# Formal Languages and Automata Theory

Homework 3: Context-Free Grammars (Implementation)

University of Bucharest

Due Date: One week from May 21, 2025

## Objectives

- Implement a parser for a context-free grammar (CFG)
- Perform leftmost and rightmost derivations programmatically
- Generate strings from a CFG
- Test if a string belongs to a language defined by a CFG

## Tools Allowed

- You may use any programming language (Python is recommended)
- You are allowed to use basic data structures but not external parsing libraries (e.g., NLTK, Lark, PLY)

## Tasks (Total: 110 Points)

**Task 1. Define a CFG** (15 points)

Create a programmatic representation of the following CFG:

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

- Define the non-terminals, terminals, start symbol, and production rules in your code.

**Task 2. String Generator** (25 points)

Implement a function that randomly generates strings from the CFG defined above. Your generator should:

- Generate up to 10 strings.

- Limit string length to at most 10 characters.

**Task 3. Derivation** (25 points)

Write a function that, given a target string, displays its derivation.

*Hint: use recursion or a stack to simulate the derivation steps.*

**Task 4. Membership Tester** (25 points)

Implement a recognizer function to check whether a given string belongs to the language of the CFG. Your function should:

- Return `True` or `False`
- Work correctly for strings of length up to 12

**Task 5. Bonus: Extend Your CFG** (+20 bonus points)

Extend the grammar to support the language:

$$L = \{a^n b^n c^n \mid n \geq 1\}$$

- Write a new CFG that generates this language.
- Implement a recognizer for it.
- Briefly explain (in a comment or README) why this language is not context-free (despite your implementation simulating recognition).

# Submission Guidelines

- Submit your source code files (e.g., `.py`, `.java`, `.cpp`)
- Include a `README.md` file with:
  - How to run your program
  - Example outputs
- Make sure the code is commented and clean
- Use a GitHub repository where the code will be stored **10 points ex-oficio**

# Grading Criteria

- Correctness and functionality
- Code readability and comments
- Proper use of CFG concepts
- Completion of all required tasks

*Academic Honesty:* You may discuss ideas with classmates but the code must be your own. Plagiarism will result in a zero grade.

# 1. Introduction

Context-Free Grammars (CFGs) are a fundamental concept in formal language theory and are used to describe programming languages, natural languages, and various formal languages. This breviar summarizes the theoretical background necessary to complete the implementation-based CFG homework.

# 2. Definitions

## 2.1 Context-Free Grammar

A **Context-Free Grammar (CFG)** is a 4-tuple:

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

where:

- $V$ is a finite set of variables (non-terminal symbols)
- $\Sigma$ is a finite set of terminal symbols ($V \cap \Sigma = \emptyset$)
- $R$ is a finite set of production rules of the form $A \to \alpha$, where $A \in V$ and $\alpha \in (V \cup \Sigma)^*$
- $S \in V$ is the start symbol

## 2.2 Language of a CFG

The language generated by a CFG $G$ is:

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$$

That is, all strings of terminal symbols derivable from $S$ using rules from $R$.

# 3. Derivations

## 3.1 Leftmost Derivation

In a **leftmost derivation**, at each step, the leftmost non-terminal is replaced using a production.

## 3.2 Rightmost Derivation

In a **rightmost derivation**, the rightmost non-terminal is replaced first at each step.

## 3.3 Derivation Notation

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aa\varepsilon bb = aabb$$

# 4. Example Grammar

Consider the grammar:

$$G = (\{S\}, \{a, b\}, R, S), \text{ where } R = \{S \to aSb \mid \varepsilon\}$$

This grammar generates strings with an equal number of $a$'s and $b$'s in the form $a^n b^n$.

### Language

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

# 5. String Generation

A CFG can be used to **generate strings** by applying production rules starting from the start symbol until a string of terminals is formed.

Example:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aa\varepsilon bb = aabb$$

# 6. Membership Testing

Given a CFG and a string, **membership testing** determines whether the string belongs to the language of the grammar. While some algorithms like CYK (Cocke–Younger–Kasami) can be used, recursive methods suffice for simple grammars.

# 7. Recognizers vs Parsers

- A **recognizer** returns a boolean indicating whether a string belongs to the language.
- A **parser** may return derivation steps or a parse tree.

# 8. The Bonus Task: $a^n b^n c^n$

The language:

$$L = \{a^n b^n c^n \mid n \geq 1\}$$

is **not context-free**. This is a classic example used to show the limitations of context-free grammars (can be proven using the pumping lemma for CFGs). However, using code, we can write a recognizer that simulates this check for small input sizes.

## 9. Practical Advice

- Avoid infinite loops in recursive derivation.
- Use maximum depth or length limits when generating strings.
- Represent grammar rules clearly, e.g., as Python dictionaries.

## 10. References


- Hopcroft, Ullman – *Introduction to Automata Theory, Languages, and Computation*
- Sipser – *Introduction to the Theory of Computation*
- Course slides and lecture notes