Context-Free Grammars

Theory, Examples, and Proofs

Sara Sorahi

Introduction to the Theory of Computation

Why Context-Free Grammars?

Regular languages fail to describe:

- matching numbers (e.g. $a^n b^n$)
- nested structures
- hierarchical syntax

Finite automata have finite memory. Context-free grammars introduce **recursion**.

Definition of a Context-Free Grammar

A **context-free grammar (CFG)** is a 4-tuple:

$$G = (V, \Sigma, R, S)$$

- V: variables (nonterminals)
- Σ: terminals
- R: production rules
- S: start symbol

Form of Production Rules

Each rule has the form:

$$A \to \alpha$$

where:

- $A \in V$
- $\alpha \in (V \cup \Sigma)^*$

Only **one variable** appears on the left-hand side. This is why the grammar is *context-free*.

How CFGs Generate Strings

A **derivation** is a sequence of rule applications starting from the start symbol.

Notation:

$$S \Rightarrow \alpha \Rightarrow \beta \Rightarrow^* w$$

Example Grammar: $0^n\#1^n$

Grammar:

$$
\begin{aligned}
A &\to 0A1 \mid B \\
B &\to \#
\end{aligned}
$$

This grammar generates:

$$\{\, 0^n\#1^n \mid n \geq 0 \,\}$$

Worked Derivation Example

Derive 000#111:

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$$

Each recursive step adds one 0 and one 1.

Parse Trees

A **parse tree** represents the hierarchical structure of a derivation.

- Root: start symbol
- Internal nodes: variables
- Leaves: terminals

Balanced Parentheses

Grammar:

$$S \to (S)S \mid \varepsilon$$

Generates all properly nested parentheses.

Why This Grammar Works

- (S) enforces matching parentheses
- SS allows concatenation
- ε allows termination

This grammar captures recursive nesting.

CFGs vs Regular Languages

Feature	Regular	Context-Free
Memory	finite	stack
Nesting	no	yes
Matching counts	no	yes
Model	DFA	CFG / PDA

Every Regular Language Is Context-Free

Theorem. Every regular language is also context-free.

Proof idea:

- Start from a DFA for the regular language
- Create one variable for each DFA state
- Convert transitions into grammar rules

Proof (Construction)

For each transition $q \xrightarrow{a} r$, add:

$$Q \to aR$$

For each accepting state q, add:

$$Q \to \varepsilon$$

The grammar generates exactly the strings accepted by the DFA.

Ambiguity

A grammar is **ambiguous** if a string has two different parse trees.

Ambiguous grammar:

$$E \to E + E \mid E * E \mid (E) \mid a$$

Ambiguity Example

String:

$$a + a * a$$

Two different parse trees exist.

Different trees $\Rightarrow$ different meanings.

Removing Ambiguity

Unambiguous grammar:

$$
\begin{aligned}
E &\rightarrow E + T \mid T \\
T &\rightarrow T * F \mid F \\
F &\rightarrow (E) \mid a
\end{aligned}
$$

This grammar enforces operator precedence.

Chomsky Normal Form

Every context-free language has an equivalent grammar in **Chomsky Normal Form (CNF)**.

Allowed rules:

- $A \to BC$
- $A \to a$
- $S \to \varepsilon$ (optional)

Exercise 1: Identifying Components of a CFG

Consider the grammar G:

$$S \to aSb \mid \varepsilon$$

Tasks:

1. Identify the set of variables V
2. Identify the set of terminals Σ
3. Identify the start symbol

Solution to Exercise 1

From the grammar:

$$S \to aSb \mid \varepsilon$$

We have:

- Variables: $V = \{S\}$
- Terminals: $\Sigma = \{a, b\}$
- Start symbol: S

Variables are symbols that appear on the left-hand side of rules.

Exercise 2: Derivation

Consider the grammar:

$$S \to aSb \mid \varepsilon$$

Task: Give a derivation for the string `aaabbb`.

Solution to Exercise 2

Derivation:

$$
\begin{aligned}
S &\Rightarrow aSb \\
&\Rightarrow aaSbb \\
&\Rightarrow aaaSbbb \\
&\Rightarrow aaa\varepsilon bbb \\
&\Rightarrow aaabbb
\end{aligned}
$$

Each recursive step adds one $\mathtt{a}$ and one $\mathtt{b}$.

Exercise 3: Describe the Language

Given the grammar:

$$S \to (S)S \mid \varepsilon$$

Task: Describe the language generated by this grammar.

Solution to Exercise 3

The grammar generates:

- properly nested parentheses
- including the empty string

Examples:

$$\varepsilon,\ (),\ (()),\ ()(),\ (()())$$

The rule (S) enforces nesting, while SS allows concatenation.

Exercise 4: Regular or Context-Free?

Consider the language:

$$L = \{ a^n b^n \mid n \geq 0 \}$$

Tasks:

1. Is L regular?
2. Is L context-free?

Solution to Exercise 4

- L is **not regular** (requires matching counts)
- **L is context-free**

A CFG for L:

$$S \to aSb \mid \varepsilon$$

This grammar enforces equal numbers of a's and b's.

