

# Grammars

## Introduction

When I took Latin, starting in 1954, I felt intuitively that the grammar of the language was somehow mathematical. But the highest math I knew was algebra, and I could not come up with any mathematical formulation of Latin grammar.

In 1957, I finally read something about the theory of grammars. Noam Chomsky was one of the pioneers of this field. Chomsky was trying to analyze natural languages, a (possibly) hopeless task. Fortunately, we do not look at natural languages in this course. Instead, we deal exclusively with formal languages and formal grammars. Henceforth, grammar means formal grammar.

Each grammar generates a language. A grammar  $G$  can be represented by a string, which we call  $\langle G \rangle$ . A grammar has a finite description, but might generate an infinite language.

There are classes of grammars, each of which generates a class of languages. Not all languages are generated by grammars, but many important languages are.

## Definition of a Grammar

A grammar  $G$  consists of the following parts.

1. Terminal alphabet  $\Sigma$ .
2. Variable alphabet  $V$ , The two alphabets may not have any symbol in common.  $\Gamma = \Sigma + V$  is called the *alphabet of grammar symbols*.
3.  $S \in V$ , the *start symbol*.<sup>1</sup>
4. Finite set of productions. Each production is of the form  $lhs \rightarrow rhs$ , where  $lhs$  and  $rhs$  are strings of grammar symbols, called the *left-hand side* and the *right-hand-side* of the production.

## Classes of Grammars

Chomsky defines five classes of grammars.

1. Left-regular grammars. This class generates the class of regular languages.
2. Right-regular grammars. This class also generates the class of regular languages.
3. Context-free grammars. This class generates the class of context-free languages.
4. Context-sensitive grammars. This class generates the class of context-sensitive languages.
5. Unrestricted grammars. This class generates the class of recursively enumerable languages.

These classes are defined by properties of their right-hand sides and left-hand sides. For each class, there must be at least one production whose  $lhs$  is the start symbol, which, by convention, is typically  $S$ .

1. For each production of a left-linear grammar, the  $lhs$  must be a single member of  $V$ . The right-hand-side must be one of the following:

- (a) A terminal

---

<sup>1</sup>There is no rule that the start symbol be called "S." It could have any name.

- (b) A terminal followed by a variable
  - (c) A variable
  - (d) The empty string
2. For each production of a right-linear grammar, the *lhs* must be a single member of  $V$ . The right-hand-side must be one of the following:
- (a) A terminal
  - (b) A variable followed by a terminal
  - (c) A variable
  - (d) The empty string

Left and right linear grammars are called *regular* grammars. Note: the (b) rules cannot be mixed. For example, a regular grammar could not have both the productions  $A \rightarrow aA$  and  $A \rightarrow Ab$ .

Our definitions of linear grammars differs from the definitions given by Definition 3.3 in our textbook. In that definition, only (a) and (b) are given. We have three justifications for this change.

(i) If we do not have (d), we cannot generate the empty string. But the empty string is a member of many regular languages.

(ii) If we do not have (c), we have to go through unnecessary contortions to define the grammar generates the language accepted by an NFA which has a  $\lambda$ -transition.

(iv) With (a), (b), (c), and (d), it is easier to understand the construction or regular grammars equivalent to finite automata.

(iii) Allowing (c) and (d) does no harm: we still get only regular languages.

3. For each production of a context-free grammar, the *lhs* must be a single member of  $\Gamma$ . The right-hand-side can be any string of grammar symbols.

4. For each production of a context-sensitive grammar, the *lhs* and *rhs* must be non-empty strings of grammar symbols, and the length of the *rhs* must be at least as great the the length of the *lhs*<sup>2</sup>

For each production of an unrestricted grammar, the *lhs* must be a non-empty string of grammar symbols, while the *rhs* may be any string of grammar symbols.

## Derivations

Let  $G$  be a grammar. A  $G$ -derivation of a string  $w \in \Sigma^*$  is a sequence of strings over  $\Gamma$  connected by the symbol " $\Rightarrow$ ," which is read as the word "derives." These strings are called *sentential forms*. In any derivation of  $w$ , the first sentential form is simply the start symbol, and the last is  $w$  itself.

