CS304: Automata and Formal Languages

Lec 13

Context-Free Grammars

Rachit Nimavat

September 1, 2025

Outline

The World of Regular Languages

Three Faces of Same Power

The Limit: What Can't Finite Automata Do?

Consider the language $L = \{a^n b^n \mid n \geq 0\} = \{\epsilon, ab, aabb, aaabbb, \dots\}$.

To accept L, a DFA/NFA must **count** the number of a's

DFA/NFA has only a finite number of states and cannot remember an arbitrarily large count n.

Next Step?

We need machine with more power:

- Context-Free Grammars: A way to describe languages with recursive structure
- Pushdown Automata: NFA + Stack

The Limits of Regular Languages

Finite Memory

DFAs lack the memory to handle unbounded counting or nested structures

Can't recognize languages like $\{a^n b^n \mid n \geq 0\}$

A New Tool: **Context-Free Grammar (CFG)**

To describe these more complex languages, we need a more powerful tool. Instead of a machine that just *recognizes* strings, we use a system of rules that *generates* these strings.

Example

Consider $L = \{a^n b^n \mid n \geq 0\}$ over $\Sigma = \{a, b\}$. Let's build a grammar!

$$S \rightarrow \varepsilon$$
$$S \rightarrow \mathsf{a}S\mathsf{b}$$

Shorthand: $S \rightarrow \varepsilon \mid \mathsf{a}S\mathsf{b}$

Derivation

is a sequence of steps applying the production rules. Generating string aabb:

$$\begin{aligned} S &\rightarrow \mathsf{a}S\mathsf{b} \\ &\rightarrow \mathsf{aa}S\mathsf{bb} \\ &\rightarrow \mathsf{aabb} \end{aligned}$$

Example - II

CFGs are excellent for defining the syntax of programming languages. Let's build a grammar for expressions over $\{a, b, c\}$.

$$E \to E + E \mid E * E \mid T$$
$$T \to \mathsf{a} \mid \mathsf{b} \mid \mathsf{c}$$

Derivations.

Generating $a * b$

$$\begin{aligned} E &\to E * E \\ &\to T * E \\ &\to T * T \\ &\to a * T \\ &\to a * b \end{aligned}$$

Generating $a + a$

$$\begin{aligned} E &\to E + E \\ &\to T + E \\ &\to T + T \\ &\to a + T \\ &\to a + a \end{aligned}$$

Generating $a + b * c$

$$\begin{aligned} E &\to E + E \\ &\to E + E * E \\ &\to T + E * E \\ &\to T + T * E \\ &\to T + T * T \\ &\to a + T * T \\ &\to a + b * T \\ &\to a + b * c \end{aligned}$$

Defn

A CFG is a set of recursive rules used to generate patterns of strings. Think of it as a blueprint for a language.

Definition 3.1

A Context-Free Grammar is a 4-tuple $G = (V, T, P, S)$, where:

- **V**: Finite set of **variables**. think: syntactic categories or placeholders. (e.g., S, E, T in previous examples)
- **T**: Finite set of **terminals** or **tokens**. actual symbols or words of the language. (e.g., $\mathsf{a}, \mathsf{b}, \mathsf{c}, +, *$ in previous examples)
- **P**: Finite set of **production rules**. Rules have the form $A \to \alpha$, where $A \in V$ and $\alpha \in (V \cup T)^*$, i.e., α is a string consisting of variables and terminals. specifies how to replace variables with strings of variables and terminals. (e.g., $S \to \mathsf{a}S\mathsf{b}$ and $E \to T$) in previous examples
- **S**: The **start symbol** ($S \in V$).

Example - III

Can you write a CFG for <u>every</u> regular language? Example: $\Sigma = 0, 1$ and $L = \Sigma^*$.

$$S \to 0S \mid 1S \mid \varepsilon$$

Derivations.

Generating ε

$$S \to \varepsilon$$

Generating 010

$$\begin{aligned} S &\to 0S \\ &\to 01S \\ &\to 010S \\ &\to 010 \end{aligned}$$

Generating 0

$$\begin{aligned} S &\to 0S \\ &\to 0 \end{aligned}$$

HW: Think about an algorithm to convert a DFA to a CFG.