Exercise 5: Ambiguity

Consider the grammar:

$$E \rightarrow E + E \mid a$$

Task: Show that this grammar is ambiguous.

Solution to Exercise 5

Consider the string:

$$a + a + a$$

Two different parses:

- $(a + a) + a$
- $a + (a + a)$

The same string has two different parse trees, so the grammar is ambiguous.

Exercise 6: Proving Context-Freeness

Consider the language:

$$L = \{\, w\#w^R \mid w \in \{0,1\}^* \,\}$$

Task: Show that L is context-free.

Solution to Exercise 6

Idea:

- Generate w on the left
- Use $\#$ as a center marker
- Generate the reverse of w

One possible grammar:

$$S \to 0S0 \mid 1S1 \mid \#$$

Recursive rules enforce symmetry around $\#$.

Exercise 7: Design a CFG (Two Conditions)

Design a context-free grammar for the language:

$$L = \{\, a^i b^j c^k \mid i = j \text{ or } j = k,\ \ i, j, k \geq 0 \,\}$$

Task:

- Give a CFG for L
- Explain why it works

Solution to Exercise 7

We split the language into two parts:

$$L = \{a^i b^i c^k\} \cup \{a^i b^j c^j\}$$

Grammar:

$$\begin{aligned}
S &\to S_1 \mid S_2 \\
S_1 &\to aS_1b \mid C \\
C &\to cC \mid \varepsilon \\
S_2 &\to AD \\
A &\to aA \mid \varepsilon \\
D &\to bDc \mid \varepsilon
\end{aligned}$$

Context-free languages are closed under union, so splitting the language is valid.

Exercise 8: Prove a Language Is Not Context-Free

Consider the language:

$$L = \{\, a^n b^n c^n \mid n \geq 0 \,\}$$

Task: Prove that L is **not** context-free.

Solution to Exercise 8

Claim: L is not context-free.

Reasoning:

- Matching two counts (e.g. $a^n b^n$) is possible with CFGs
- Matching three independent counts is not

Formal proof requires the **pumping lemma for CFLs**.

This language exceeds the expressive power of context-free grammars.

Exercise 9: Ambiguity vs Unambiguity

Consider the grammar:

$$S \rightarrow SS \mid a$$

Tasks:

1. Describe $L(G)$
2. Show that the grammar is ambiguous

Solution to Exercise 9

Language:

$$L(G) = \{ a^n \mid n \geq 1 \}$$

Ambiguity: The string aaa can be derived in multiple ways:

- $(a)(aa)$
- $((a)a)a$

Each corresponds to a different parse tree.

Ambiguity is about structure, not just derivation length.

Exercise 10: Leftmost Derivations

Given the grammar:

$$S \rightarrow aSb \mid SS \mid \varepsilon$$

Tasks:

1. Give two different leftmost derivations of `aabb`
2. What does this tell you about the grammar?

Solution to Exercise 10

Leftmost derivation 1:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

Leftmost derivation 2:

$$S \Rightarrow SS \Rightarrow aSbS \Rightarrow aabb$$

Two different leftmost derivations imply that the grammar is ambiguous.

Exercise 11: CFG from DFA (Construction)

Given a DFA M recognizing:

$$L = \{\, w \in \{0,1\}^* \mid w \text{ ends with } 01 \,\}$$

Task: Construct a CFG that generates L.

Solution to Exercise 11

Let DFA states be q_0, q_1, q_2, where:

- q_0: start
- q_2: accepting

Grammar variables: Q_0, Q_1, Q_2
Rules (from transitions):

$$
\begin{aligned}
Q_0 &\to 0Q_0 \mid 1Q_1 \\
Q_1 &\to 0Q_2 \mid 1Q_1 \\
Q_2 &\to 0Q_0 \mid 1Q_1 \mid \varepsilon
\end{aligned}
$$

Accepting states produce ε to terminate derivations.

Exercise 12: Convert to Chomsky Normal Form

Convert the grammar below into Chomsky Normal Form:

$$S \to aSb \mid \varepsilon$$

Task: Give an equivalent grammar in CNF.

Solution to Exercise 12

Step 1: Introduce new variables for terminals:

$$A \to a, \quad B \to b$$

Step 2: Rewrite productions:

$$S \to ASB \mid \varepsilon$$

Step 3: Binzarize:

$$S \to AC \mid \varepsilon, \quad C \to SB$$

CNF restricts productions but preserves the language.

Exercise 13: Basic Parse Tree

Given the grammar:

$$S \to aSb \mid \varepsilon$$

Task:

- Draw the parse tree for the string `aabb`

Solution to Exercise 13

Derivation:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

Parse tree structure:

- Root: S
- First expansion: $S \to aSb$
- Second expansion: inner $S \to aSb$
- Final $S \to \varepsilon$

The tree is perfectly symmetric, reflecting equal numbers of a's and b's.