Each step of a derivation makes use of just one production. The *lhs* of that production is replaced by the *rhs* of that same production. That is, if  $u$  and  $v$  are consecutive sentential forms, *i.e.*,  $u \Rightarrow v$ , there must be a production  $\alpha \rightarrow \beta$  such that  $\alpha$  is replaced by  $\beta$  at that step. That is, there are strings  $x, y \in \Gamma^*$  such that

1.  $u = x\alpha y$

---

<sup>2</sup>This rule does not permit a context-sensitive language to contain the empty string. However, we usually want to allow the empty string. We can achieve that by permitting the production  $S \rightarrow \lambda$ , as long as  $S$  is not on the *rhs* of any production.

2.  $v = x\beta y$

If productions are labeled, we sometimes place the label of the production above the “derives” symbol  $\Rightarrow$  for clarity.

The language generated by  $G$ , called  $L(G)$ , is defined to be the language of all  $w \in \Sigma^*$  which can be *derived* from the start symbol.

**Example.** Let  $L = \{a^n b^m : n, m \geq 0\}$ , which is described by the regular expression  $a^* b^*$ . Then  $L$  is generated by the regular grammar:

1.  $S \rightarrow aS$
2.  $S \rightarrow B$
3.  $B \rightarrow bB$
4.  $B \rightarrow \lambda$

We now give a  $G$ -derivation of  $w = aabbb$ .

$$S \xrightarrow{1} aS \xrightarrow{1} aaS \xrightarrow{2} aaB \xrightarrow{3} aabB \xrightarrow{3} aabbB \xrightarrow{3} aabbbB \xrightarrow{4} aabbb$$

The above derivation proves that  $w$  is generated by  $G$ , that is,  $w \in L(G)$ .

## Computing a Regular Grammar from a DFA

Given any DFA  $M$ , there is a straightforward way to find a regular grammar which generates the language accepted by  $M$ . Suppose  $\Sigma = \{a_1, \dots, a_n\}$  and  $Q = \{q_0, q_1, \dots, q_m\}$ .<sup>3</sup>

We use  $\delta$  to define a left-linear grammar  $G$ . We let  $\Sigma$  be the terminal alphabet of  $G$ . We let  $V = \{A_0, A_1, \dots, A_m\}$  be the alphabet of variables. We let  $A_0$  be the start state of  $G$ . Each arrow in the state diagram defines a production of the grammar, and each final state also defines a production.

If  $\delta(q_i, a_j) = q_k$ ,  $A_i \rightarrow a_j A_k$  is a production.

If  $q_i$  is a final state,  $A_i \rightarrow \lambda$  is a production.

Let  $M$  be the DFA illustrated in Figure 1.

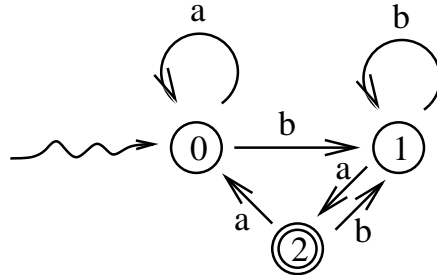


Figure 1

<sup>3</sup>Again, to avoid clutter, we write merely “ $i$ ” to denote “ $q_i$ ” in state diagrams.

$M$  has one final state and its state diagram has SIX arrows, thus  $G$  has seven productions:

1.  $A_0 \rightarrow aA_0$
2.  $A_0 \rightarrow bA_1$
3.  $A_1 \rightarrow bA_1$
4.  $A_1 \rightarrow aA_2$
5.  $A_2 \rightarrow aA_0$
6.  $A_2 \rightarrow bA_1$
7.  $A_2 \rightarrow \lambda$

Here is a  $G$ -derivation of  $abbaba$ :

$$A_0 \xrightarrow{1} aA_0 \xrightarrow{2} abA_1 \xrightarrow{3} abbA_1 \xrightarrow{4} abbaA_2 \xrightarrow{6} abbabA_1 \xrightarrow{4} abbabaA_2 \xrightarrow{7} abbaba$$

## Computing a Regular Grammar from an NFA

Similarly, given an NFA  $M$  which accepts a language  $L$ , there is we can find a left-linear grammar which generates  $L$ . Again, let  $\Sigma = \{a_1, \dots, a_n\}$  and  $Q = \{q_0, q_1, \dots, q_m\}$ ,

As in the case of a DFA, we let  $\Sigma$  be the terminal alphabet of  $G$ , and  $V = \{A_0, A_1, \dots, A_m\}$  the alphabet of variables, and  $A_0$  the start state of  $G$ . As in the case of a DFA, each final state and each arrow in the state diagram defines a production.

If  $q_k \in \delta(q_i, a_j)$ ,  $A_i \rightarrow a_j A_k$  is a production.

