrow \alpha \gamma \beta$ if $A \rightarrow \gamma$ is a production: one variable is replaced by the right side of a production.
Derivation $\Rightarrow^*$	Zero or more derivation steps. $S \Rightarrow^* w$ means w can be derived from S .
Sentential form	Any string $\alpha \in (V \cup T)^*$ with $S \Rightarrow^* \alpha$ (may contain variables).
Sentence	A sentential form made only of terminals.
Language of G	$L(G) = \{ w \in T^* \mid S \Rightarrow^* w \}$.
Context free language (CFL)	A language L for which some CFG G has $L = L(G)$.

Table 4.3: Key terms

4.1.3 Examples of Context Free Languages and Grammars

Language	Context free grammar	Idea
$\{a^n b^n \mid n \geq 0\}$	$S \rightarrow aSb \mid \varepsilon$	Each step adds one a on the left and one b on the right.
$\{a^n b^{2n} \mid n \geq 0\}$	$S \rightarrow aSbb \mid \varepsilon$	Two b's for every a.
Palindromes over $\{a, b\}$	$S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon$	Match the ends, recurse inside; a, b, ε give the middle.
$\{ww^R \mid w \in \{a, b\}^*\}$ (even palindromes)	$S \rightarrow aSa \mid bSb \mid \varepsilon$	Same, but no single middle symbol.
Balanced parentheses	$S \rightarrow (S)S \mid \varepsilon$	Every "(" is closed; sequences allowed through the second S.
Equal number of a's and b's	$S \rightarrow aSbS \mid bSaS \mid \varepsilon$	Each a is paired with a b in either order.
$\{a^n b^m \mid n \neq m\}$	$S \rightarrow aSb \mid A \mid B$ $A \rightarrow aA \mid a$ $B \rightarrow bB \mid b$	Match as many pairs as possible, then add extra a's (A) or extra b's (B).
Any string over $\{a, b\}$	$S \rightarrow aS \mid bS \mid \varepsilon$	Right-linear (regular) grammar - regular $\subset$ context free.
Arithmetic expressions (using +, *, brackets)	$E \rightarrow E + T \mid T$ $T \rightarrow T * F \mid F$ $F \rightarrow (E) \mid a$	Levels give * higher precedence than +.

Table 4.4: Examples of CFLs and their grammars

Languages that are *not* context free include $\{a^n b^n c^n \mid n \geq 1\}$ and $\{ww \mid w \in \{a, b\}^*\}$; they need two independent counters or an exact copy, which one stack cannot provide.

Example 1: Show that $S \rightarrow aSb \mid \varepsilon$ generates $\{a^n b^n \mid n \geq 0\}$.

Applying $S \rightarrow aSb$ n times and then $S \rightarrow \varepsilon$ gives $S = aSb = aaSbb = \dots = a^n S b^n = a^n b^n$. No other string can be produced because every sentential form is of the form $a^k S b^k$ or $a^n b^n$. For example, $S = aSb = aaSbb = aabb$.

4.1.4 Leftmost and Rightmost Derivations

A sentential form usually has several variables, so there are many ways to order the replacements. Two orders are standard:

Derivation	Rule	Notation
Leftmost	At every step the leftmost variable is replaced.	$\Rightarrow_{lm}$
Rightmost (canonical)	At every step the rightmost variable is replaced.	$\Rightarrow_{rm}$

Table 4.5: Leftmost and rightmost derivations

Example 2: Consider the grammar $E \rightarrow E + T \mid T$, $T \rightarrow T * F \mid F$, $F \rightarrow (E) \mid a$. Give the leftmost and rightmost derivations of $a + a * a$.

Leftmost derivation:

$E \Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow a + T \Rightarrow a + T * F \Rightarrow a + F * F \Rightarrow a + a * F \Rightarrow a + a * a$

Rightmost derivation:

$E \Rightarrow E + T \Rightarrow E + T * F \Rightarrow E + T * a \Rightarrow E + F * a \Rightarrow E + a * a \Rightarrow T + a * a \Rightarrow F + a * a \Rightarrow a + a * a$

Both derivations use the same productions (the same tree) but in a different order. A string has a leftmost derivation if and only if it has a rightmost derivation, and both exist for every string of $L(G)$.

4.1.5 Derivation Tree (Parse Tree)

A **derivation tree** shows how a string is derived without fixing the order of replacements. For a CFG $G = (V, T, P, S)$ it is an ordered tree with these properties:

1. The **root** is labelled by the start symbol S .
2. Every **interior node** is labelled by a variable.
3. Every **leaf** is labelled by a terminal or by ϵ .
4. If an interior node is labelled A and its children, from left to right, are labelled $X_1, X_2, \dots, X_k$, then $A \rightarrow X_1 X_2 \dots X_k$ must be a production of P . (An ϵ -leaf must be the only child of its parent.)

