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. By that time Noam Chomsky had given a mathematical definition of a *formal grammar*. 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 described 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 grammers, each of which generates a class of lenguages. 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 Σ.

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*.[1]

4. Finite set of productions. Each production is of the form *lhs*→*rhs*, where *lhs* and *rhs* are strings of grammar symbols, called the *left-hand side* and the *right-hand-side* of the production.
The left-hand side cannot be the empty string, and must be S for at least one 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.

Those two classes are, together, called regular grammars. 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:

[1] There is no rule that the start symbol be called "S." It could have any name.

(a) A terminal
(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 grammer could not have both the productions $A \to aA$ and $A \to Ab$.

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 a grammar which generates the language accepted by an NFA which has a λ-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 Γ. 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*[2]

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 Γ 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 $\alpha \to \beta$ such that α is replaced by β at that step. That is, there are strings $x, y \in \Gamma^*$ such that

[2]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 $S \to \lambda$, as long as S is not on the *rhs* of any production.

1. $u = x\alpha y$
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. $S \to aS$
2. $S \to B$
3. $B \to bB$
4. $B \to \lambda$

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

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

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 $\Sigma = \{a_1, \ldots a_n\}$ and $Q = \{q_0, q_1, \ldots q_m\}$.[3]

We use δ to define a left-linear grammar G. We let Σ be the terminal alphabet of G. We let $V = \{A_0, A_1, \ldots A_m\}$ 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 $\delta(q_i, a_j) = q_k$, $A_i \to a_j A_k$ is a production.

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

Let M be the DFA illustrated in Figure 1.

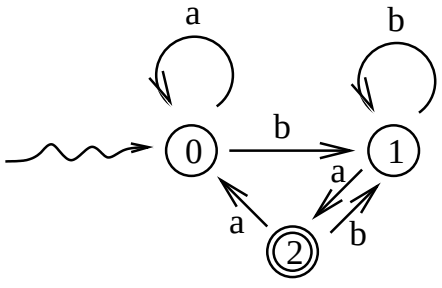


Figure 1

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

[3]Again, to avoid clutter, we write merely "i" to denote "q_i" in state diagrams.

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

Here is a G-derivation of $abbaba$:

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

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 $\Sigma = \{a_1, \ldots a_n\}$ and $Q = \{q_0, q_1, \ldots q_m\}$,

As in the case of a DFA, we let Σ be the terminal alphabet of G, and $V = \{A_0, A_1, \ldots A_m\}$ 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 $q_k \in \delta(q_i, a_j)$, $A_i \to a_j A_k$ is a production.
If $q_k \in \delta(q_i, \lambda)$, $A_i \to A_k$ is a production.
If q_i is a final state, $A_i \to \lambda$ is a production.

Let M be the NFA illustrated in Figure 2.

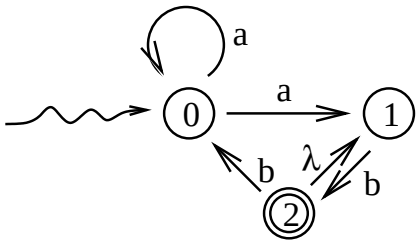


Figure 2

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

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

Here is a G-derivation of $aabbbab$:

$$A_0 \overset{1}{\Rightarrow} aA_0 \overset{2}{\Rightarrow} aaA_1 \overset{3}{\Rightarrow} aabA_2 \overset{5}{\Rightarrow} aabA_1 \overset{3}{\Rightarrow} aabbA_2 \overset{4}{\Rightarrow} aabbbA_0 \overset{2}{\Rightarrow} aabbbaA_1 \overset{3}{\Rightarrow} aabbbabA_2 \overset{6}{\Rightarrow} aabbbab$$

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

By this constructiohn, 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.*

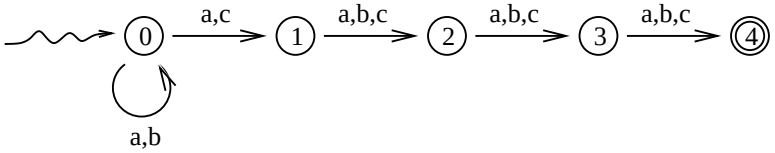


Figure 3: NFA M for Exercise 1

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. $A_0 \to aA_0$
2. $A_0 \to bA_0$
3. $A_0 \to aA_1$
4. $A_0 \to cA_1$
5. $A_1 \to aA_2$
6. $A_1 \to bA_2$
6. $A_1 \to cA_2$
7. $A_2 \to aA_3$
8. $A_2 \to bA_3$
8. $A_2 \to cA_3$
9. $A_3 \to aA_4$
10. $A_3 \to bA_4$
10. $A_3 \to cA_4$
11. $A_4 \to \lambda$

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. We write CFL to mean context-free language. 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.

