

# Formal Grammar

YEGOR BUGAYENKO

Lecture #1 out of 10

80 minutes

The slidedeck was presented by the author in this [YouTube Video](#)

All visual and text materials presented in this slidedeck are either originally made by the author or taken from public Internet sources, such as web sites. Copyright belongs to their respected authors.




Notation

Chomsky Hierarchy

Parse Tree

Ambiguity

Non-determinism



I promise, there will be no more  
*formalism* than it's necessary!

## Chapter #1: Notation

By the way, if a language is simple, it's possible to do it without a grammar, for example (we just split the text by a space):

```
1 | PRINT 42
2 | PRINT 256
3 | PRINT 0
```

A practical example: I have a project **Xembly**, which is using ANTLR4 for parsing its own language:

```
1 | XPATH "/car/price";  
2 | SET "$2000";  
3 | ATTR "time", "2023/02/01";
```

However, I have a task in the backlog: get rid of the grammar and use string manipulations instead, because it's faster.

A *grammar* is a finite set of formal rules for generating (!) syntactically correct sentences. Pay attention to the word “formal.” A grammar may be informal, if the rules are informal. For example:

*“Commands go one after another sometimes with arguments”*

This is a rule, but it is not formal and may not be understood by a computer.

Assume, we want to create a new programming language (very similar to Basic), which will allow us to write programs that look like this:

```
1 | 10 PRINT "What is your name?"  
2 | 20 INPUT X  
3 | 30 PRINT "Hello,", X
```

It's impossible (or very hard) to parse this program by splitting strings, for example, because of the possible commas inside the "Hello," string.



A formal *grammar*  $G$ , according to Noam Chomsky (1956), is a tuple  $\langle N, T, P, S \rangle$ , where:

- $N = \{P_{\text{rogram}}, L_{\text{ine}}, N_{\text{umber}}, C_{\text{ommand}}, A_{\text{rgument}}, \dots\}$  (*non-terminals or variables*)
- $T = \{10, 20, \text{PRINT}, X, ,, \text{"Hello"}, \dots\}$  (*terminals or alphabet*)
- $P = \{\dots\}$  (*production rules*)
- $S \in N$  (*start symbol*)

By the way,  $N \cap T = \emptyset$ .

A *language* that can be built by  $G$  is denoted as  $\mathbf{L}(G)$ : set of all strings that can be generated by  $G$ .

A *production rule* specifies a replacement of its *left-hand side* with its *right-hand side*, for example:

$$1. L_{\text{ine}} \rightarrow N_{\text{umber}} \text{ INPUT } A_{\text{rgument}}$$

$$2. N_{\text{umber}} \rightarrow 10$$

$$3. N_{\text{umber}} \rightarrow 20$$

Formally, a production rule is (using *Kleene star*, by Stephen Kleene):

$$\begin{aligned} (T \cup N)^* n (T \cup N)^* &\rightarrow (T \cup N)^* \quad n \in N \\ V^* n V^* &\rightarrow V^* \quad V = (T \cup N) \end{aligned}$$

Each left-hand side must contain at least one non-terminal symbol.

Grammars are said to be *equivalent* if they produce the same language.

Chapter #2:

# Chomsky Hierarchy

There are four types in Chomsky Hierarchy of grammars:

Type-0: Unrestricted grammars

Type-1: Context-sensitive grammars

Type-2: Context-free grammars

Type-3: Regular grammars

## Type-0: Unrestricted Grammar

The only restriction is that  $\alpha$  is not empty (not  $\epsilon$ ) in each rule:

$$\alpha \rightarrow \beta \quad \alpha, \beta \in N \cup T$$

For every unrestricted grammar  $G$  there exists some Turing machine capable of recognizing  $L(G)$  and vice versa.

[ [Unrestricted](#) CSG CFG Regular ]

The decision problem of whether a given string  $s$  can be generated by a given unrestricted grammar is equivalent to the problem of whether it can be accepted by the *Turing machine* equivalent to the grammar. The latter problem is called the *Halting problem* and is undecidable.