The **yield** of the tree is the string obtained by reading the leaves from left to right.

Theorem: for a CFG, $S \Rightarrow^* \alpha$ if and only if there is a derivation tree with root S and yield α .

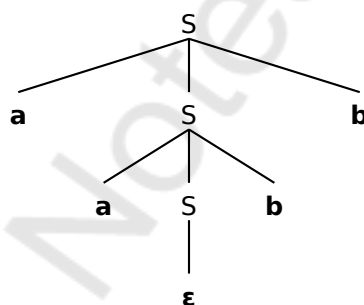


Fig. 4.1: Derivation tree of $aabb$ for $S \rightarrow aSb \mid \epsilon$ (yield = $aabb$, ϵ contributes nothing)

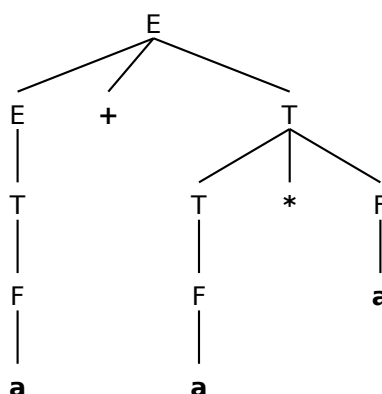


Fig. 4.2: Derivation tree of $a + a * a$ for the expression grammar of Example 2

Relationship between derivations and trees:

- A derivation tree represents *many* derivations (different orders of replacement) but exactly **one leftmost** and **one rightmost** derivation.
- The leftmost derivation corresponds to expanding the tree in pre-order (depth-first, left to right).
- Different derivation trees for the same string indicate ambiguity (Section 4.2).

Poly Notes Hub

4.2 Ambiguity in Context Free Grammar

4.2.1 Ambiguous Grammar and Parse Tree

A **parse tree** is the derivation tree used to give a string its structure; a compiler uses it to decide the meaning of a program. If one string can be parsed in two different ways its meaning is not unique, which is undesirable.

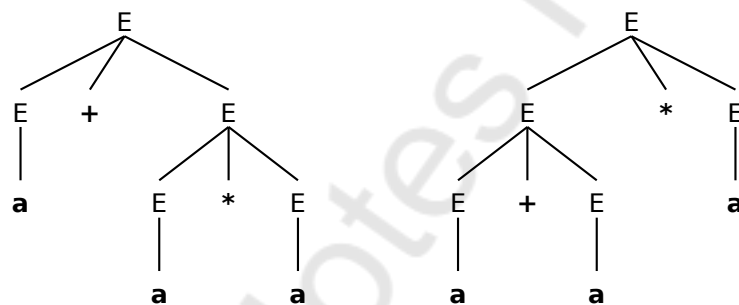
Definition. A context free grammar G is **ambiguous** if there is at least one string $w \in L(G)$ that has **more than one derivation tree**. Equivalently, w has more than one leftmost derivation (or more than one rightmost derivation). Otherwise G is **unambiguous**.

Example 3: Show that $E \rightarrow E + E \mid E * E \mid (E) \mid a$ is ambiguous.

Take the string $a + a * a$. It has two different leftmost derivations:

(i) $E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + E * E \Rightarrow a + a * E \Rightarrow a + a * a$

(ii) $E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow a + E * E \Rightarrow a + a * E \Rightarrow a + a * a$



Tree 1: $a + (a * a)$

Tree 2: $(a + a) * a$

Fig. 4.3: Two different derivation trees for $a + a * a$

Tree 1 evaluates the product first ($a + (a * a)$); Tree 2 evaluates the sum first ($(a + a) * a$). For $a = 2$ the two trees give 6 and 8. Hence the grammar is ambiguous.

More examples of ambiguity

Grammar	Ambiguous string	Reason
$S \rightarrow SS \mid a$	aaa	$S = SS$ can group as $(aa)a$ or $a(aa)$.
$S \rightarrow aS \mid Sa \mid a$	aa	The second a can come from aS or from Sa .
$S \rightarrow iEtS \mid iEtSeS \mid a, E \rightarrow b$ (dangling else)	ibtibtaea	The else can belong to the inner or the outer if.
$A \rightarrow A + A \mid a$	$a + a + a$	No fixed associativity.