If  $q_k \in \delta(\lambda, a_j)$ ,  $A_i \rightarrow A_k$  is a production.

If  $q_i$  is a final state,  $A_i \rightarrow \lambda$  is a production.

Let  $M$  be the NFA illustrated in Figure 2.

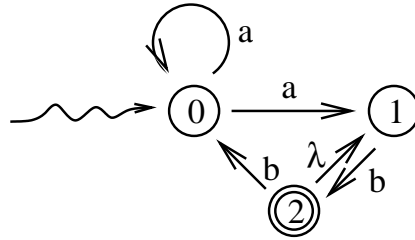


Figure 2

$M$  has one final state and its state diagram has five arrows, thus  $G$  has six productions:

1.  $A_0 \rightarrow aA_0$
2.  $A_0 \rightarrow aA_1$
3.  $A_1 \rightarrow bA_2$
4.  $A_2 \rightarrow bA_0$
5.  $A_2 \rightarrow A_1$
6.  $A_2 \rightarrow \lambda$

Here is a  $G$ -derivation of  $aabbab$ :

$$A_0 \xrightarrow{1} aA_0 \xrightarrow{2} aaA_1 \xrightarrow{3} aabA_2 \xrightarrow{5} aabA_1 \xrightarrow{3} aabbA_2 \xrightarrow{4} aabbbA_0 \xrightarrow{2} aabbbA_1 \xrightarrow{3} aabbbabA_2 \xrightarrow{6} aabbbab$$

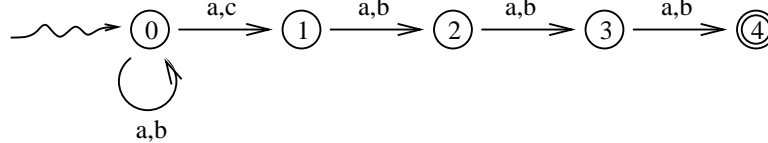
The above derivation proves that  $aabbbab$  is generated by  $G$ , that is,  $aabbbab \in L(G)$ .

By this construction, we have the following theorem.

**Theorem 1** *If a language  $L$  is accepted by an NFA with  $n$  states, then  $L$  is generated by a left-linear grammar with  $n$  variables.*

*Proof:* We just use the above construction. The formal proof that it yields a left-linear grammar for the same language is more detailed, but not too hard to understand. ■

**Exercise 1** *Give a regular grammar for the language accepted by  $M$ .*



**Figure 3:** NFA  $M$  for Exercise 1

Ans:  $M$  has five states, and the minimal DFA which accepts  $L(M)$  has  $2^5 = 32$  states. The following regular grammar generates  $L(M)$ , where  $A_0$  is the start state.

1.  $A_0 \rightarrow aA_0$
2.  $A_0 \rightarrow bA_0$
3.  $A_0 \rightarrow aA_1$
4.  $A_0 \rightarrow bA_1$
5.  $A_1 \rightarrow aA_2$
6.  $A_1 \rightarrow bA_2$
7.  $A_2 \rightarrow aA_3$
8.  $A_2 \rightarrow bA_3$
9.  $A_3 \rightarrow aA_4$
10.  $A_3 \rightarrow bA_4$
11.  $A_4 \rightarrow \lambda$

## Context-Free Grammars

A grammar  $G$  is *context-free* if, for every production, the left-hand side is one variable. We write CFG to mean context-free grammar. The right hand side of production of a CFG can be any string of grammar symbols. The class of context-free grammars is, arguably, the most important class of grammars we study.

A language  $L$  is called *context-free* if it is generated by some context-free grammar. Two grammars are said to be *equivalent* if they generate the same language. Every context-free language is generated by infinitely many different equivalent context-free grammars. We write CFL to mean context-free language.

**Remark 1** *Every regular language is a context-free language.*

*Proof:* Every regular grammar is a context-free grammar. ■

## Simple Examples

**Simplest Example.** The grammar  $G$ , where  $V = \{S\}$  and  $\Sigma = \{a, b\}$ , the start symbol is  $S$ , and the productions are:

1.  $S \rightarrow aSb$
2.  $S \rightarrow \lambda$

$L(G) = \{a^n b^n : n \geq 0\}$ , arguably the simplest non-regular context-free language.

Here is a  $G$ -*derivation* (or just *derivation* if  $G$  is understood) of  $w = aabb$ .