## Type-1: Context-Sensitive Grammar

A *context-sensitive grammar* (CSG) are “non-erasing” grammars. A grammar is *noncontracting* (or *monotonic*) if all of its production rules are of the form  $\alpha \rightarrow \beta$  where the length of  $\alpha$  is less than or equal to that of  $\beta$ .

Some textbooks define CSGs as non-contracting, although this is not how Noam Chomsky defined them in 1959.

A canonical example is  $\{a^n b^n c^n : n \geq 1\}$ .

[ Unrestricted CSG [CFG](#) Regular ]

## Type-2: Context Free Grammar

A *context-free grammar* (CFG) is a grammar in which the left-hand side of each production rule consists of only a single non-terminal symbol, for example:

$$p_1: P_{\text{rogram}} \rightarrow P_{\text{rogram}} L_{\text{ine}}$$

$$p_2: P_{\text{rogram}} \rightarrow \epsilon$$

$$p_3: L_{\text{ine}} \rightarrow I_{\text{nteger}} C_{\text{ommand}} T_{\text{ail}}$$

$$p_4: T_{\text{ail}} \rightarrow T_{\text{ail}} A_{\text{rgument}}$$

$$p_5: T_{\text{ail}} \rightarrow \epsilon$$

$$p_6: I_{\text{nteger}} \rightarrow 10$$

$$p_7: I_{\text{nteger}} \rightarrow 20$$

[ Unrestricted CSG [CFG](#) Regular ]

Derivation process may be described using  $\Rightarrow_{p_i}$  notation:

$$\begin{aligned}
 P &\Rightarrow_{p_1} \mathbf{P L} \\
 &\Rightarrow_{p_3} P \mathbf{I C T} \\
 &\Rightarrow_{p_8} P 30 C T \\
 &\Rightarrow_{p?} P 30 \text{ PRINT } T \\
 &\Rightarrow_{p_1} \mathbf{P L} 30 \text{ PRINT } T \\
 &\Rightarrow_{p?} \dots
 \end{aligned}$$

[ Unrestricted CSG [CFG](#) Regular ]

We can say that “ $G$  derives in zero or more steps”:  $\xRightarrow{*}_G$  (it is *reflexive transitive closure* of  $\Rightarrow_G$ ). For example:

$$P \xRightarrow{*}_G P \ L \ 30 \ \text{PRINT} \ T$$

[ Unrestricted CSG [CFG](#) Regular ]

Languages generated by context-free grammars are known as *context-free languages* (CFL).

Not all languages can be generated by CFGs.

The *language equality* question (do two given context-free grammars generate the same language?) is undecidable.

The *language inclusion* question is also undecidable: Given two CFGs, can the first one generate all strings that the second one can generate?

The *emptiness problem* (whether the grammar generates any terminal strings at all), is undecidable for context-sensitive grammars, but decidable for CFGs.

*Leftmost derivation*: always expands leftmost non-terminal.

There are *left recursive* CFGs: when non-terminals stay always on the left side of the right-side hand of the rule. Similarly, there are *right recursive* CFGs.



## Type-3: Regular Grammar

In a *regular grammar* all production rules have at most one non-terminal symbol in the rightmost or leftmost position in the rule ( $A$  and  $B$  are non-terminals and  $a$  is a string of terminals):

$$A \rightarrow a$$

$$A \rightarrow a B \quad (\text{right-linear grammar})$$

$$A \rightarrow B a \quad (\text{left-linear grammar})$$

$$A \rightarrow \epsilon$$

[ Unrestricted CSG CFG [Regular](#) ]

Left-linear grammar is just another name for left-regular grammar (the same for right-).