Table 4.6: Ambiguous grammars

Important facts

- Ambiguity is a property of the *grammar*, not of the language: one language may have both ambiguous and unambiguous grammars.

- There is **no algorithm** that decides, for an arbitrary CFG, whether it is ambiguous (the problem is undecidable), and no algorithm that always removes ambiguity.
- A CFL is **inherently ambiguous** if *every* grammar for it is ambiguous. An example is $\{a^i b^j c^k \mid i = j \text{ or } j = k\}$.

4.2.2 Removal of Ambiguity

Ambiguity can often (not always) be removed by rewriting the grammar so that it generates the same language but only one tree per string. The general techniques are:

Technique	How it works
Precedence levels	Use a separate variable for each priority level; operators that bind tighter are produced deeper in the tree.
Left recursion	$A \rightarrow A \text{ op } B \mid B$ forces left associativity ($a + b + c = (a + b) + c$).
Right recursion	$A \rightarrow B \text{ op } A \mid B$ forces right associativity (used for exponent, assignment).
Matching rule	Introduce variables that force a unique choice, e.g. each else matches the nearest unmatched if.
Remove redundant productions	Delete rules that produce the same strings in two ways, e.g. keep only $S \rightarrow aS \mid a$ in place of $S \rightarrow aS \mid Sa \mid a$.

Table 4.7: Techniques for removing ambiguity

Example 4: Remove the ambiguity from $E \rightarrow E + E \mid E * E \mid (E) \mid a$, giving $*$ higher precedence than $+$ and making both left associative.

Use one variable per precedence level – E (sums), T (products) and F (factors):

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid a$$

Now $a + a * a$ has only one tree (shown earlier in Fig. 4.2): the product $a * a$ must be formed by T before it can be added to E, so $*$ binds tighter. Because $E \rightarrow E + T$ and $T \rightarrow T * F$ are left recursive, $a + a + a$ is grouped as $(a + a) + a$.

Example 5: Remove the ambiguity of the dangling-else grammar $S \rightarrow iEtS \mid iEtSeS \mid a$, $E \rightarrow b$.

Rule: an *else* is matched with the **closest unmatched then**. Split statements into *matched* (M) and *unmatched* (U) statements:

$$S \rightarrow M \mid U$$

$$M \rightarrow iEtMeM \mid a$$

$$U \rightarrow iEtS \mid iEtMeU$$

$$E \rightarrow b$$

In a matched statement M every if already has its else, so the only way to attach an else is to the nearest unmatched if. The string $ibtibtaea$ now has a unique parse, in which the else belongs to the inner if.

4.3 Simplification of Context Free Grammars

A grammar may contain symbols and productions that do nothing useful: variables that can never produce a terminal string, variables that can never be reached, rules such as $A \rightarrow B$ that only rename, and ϵ -rules. Removing them gives a smaller, equivalent grammar that is easier to analyse and is needed before converting to normal forms. The simplification does not change the language (except that ϵ must be handled separately).

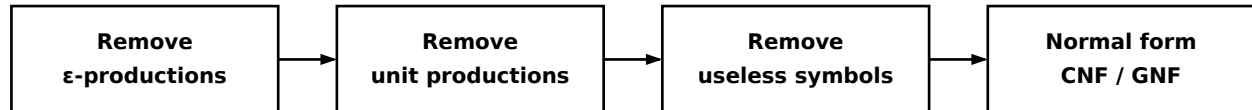


Fig. 4.4: Recommended order of simplification

Why the order matters: removing ϵ -productions can create new unit productions, and removing unit productions can leave symbols unreachable; so apply the steps in the order $\epsilon \rightarrow \text{unit} \rightarrow \text{useless}$. Each step is explained below (useless symbols first, as in the syllabus).

4.3.1 Removal of Useless Symbols

A symbol X (variable or terminal) is **useful** if there is a derivation $S \Rightarrow^* \alpha X \beta \Rightarrow^* w$ for some terminal string w . Otherwise it is **useless**. A symbol is useful only if it satisfies *both* conditions:

Condition	Meaning	Violated by
Generating	$X \Rightarrow^* w$ for some $w \in T^*$	Non-generating variable (never ends in terminals)
Reachable	$S \Rightarrow^* \alpha X \beta$ for some α, β	Unreachable variable (cannot be obtained from S)

Table 4.8: Conditions for a useful symbol

Algorithm (the two steps must be done in this order):

1. **Find generating variables.** Every variable with a production whose right side contains only terminals is generating. Repeatedly add a variable A if it has a production $A \rightarrow \alpha$ in which every symbol is a terminal or an already-generating variable. Delete all non-generating variables and every production that contains them.
2. **Find reachable symbols.** Starting from S , mark every symbol on the right side of the productions of marked variables (draw the dependency graph). Delete all unmarked symbols and their productions.