Exercise 14: Nested Structure

Given the grammar:

$$S \to (S)S \mid \varepsilon$$

Task: Draw the parse tree for the string:

$$()()$$

Solution to Exercise 14

High-level structure:

$$()() = () \cdot ()$$

Parse tree outline:

- Root expands using $S \to SS$
- Each S expands as (S)
- Inner S in each pair expands to ε

Concatenation is represented by sibling subtrees.

Exercise 15: Deeper Nesting

Using the same grammar:

$$S \to (S)S \mid \varepsilon$$

Task: Draw the parse tree for:

$$(()())$$

Solution to Exercise 15

Structural reasoning:

- The outermost parentheses correspond to one (S)
- Inside, S expands into ()()
- This requires both nesting and concatenation

Parse tree properties:

- deeper height than previous example
- inner S nodes branch into multiple subtrees

Parse trees make recursion depth explicit.

Exercise 16: Expression Grammar

Given the grammar:

$$
\begin{aligned}
E &\rightarrow E + T \mid T \\
T &\rightarrow T * F \mid F \\
F &\rightarrow (E) \mid a
\end{aligned}
$$

Task: Draw the parse tree for:

$$a + a * a$$

Pumping Lemma for CFLs (Statement)

Pumping Lemma (for Context-Free Languages).
If L is context-free, then there exists a pumping length $p \geq 1$ such that any string $s \in L$ with $|s| \geq p$ can be written as:

$$s = uvxyz$$

satisfying:

1. $|vxy| \leq p$
2. $|vy| \geq 1$
3. For all $i \geq 0$, $uv^ixy^iz \in L$

How it is used: Assume L is CFL, pick a long $s \in L$, and show that *some* pumping ($i = 0$ or $i = 2$) breaks membership, causing a contradiction.

Exercise 17: Prove a Language Is Not Context-Free

Consider the language:

$$L_1 = \{\, a^n b^n c^n \mid n \geq 0 \,\}$$

Task: Use the pumping lemma for CFLs to show that L_1 is **not** context-free.

Hint: choose $s = a^p b^p c^p$ and analyze where vxy can lie.

Solution to Exercise 17 (Key Case Analysis)

Assume (for contradiction) that L_1 is context-free. Let p be the pumping length. Choose:

$$s = a^p b^p c^p \in L_1$$

By the lemma, $s = uvxyz$ with $|vxy| \leq p$ and $|vy| \geq 1$.

Because $|vxy| \leq p$, the substring vxy cannot cover all three blocks of a's, b's, and c's. So vxy lies in one of these regions:

- entirely within a^p, or within b^p, or within c^p
- or crossing only one boundary: $a^p b^p$ or $b^p c^p$

Solution to Exercise 17 (Pump and Contradict)

We pump with $i = 0$ (delete v and y). Since $|vy| \geq 1$, at least one symbol is removed from the region where v and/or y occur.

Case 1: vxy is within one block (only a's or only b's or only c's). Then pumping changes exactly one count, so the three counts are no longer equal. Hence $uv^0xy^0z \notin L_1$.

Case 2: vxy crosses a/b or b/c. Then pumping changes the number of symbols in two adjacent blocks, but the third block remains p. Therefore the counts cannot all be equal after pumping. So again $uv^0xy^0z \notin L_1$.

In all cases, pumping breaks the required equality $|a| = |b| = |c|$. Contradiction $\Rightarrow$ L_1 is not context-free.

Exercise 18: Another Non-CFL Proof

Consider the language:

$$L_2 = \{\, 0^n 1^n 0^n \mid n \geq 0 \,\}$$

Task: Use the pumping lemma for CFLs to show that L_2 is **not** context-free.

Hint: choose $s = 0^p 1^p 0^p$. Think about what happens if you pump inside only one region, or across a boundary.

Solution to Exercise 18 (Setup)

Assume (for contradiction) that L_2 is context-free. Let p be the pumping length and choose:

$$s = 0^p 1^p 0^p \in L_2$$

Write $s = uvxyz$ as in the pumping lemma, with $|vxy| \leq p$ and $|vy| \geq 1$.

Because $|vxy| \leq p$, the substring vxy cannot include both the left 0^p and the right 0^p at the same time (they are separated by 1^p of length p). So vxy lies:

- entirely inside the left 0^p, or inside the middle 1^p, or inside the right 0^p, or
- crosses only one boundary: left 0^p/middle 1^p or middle 1^p/right 0^p

Solution to Exercise 18 (Pump and Contradict)

Pump down with $i = 0$: consider $uv^0xy^0z = uxz$.

Case 1: vxy is inside left 0^p. Then pumping changes the number of leading 0s but not the number of 1s and trailing 0s. So the first and last 0-block lengths cannot match n anymore. Thus $uxz \notin L_2$.

Case 2: vxy is inside middle 1^p. Then the number of 1s changes but both 0-blocks remain p. So $uxz \notin L_2$.

Case 3: vxy is inside right 0^p. Then trailing 0s change, but leading 0s and 1s do not. So $uxz \notin L_2$.

Case 4: vxy crosses a boundary (left 0s / 1s or 1s / right 0s). Pumping changes symbols in two adjacent regions but not the third, so it is impossible to keep the exact form $0^n1^n0^n$. Hence $uxz \notin L_2$.