Formal Grammar

Yegor Bugayenko

Lecture #1 out of 10
80 minutes

The slidedeck was presented by the author in this YouTube Video

> 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):

```
PRINT 42
PRINT 256
PRINT 0
```

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

```
XPATH "/car/price";
SET "$2000";
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:

```
10 PRINT "What is your name?"
20 INPUT X
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:

- $N = \{P_{\texttt{rogram}}, L_{\texttt{ine}}, N_{\texttt{umber}}, C_{\texttt{ommand}}, A_{\texttt{rgument}}, \ldots\}$ (*non-terminals* or *variables*)
- $T = \{\texttt{10}, \texttt{20}, \texttt{PRINT}, \texttt{X}, \texttt{,}, \texttt{"Hello"}, \ldots\}$ (*terminals* or *alphabet*)
- $P = \{\ldots\}$ (*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 $\boldsymbol{L}(G)$: 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. $L_{\mathtt{ine}} \rightarrow N_{\mathtt{umber}}\ \mathtt{INPUT}\ A_{\mathtt{rgument}}$
2. $N_{\mathtt{umber}} \rightarrow \mathtt{10}$
3. $N_{\mathtt{umber}} \rightarrow \mathtt{20}$

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

$$(T \cup N)^* n (T \cup N)^* \rightarrow (T \cup N)^* \quad n \in N$$
$$V^* n V^* \rightarrow V^* \quad V = (T \cup N)$$

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 α is not empty (not ϵ) in each rule:

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

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

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 $\alpha \to \beta$ where the length of α is less than or equal to that of β.

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

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

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:

$$p_1\colon P_{\texttt{rogram}} \to P_{\texttt{rogram}}\ L_{\texttt{ine}}$$
$$p_2\colon P_{\texttt{rogram}} \to \epsilon$$
$$p_3\colon L_{\texttt{ine}} \to I_{\texttt{nteger}}\ C_{\texttt{ommand}}\ T_{\texttt{ail}}$$
$$p_4\colon T_{\texttt{ail}} \to T_{\texttt{ail}}\ A_{\texttt{rgument}}$$
$$p_5\colon T_{\texttt{ail}} \to \epsilon$$
$$p_6\colon I_{\texttt{nteger}} \to \texttt{10}$$
$$p_7\colon I_{\texttt{nteger}} \to \texttt{20}$$

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

$$
\begin{aligned}
P &\underset{p_1}{\Longrightarrow} \boldsymbol{P}\ \boldsymbol{L} \\
&\underset{p_3}{\Longrightarrow} P\ \boldsymbol{I}\ \boldsymbol{C}\ \boldsymbol{T} \\
&\underset{p_8}{\Longrightarrow} P\ \text{30}\ C\ T \\
&\underset{p_?}{\Longrightarrow} P\ \text{30 PRINT}\ T \\
&\underset{p_1}{\Longrightarrow} \boldsymbol{P}\ \boldsymbol{L}\ \text{30 PRINT}\ T \\
&\underset{p_?}{\Longrightarrow} \ldots
\end{aligned}
$$

We can say that "G derives in zero or more steps": $\overset{*}{\underset{G}{\Rightarrow}}$ (it is *reflexive transitive closure* of $\underset{G}{\Rightarrow}$). For example:

$$P \overset{*}{\underset{G}{\Rightarrow}} P\ L\ \texttt{30 PRINT}\ T$$

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):

$$\begin{array}{ll} A \to a & \\ A \to a\ B & \text{(\textit{right-linear grammar})} \\ A \to B\ a & \text{(\textit{left-linear grammar})} \\ A \to \epsilon & \end{array}$$