Example 6: Eliminate useless symbols from $S \rightarrow aS \mid A \mid C, A \rightarrow a, B \rightarrow aa, C \rightarrow aCb$.

Step 1 - generating: $A \rightarrow a$ and $B \rightarrow aa$ give $\{A, B\}$; $S \rightarrow A$ then gives S . $C \rightarrow aCb$ always contains C , so C never ends in terminals - it is **non-generating**. Delete C , the production $S \rightarrow C$ and $C \rightarrow aCb$:

$$S \rightarrow aS \mid A$$

$$A \rightarrow a$$

$$B \rightarrow aa$$

Step 2 - reachable: from S we reach S and A (and the terminal a). B is not reachable, so delete it.

Result:

$$S \rightarrow aS \mid A$$

$$A \rightarrow a$$

Example 7: Eliminate useless symbols from $S \rightarrow AB \mid CA$, $A \rightarrow a$, $B \rightarrow BC \mid AB$, $C \rightarrow aB \mid b$.

- **Generating:** A ($A \rightarrow a$), C ($C \rightarrow b$), then S ($S \rightarrow CA$). B only produces strings containing B ($B \rightarrow BC$, $B \rightarrow AB$) so B is non-generating.
- Delete B and every production with B : $S \rightarrow AB$, $B \rightarrow BC$, $B \rightarrow AB$, $C \rightarrow aB$. Remaining: $S \rightarrow CA$, $A \rightarrow a$, $C \rightarrow b$.
- **Reachable from S :** S , C , A - all reachable.

Result: $S \rightarrow CA$, $A \rightarrow a$, $C \rightarrow b$.

Caution: if you check reachability first, you may keep a variable that becomes unreachable only after non-generating variables are deleted.

4.3.2 Removal of Unit Productions

A **unit production** is a production of the form $A \rightarrow B$ where both A and B are variables. It only renames a variable and increases the length of derivations.

Algorithm:

1. For every variable A , find all variables B with $A \Rightarrow^* B$ using unit productions only (the “unit closure” of A ; A itself is included).
2. For every such B and every *non-unit* production $B \rightarrow \alpha$, add $A \rightarrow \alpha$.
3. Delete all unit productions.

Example 8: Remove the unit productions from $S \rightarrow Aa \mid B$, $B \rightarrow A \mid bb$, $A \rightarrow a \mid bc \mid B$.

Unit productions: $S \rightarrow B$, $B \rightarrow A$, $A \rightarrow B$. **Unit closures:** S : $\{S, B, A\}$; A : $\{A, B\}$; B : $\{B, A\}$.

Non-unit productions: $S \rightarrow Aa$; $B \rightarrow bb$; $A \rightarrow a \mid bc$.

Variable	Reachable by units	New productions (non-unit ones of the closure)
S	S, B, A	$S \rightarrow Aa \mid bb \mid a \mid bc$
A	A, B	$A \rightarrow a \mid bc \mid bb$
B	B, A	$B \rightarrow bb \mid a \mid bc$

Table 4.9: Unit production removal

Now B is no longer reachable from S , so it is a useless symbol and can be deleted.

Result:

$$S \rightarrow Aa \mid bb \mid a \mid bc$$

$$A \rightarrow a \mid bc \mid bb$$

4.3.3 Removal of ϵ -Productions

An **ϵ -production** is a production $A \rightarrow \epsilon$. A variable A is **nullable** if $A \Rightarrow^* \epsilon$. If $\epsilon \in L(G)$ it is impossible to remove all ϵ -productions without changing the language, so the theorem is stated for $L(G) - \{\epsilon\}$.

Algorithm:

1. **Find nullable variables:** every A with $A \rightarrow \varepsilon$ is nullable; if $A \rightarrow X_1 \dots X_k$ and all X_i are nullable, then A is nullable. Repeat until nothing changes.
2. For every production $A \rightarrow X_1 \dots X_k$, add all productions obtained by **deleting any subset of the nullable symbols** from the right side (a production with an empty right side is not added).
3. Delete all ε -productions.
4. If $\varepsilon \in L(G)$ and an ε -production for the start symbol is needed, add a new start symbol $S' \rightarrow S \mid \varepsilon$.

Example 9: Remove the ε -productions from $S \rightarrow ABA$, $A \rightarrow aA \mid \varepsilon$, $B \rightarrow bB \mid \varepsilon$.