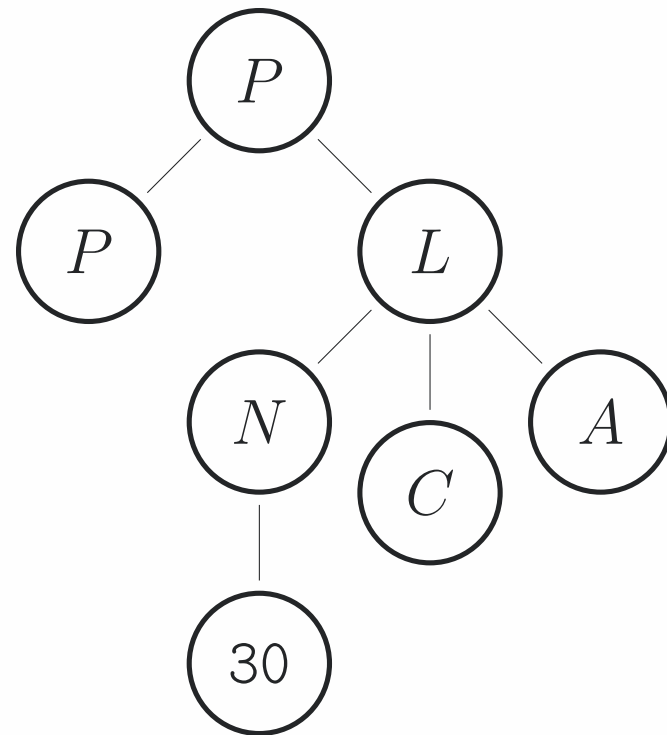
Some textbooks and articles disallow empty rules (with  $\epsilon$ ).

A regular grammar generates exactly the language a nondeterministic finite automaton accepts.

Chapter #3:

## Parse Tree

A *parse tree* (parsing tree, derivation tree, concrete syntax tree) is an ordered, rooted tree that represents the syntactic structure of a string according to some CFG.



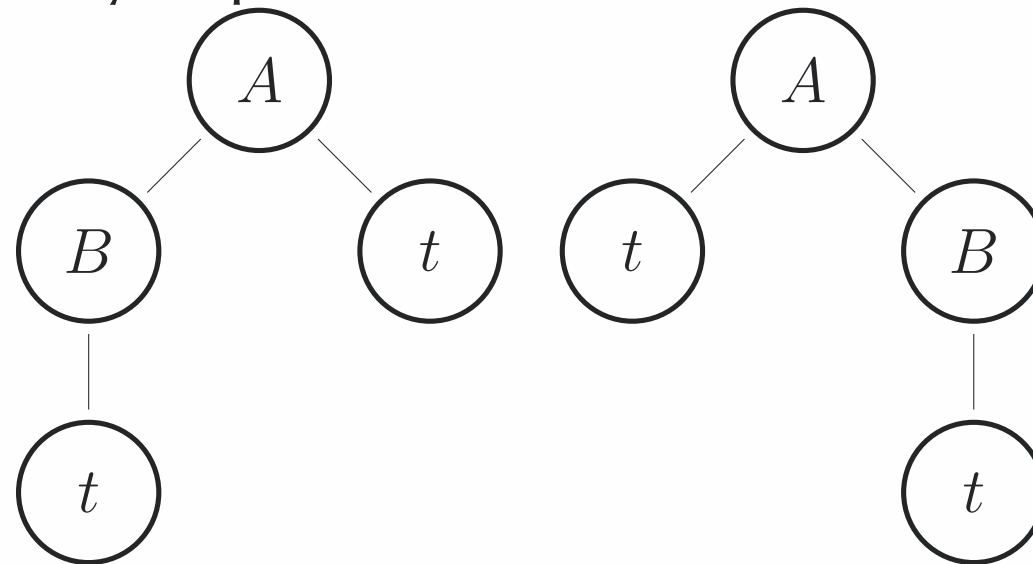
Chapter #4:

# Ambiguity

An *ambiguous grammar* is a CFG for which there exists a string that can have more than one leftmost derivation or parse tree. For example, this grammar:

$$\begin{aligned} A &\rightarrow B t \mid t B \\ B &\rightarrow t \end{aligned}$$

May be parsed as two different trees:





Chapter #5:

## Non-determinism

Non-deterministic CFG:

$$A \rightarrow B x$$

$$A \rightarrow B y$$

$$A \rightarrow B z$$

*Backtracking* in a parser is required in order to parse this grammar.

By using *left factoring* it is possible to remove non-determinism:

$$A \rightarrow B C$$

$$C \rightarrow x \mid y \mid z$$

# References