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

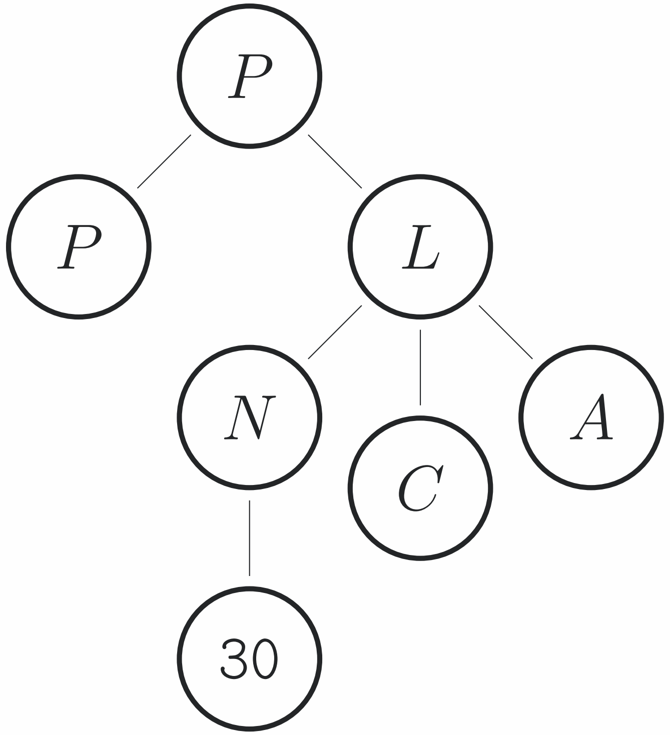
Some textbooks and articles disallow empty rules (with ϵ).

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.



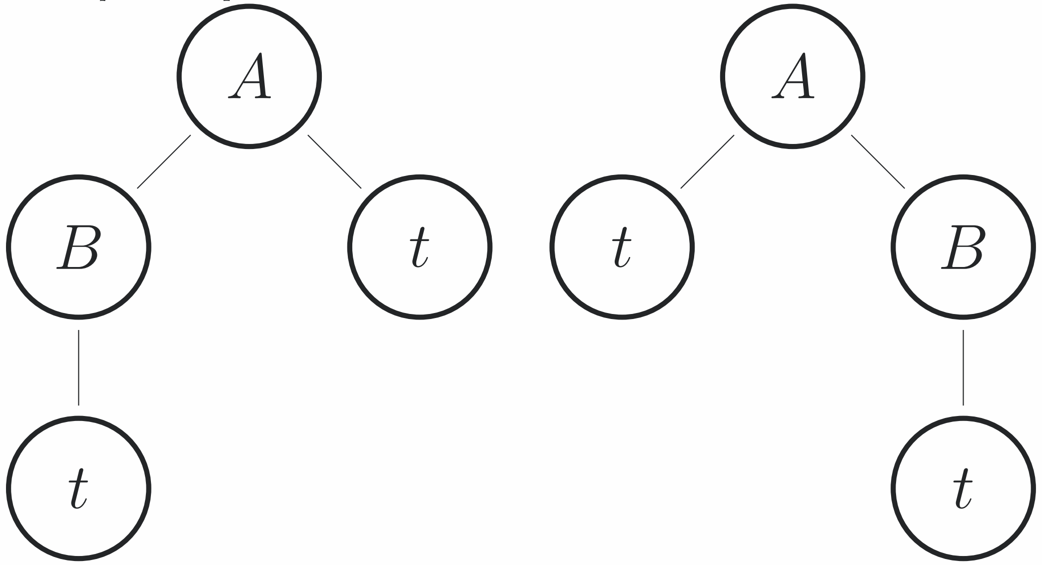
Chapter #4:

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:

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

May be parsed as two different trees:



Chapter #5:

Non-determinism

Non-deterministic CFG:

$$
\begin{aligned}
A &\to B\ x \\
A &\to B\ y \\
A &\to B\ z
\end{aligned}
$$

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

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

$$
\begin{aligned}
A &\to B\ C \\
C &\to x \mid y \mid z
\end{aligned}
$$

References