Nullable: A and B (by $A \rightarrow \varepsilon$, $B \rightarrow \varepsilon$) and S (since $S \rightarrow ABA$ and A, B are nullable): $\{S, A, B\}$.

Production	Variants after deleting nullable symbols	Kept
$S \rightarrow ABA$	ABA, BA (drop first A), AA (drop B), AB (drop last A), B (drop both A 's), A (drop first A and B , or B and last A), ε (drop all)	$ABA \mid BA \mid AA \mid AB \mid B \mid A$
$A \rightarrow aA$	aA, a	$aA \mid a$
$B \rightarrow bB$	bB, b	$bB \mid b$

Table 4.10: Removal of ε -productions

Result (generates $L(G) - \{\varepsilon\}$):

$$S \rightarrow ABA \mid AB \mid BA \mid AA \mid A \mid B$$

$$A \rightarrow aA \mid a$$

$$B \rightarrow bB \mid b$$

Since $\varepsilon \in L(G)$, to preserve the language exactly we may add a new start symbol $S' \rightarrow S \mid \varepsilon$.

Combined example

Example 10: Simplify $S \rightarrow ASB \mid \varepsilon$, $A \rightarrow aAS \mid a$, $B \rightarrow SbS \mid A \mid bb$.

1. **ε -productions.** Only S is nullable. Replacing S in each production (deleting it or keeping it):
 $S \rightarrow ASB \mid AB$; $A \rightarrow aAS \mid aA \mid a$; $B \rightarrow SbS \mid bS \mid Sb \mid b \mid A \mid bb$.
2. **Unit productions.** $B \rightarrow A$ is the only one; closure of $B = \{B, A\}$. Add the non-unit productions of A to B and delete $B \rightarrow A$:
 $B \rightarrow SbS \mid bS \mid Sb \mid b \mid bb \mid aAS \mid aA \mid a$.
3. **Useless symbols.** A, B and S are all generating ($A \rightarrow a$, $B \rightarrow b$, $S \rightarrow AB$) and all reachable from S . Nothing is removed.

Simplified grammar (for $L(G) - \{\varepsilon\}$):

$$S \rightarrow ASB \mid AB$$

$$A \rightarrow aAS \mid aA \mid a$$

$$B \rightarrow SbS \mid bS \mid Sb \mid b \mid bb \mid aAS \mid aA \mid a$$

4.4 Normal Forms of Context Free Grammars

A **normal form** restricts the shape of productions without reducing the power of the grammar. Every CFL (not containing ϵ) can be generated by a grammar in either of the two normal forms studied here. Before conversion, the grammar must be simplified (ϵ -productions, unit productions and useless symbols removed).

4.4.1 Chomsky Normal Form (CNF)

A CFG is in **Chomsky normal form** if every production has one of the forms:

- $A \rightarrow BC$ (right side is exactly two variables), or
- $A \rightarrow a$ (right side is a single terminal).

(The production $S \rightarrow \epsilon$ is allowed only if $\epsilon \in L$ and S does not appear on any right side.)

Theorem: every CFL without ϵ is generated by a grammar in CNF.

Conversion algorithm:

1. **Simplify** the grammar: remove ϵ -productions, unit productions and useless symbols.
2. **Terminals in long right sides:** in every production with right side of length ≥ 2 , replace each terminal a by a new variable C_a and add the production $C_a \rightarrow a$.
3. **Long right sides:** replace a production $A \rightarrow B_1 B_2 \dots B_k$ ($k \geq 3$) by the chain $A \rightarrow B_1 D_1$, $D_1 \rightarrow B_2 D_2$, ..., $D_{k-2} \rightarrow B_{k-1} B_k$, where D_i are new variables.

Example 11: Convert $S \rightarrow aSb \mid ab$ into CNF.

- No ϵ , unit or useless productions are present.
- **Step 2:** $S \rightarrow C_a S C_b \mid C_a C_b$, $C_a \rightarrow a$, $C_b \rightarrow b$.
- **Step 3:** $S \rightarrow C_a S C_b$ has length 3: replace it by $S \rightarrow C_a D$ and $D \rightarrow S C_b$.

CNF grammar:

$$\begin{aligned} S &\rightarrow C_a D \mid C_a C_b \\ D &\rightarrow S C_b \\ C_a &\rightarrow a \\ C_b &\rightarrow b \end{aligned}$$

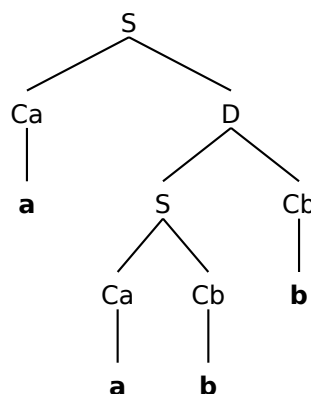


Fig. 4.5: CNF derivation tree of aabb – every interior node has two children (or one terminal child)

Example 12: Convert to CNF: $S \rightarrow bA \mid aB$, $A \rightarrow bAA \mid aS \mid a$, $B \rightarrow aBB \mid bS \mid b$.