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

**Dyck Language.** The Dyck language is the language of all balanced strings of left and right parentheses, which is over the alphabet  $\Sigma = \{(, )\}$ . Here are three grammars for the Dyck language. In each case,  $S$  is the start symbol and  $\Gamma = \{S\}$ .

$G_1$

1.  $S \rightarrow (S)$
2.  $S \rightarrow SS$
3.  $S \rightarrow \lambda$

$G_2$

1.  $S \rightarrow S(S)$
2.  $S \rightarrow \lambda$

$G_3$

1.  $S \rightarrow (S)S$
2.  $S \rightarrow \lambda$

**Palindromes.** A *palindrome* is a word which is its own reverse, such as “level” or “noon.” Let  $L$  be the language of all palindromes over the alphabet  $a, b$ .  $L$  is generated by the CFG

1.  $S \rightarrow aSa$
2.  $S \rightarrow bSb$
3.  $S \rightarrow a$
4.  $S \rightarrow b$
5.  $S \rightarrow \lambda$

## Derivations and Parse Trees

If  $G$  is a CFG and  $L = L(G)$ , then each  $w \in L$  has at least one derivation, frequently multiple derivations. For example, let  $L = \{a^n c^n b^m d^m : n, m \geq 0\}$ . Then  $L$  is generated by a grammar with variables  $S, A, B$  and start symbol  $S$ :

1.  $S \rightarrow AB$

2.  $A \rightarrow aAc$
3.  $A \rightarrow \lambda$
4.  $B \rightarrow bBd$
5.  $B \rightarrow \lambda$

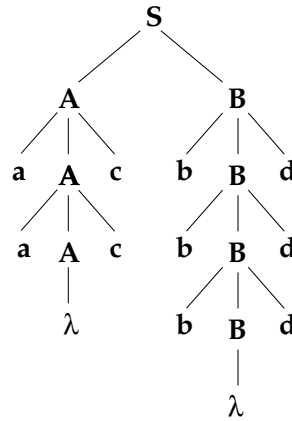
Let  $w = aaccbbddd$ . Here are two derivations of  $w$ .

$$S \Rightarrow AB \Rightarrow aAcB \Rightarrow aaAccB \Rightarrow aaccB \Rightarrow aaccbBd \Rightarrow aaccbbBdd \Rightarrow aaccbbbBddd \Rightarrow aaccbbddd$$

$$S \Rightarrow AB \Rightarrow AbBd \Rightarrow AbbBcc \Rightarrow AbbbBccc \Rightarrow Abbbddd \Rightarrow aAcbbddd \Rightarrow aaAccbbddd \Rightarrow aaccbbddd$$

The first of these is a *left-most* derivation, since at each step of the derivation, the left-most variable of the sentential form is replaced by the *rhs* of a production. For example, in the second step,  $A$  is replaced by  $aAc$  and  $B$  is not replaced. Similarly, the second derivation is a *right-most* derivation.

**Parse Trees.** For each derivation of a string  $w \in L(G)$ , there is a parse tree of  $w$ . The internal nodes of this tree are the variables in the derivation and each leaf is either a terminal or  $\lambda$ . The two derivations of  $aaccbbddd$  shown above give rise to the same parse tree, shown to the right:



## Ambiguous and Unambiguous Grammars

A CFG  $G$  is called *unambiguous* if every string  $w \in L(G)$  has exactly one left-most derivation.

### Theorem 2

- (a) A CFG  $G$  is unambiguous if and only if every string  $w \in L(G)$  has exactly one right-most derivation.
- (b) A CFG  $G$  is unambiguous if and only if every string  $w \in L(G)$  has exactly one parse tree.

A CFG is *ambiguous* if it is not unambiguous. A CFL is called *inherently ambiguous* if it has no unambiguous CFG.

The grammar  $G_1$  for the Dyck language is ambiguous, while  $G_2$  and  $G_3$  are both unambiguous.

## Dangling Else

Let  $G$  be the following context-free grammar, with start symbol  $S$  and terminals  $\{a, i, e\}$

1.  $S \rightarrow a$
2.  $S \rightarrow iS$

3.  $S \rightarrow iSeS$

**Exercise 2** Show that  $G$  is ambiguous by giving two parse trees for the string  $iaea$ .

$L(G)$  does have an unambiguous grammar, but it is more complex than the ambiguous grammar.

$G$  models the “dangling else” problem for programming languages. In the following C++ fragment, what value will be output?

```
int x = 0;
int y = 0;
int z = 3;
if(x == 1)
if(y == 0)
z = 2;
else z = 4;
cout << z << endl;
```

What value will be output?