- **Step 2** (terminals): introduce $C_a \rightarrow a$ and $C_b \rightarrow b$:
 $S \rightarrow C_b A \mid C_a B$; $A \rightarrow C_b AA \mid C_a S \mid a$; $B \rightarrow C_a BB \mid C_b S \mid b$.
- **Step 3** (length 3): $A \rightarrow C_b AA$ becomes $A \rightarrow C_b D_1$, $D_1 \rightarrow AA$; $B \rightarrow C_a BB$ becomes $B \rightarrow C_a D_2$, $D_2 \rightarrow BB$.

CNF grammar:

$S \rightarrow C_b A \mid C_a B$
 $A \rightarrow C_b D_1 \mid C_a S \mid a$
 $B \rightarrow C_a D_2 \mid C_b S \mid b$
 $D_1 \rightarrow AA, D_2 \rightarrow BB$
 $C_a \rightarrow a, C_b \rightarrow b$

Useful property: in a CNF grammar, every derivation tree is a binary tree, and a string of length $n \geq 1$ is derived in exactly $2n - 1$ steps ($n - 1$ steps of type $A \rightarrow BC$ and n steps of type $A \rightarrow a$). This is used by the CYK parsing algorithm.

4.4.2 Greibach Normal Form (GNF)

A CFG is in **Greibach normal form** if every production has the form

$A \rightarrow a\alpha$, where a is a terminal and $\alpha \in V^*$ (a string of zero or more variables).

Every production starts with exactly one terminal followed by variables. **Theorem:** every CFL without ϵ is generated by a grammar in GNF. In GNF every step of a leftmost derivation produces one new terminal, so a string of length n is derived in exactly **n steps**; this makes GNF useful for building pushdown automata and for top-down parsing.

Two lemmas used in the conversion:

Lemma	Statement
Substitution	If $A \rightarrow B\gamma$ and $B \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_k$ are productions, replace $A \rightarrow B\gamma$ by $A \rightarrow \beta_1\gamma \mid \beta_2\gamma \mid \dots \mid \beta_k\gamma$.
Left-recursion removal	If $A \rightarrow A\alpha_1 \mid \dots \mid A\alpha_r \mid \beta_1 \mid \dots \mid \beta_s$ (β 's do not start with A), replace by $A \rightarrow \beta_i \mid \beta_i Z$ and $Z \rightarrow \alpha_j \mid \alpha_j Z$, where Z is a new variable.

Table 4.11: Lemmas for GNF conversion

Conversion algorithm:

1. Simplify the grammar and convert it to CNF (or at least remove ϵ , unit productions and useless symbols).
2. Rename the variables $A_1, A_2, \dots, A_n$ ($A_1 = S$).
3. Modify the productions so that every $A_i \rightarrow A_j\gamma$ has **$j > i$** : substitute when $j < i$, and remove left recursion ($j = i$) using new variables Z_i .
4. Now A_n productions start with a terminal. Substitute **backwards** into $A_{n-1}, \dots, A_1$ so that all of them begin with a terminal.
5. Substitute into the Z -productions so that they also begin with a terminal.

Example 13: Convert to GNF: $S \rightarrow AB$, $A \rightarrow aA \mid bB \mid b$, $B \rightarrow b$.

A and B already start with terminals. Only $S \rightarrow AB$ starts with a variable: substitute the A -productions into it.

GNF grammar:

$$\begin{aligned} S &\rightarrow aAB \mid bBB \mid bB \\ A &\rightarrow aA \mid bB \mid b \\ B &\rightarrow b \end{aligned}$$
Example 14: Convert to GNF: $S \rightarrow Sa \mid b$.

$S \rightarrow Sa$ is left recursive with $\alpha = a$ and $\beta = b$. Apply the lemma with a new variable Z :

$$\begin{aligned} S &\rightarrow b \mid bZ \\ Z &\rightarrow a \mid aZ \end{aligned}$$

This generates ba^* , the same language as $S \rightarrow Sa \mid b$.

Example 15: Convert to GNF: $S \rightarrow AA \mid a$, $A \rightarrow SS \mid b$.

Rename $S = A_1$, $A = A_2$: $A_1 \rightarrow A_2A_2 \mid a$; $A_2 \rightarrow A_1A_1 \mid b$.

- $A_1 \rightarrow A_2A_2$ has $2 > 1$ - allowed.
- $A_2 \rightarrow A_1A_1$ has $1 < 2$: substitute $A_1 \rightarrow A_2A_2 \mid a$ to get $A_2 \rightarrow A_2A_2A_1 \mid aA_1 \mid b$.
- This is left recursive. Removing it with Z : $A_2 \rightarrow aA_1 \mid b \mid aA_1Z \mid bZ$; $Z \rightarrow A_2A_1 \mid A_2A_1Z$.
- Substitute A_2 into $A_1 \rightarrow A_2A_2$ and into the Z -productions.

GNF grammar ($A_1 = S$, $A_2 = A$):

$$\begin{aligned} S &\rightarrow aSA \mid bA \mid aSZA \mid bZA \mid a \\ A &\rightarrow aS \mid b \mid aSZ \mid bZ \\ Z &\rightarrow aSS \mid bS \mid aSZS \mid bZS \mid aSSZ \mid bSZ \mid aSZSZ \mid bZSZ \end{aligned}$$

Check: $S \Rightarrow AA \Rightarrow bA \Rightarrow bb$ in the original grammar; in GNF, $S \Rightarrow bA \Rightarrow bb$ (using $A \rightarrow b$). Both give bb .

4.4.3 Comparison of CNF and GNF

Feature	Chomsky normal form	Greibach normal form
Production form	$A \rightarrow BC$ or $A \rightarrow a$	$A \rightarrow a\alpha$ ($\alpha \in V^*$)
Derivation tree	Binary tree	Not necessarily binary
Steps for string of length n	$2n - 1$	n
Left recursion	May be present	Not present
Main use	CYK parsing, proofs (pumping lemma for CFLs)	Constructing PDA, top-down parsing
Needs	Simplified grammar	Simplified grammar + left recursion removal

Table 4.12: CNF versus GNF

Chapter Summary

Topic	Key point
CFG	$G = (V, T, P, S)$ with productions $A \rightarrow \alpha$, A a single variable.
Derivation	Leftmost replaces the leftmost variable; rightmost the rightmost; a tree gives exactly one of each.
Ambiguity	Some string has two or more parse trees; removed by precedence levels / associativity; undecidable in general.
Useless symbols	Remove non-generating, then unreachable symbols.
Unit productions	$A \rightarrow B$: replace by the non-unit productions of B .
ϵ -productions	Find nullable variables, add all variants, delete $A \rightarrow \epsilon$.
CNF	$A \rightarrow BC \mid a$; $2n - 1$ steps.
GNF	$A \rightarrow a\alpha$; n steps; remove left recursion first.

Table 4.13: Chapter 4 at a glance

Multiple Choice Questions (15)

Choose the correct option for each question. Answers are given in the answer key that follows.

- Q1.** In a context free grammar, the left side of every production is
(a) a terminal (b) a single variable
(c) a string of terminals (d) any string of symbols
- Q2.** Which grammar generates $\{a^n b^n \mid n \geq 0\}$?
(a) $S \rightarrow aSb \mid ab$ (b) $S \rightarrow aS \mid bS \mid \epsilon$
(c) $S \rightarrow SS \mid a$ (d) $S \rightarrow aSb \mid \epsilon$
- Q3.** A derivation in which the leftmost variable is replaced at every step is called
(a) rightmost derivation (b) canonical derivation
(c) leftmost derivation (d) none of these
- Q4.** The yield of a derivation tree is
(a) the string formed by the leaves read from left to right (b) the label of the root
(c) the number of interior nodes (d) the height of the tree
- Q5.** A CFG is ambiguous if some string of its language has
(a) no derivation (b) exactly one derivation tree
(c) more than one derivation tree (d) an infinite derivation
- Q6.** The ambiguity of $E \rightarrow E + E \mid E * E \mid a$ is removed by
(a) adding ϵ -productions (b) introducing a separate variable for each precedence level
(c) converting to GNF (d) deleting the terminals
- Q7.** Which of the following languages is **not** context free?
(a) $\{a^n b^n\}$ (b) $\{ww^R\}$
(c) $\{a^n b^n c^n \mid n \geq 1\}$ (d) balanced parentheses
- Q8.** A symbol is useless if it is
(a) either non-generating or unreachable from S (b) nullable
(c) a terminal (d) on the right side of a unit production
- Q9.** In $S \rightarrow ABA$, $A \rightarrow aA \mid \epsilon$, $B \rightarrow bB \mid \epsilon$, the nullable variables are
(a) $\{A\}$ (b) $\{A, B\}$
(c) $\{S\}$ (d) $\{S, A, B\}$
- Q10.** Which of the following is a unit production?
(a) $A \rightarrow a$ (b) $A \rightarrow BC$
(c) $A \rightarrow \epsilon$ (d) $A \rightarrow B$

Q11. In Chomsky normal form every production is of the form

- (a) $A \rightarrow aB$
- (b) $A \rightarrow BC$ or $A \rightarrow a$
- (c) $A \rightarrow a\alpha$
- (d) $A \rightarrow B$

Q12. The number of derivation steps needed for a string of length $n \geq 1$ in a CNF grammar is

- (a) n
- (b) $n + 1$
- (c) $2n - 1$
- (d) $2n$

Q13. Greibach normal form requires every production to be of the form

- (a) $A \rightarrow a\alpha$ with a a terminal and α a string of variables
- (b) $A \rightarrow BC$
- (c) $A \rightarrow a$ only
- (d) $A \rightarrow \epsilon$

Q14. The recommended order for simplifying a CFG is

- (a) useless $\rightarrow$ unit $\rightarrow \epsilon$
- (b) unit $\rightarrow \epsilon \rightarrow$ useless
- (c) useless $\rightarrow \epsilon \rightarrow$ unit
- (d) $\epsilon \rightarrow$ unit $\rightarrow$ useless

Q15. Deciding whether an arbitrary CFG is ambiguous is

- (a) always decidable
- (b) undecidable in general
- (c) solved by the CYK algorithm
- (d) possible only for regular grammars

Answer Key

Q.	Ans.	Q.	Ans.	Q.	Ans.
1	(b)	6	(b)	11	(b)
2	(d)	7	(c)	12	(c)
3	(c)	8	(a)	13	(a)
4	(a)	9	(d)	14	(d)
5	(c)	10	(d)	15	(b)

Table 4.14: Answer key for MCQs

Poly Notes Hub

Broad Questions (10)

Q1. Define a context free grammar. Write context free grammars for (i) $\{a^n b^{2n} \mid n \geq 1\}$, (ii) the set of palindromes over $\{a, b\}$, and (iii) strings over $\{a, b\}$ having equal numbers of a's and b's.

Q2. Explain leftmost derivation, rightmost derivation and derivation tree. For the grammar $S \rightarrow aB \mid bA$, $A \rightarrow a \mid aS \mid bAA$, $B \rightarrow b \mid bS \mid aBB$, obtain the leftmost and rightmost derivations and the derivation tree of the string aabbab.

Q3. Define an ambiguous grammar. Show that $E \rightarrow E + E \mid E * E \mid (E) \mid id$ is ambiguous by drawing two different derivation trees for $id + id * id$.

Q4. Explain the methods of removing ambiguity. Convert the grammar $E \rightarrow E + E \mid E * E \mid (E) \mid id$ into an unambiguous grammar giving $*$ higher precedence than $+$, and write an unambiguous grammar for the dangling-else problem.

Q5. What are useless symbols? Describe the algorithm to remove them and simplify the grammar $S \rightarrow AB \mid CA$, $A \rightarrow a$, $B \rightarrow BC \mid AB$, $C \rightarrow aB \mid b$.

Q6. What is a unit production? Eliminate the unit productions from $S \rightarrow Aa \mid B$, $B \rightarrow A \mid bb$, $A \rightarrow a \mid bc \mid B$.

Q7. Define nullable variable and ϵ -production. Remove the ϵ -productions from $S \rightarrow ABC$, $A \rightarrow BB \mid \epsilon$, $B \rightarrow CC \mid a$, $C \rightarrow AA \mid b$.

Q8. Simplify the grammar $S \rightarrow AaB \mid aaB$, $A \rightarrow \epsilon$, $B \rightarrow bbA \mid \epsilon$ by removing ϵ -productions, unit productions and useless symbols (state the order in which you apply them).

Q9. Define Chomsky normal form. Convert the grammar $S \rightarrow aAD$, $A \rightarrow aB \mid bAB$, $B \rightarrow b$, $D \rightarrow d$ into CNF. How many derivation steps are needed for a string of length n in CNF?

Q10. Define Greibach normal form and explain the conversion procedure with the two lemmas. Convert $A_1 \rightarrow A_2 A_3$, $A_2 \rightarrow A_3 A_1 \mid b$, $A_3 \rightarrow A_1 A_2 \mid a$ into GNF, and compare CNF with GNF.

— End of Chapter 4 —