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. $S \to aSb$
2. $S \to \lambda$

$L(G) = \{a^n b^n : n \geq 0\}$, 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. $S \to (S)$
2. $S \to SS$
3. $S \to \lambda$

G_2
1. $S \to S(S)$
2. $S \to \lambda$

G_3
1. $S \to (S)S$
2. $S \to \lambda$

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

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. $S \to AB$
2. $A \to aAc$
3. $A \to \lambda$
4. $B \to bBd$
5. $B \to \lambda$

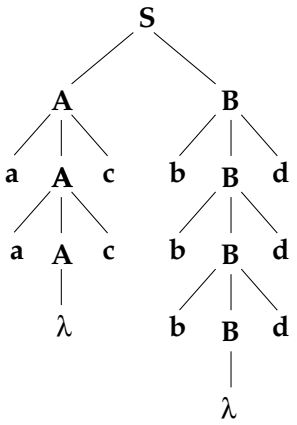
Let $w = aaccbbbddd$. Here are two derivations of w.

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

$$S \Rightarrow AB \Rightarrow AbBd \Rightarrow AbbBcc \Rightarrow AbbbBccc \Rightarrow Abbbddd \Rightarrow aAcbbbddd \Rightarrow aaAccbbbddd \Rightarrow aaccbbbddd$$

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 λ. The two derivations of *aaccbbbddd* 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* unambigous *if and only if every string $w \in L(G)$ has exactly one right-most derivation.*

(b) *A CFG G is* unambigous *if and only if every string $w \in G(L)$ 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 ambigous, 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. $S \to a$
2. $S \to iS$
3. $S \to iSeS$

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

$L(G)$ does have an unambigous 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;
```

Ans: C++ is not actually a CFL, but it has an "underlying" CF grammar which is ambiguous, like the grammar above. The "dangling else" ambiguity is resolved during the parsing phase of the compiler. Each "else" is paired with the nearest "if" if there is a choice. For example the "else" on line 7 is paired with the "if" on line 5. Thus the output is 3.

Simple CF Exercises

1. Find a CFG for $L = \{a^m b^n : 0 \leq n \leq m\}$
   Ans:
   1. $S \to aS$
   2. $S \to B$
   3. $B \to aBb$
   4. $B \to \lambda$

   This grammar is unambiguous.

2. Find a simple CFG for the language $L = \{a^n b^m c^n : n, m \geq 0\}$
   Ans: There are many correct answers. Here is (perhaps) the simplest one:
   1. $S \to aSc$
   2. $S \to B$
   3. $B \to bB$
   4. $B \to \lambda$

   This grammar is unambiguous.

3. Find a CFG for $L = \{a^i b^j c^k : i, j, k \geq 0 \text{ and either } i = j \text{ or } j = k\}$

   Ans:

   1. $S \to S_1$
   2. $S \to S_2$
   3. $S_1 \to S_1 c$
   4. $S_1 \to A$
   5. $A \to aAb$
   6. $A \to \lambda$
   7. $S_2 \to aS_2$
   8. $S_2 \to C$
   9. $C \to bCc$
   10. $C \to \lambda$

   This grammar is ambiguous. Can you prove it?

   Ans:

   There are two parse trees for abc.

   Actually, L is inherently ambiguous.

4. (a) Find a CFG for $L = \{a^n b^n c^n : n \geq 0\}$

   Ans: There is no answer. L is not context-free.

   (b) Let L' be the complement of L. That is, L', $L' = \{w \in \Sigma^* : w \notin L\}$, where $\Sigma = \{a, b, c\}$. Prove that L' is a CFL.

Context-Sensitive Grammars

There are two definitions of context-sensitive grammars. The standard definition found on Wikipedia is hard to work with, but our textbook (Linz) gives an algernative definition which is simpler, which I give below.

Definition: A grammar is *context-sensitive* if the left side of every production is exactly the start symbol and the right side of production is at least as long (number of symbols) as the left side. However a CSG cannot generate the empty string, thus, we define a language L to be context-sensitive if there is a context sensitive grammar G such that either $L = L(G)$ or $L = L(G) + \{\lambda\}$.

Example. Let $L = \{a^n b^n c^n : n \geq 0\}$. The empty string is a member of L, but if we delete the empty string, we obtain the language $L' = \{a^n b^n c^n : n \geq 1\}$, which is generated by the CSG G below:

1. $S \to abc$
2. $aAbc$
3. $Ab \to bA$
4. $Ac \to Bbcc$
5. $bB \to Bb$
6. $aB \to aa$
7. aaA

Here is the derivation of $aaabbbccc$ using that G:
$S \Rightarrow aAbc \Rightarrow abAc \Rightarrow abBbcc \Rightarrow aBbbcc \Rightarrow aaAbbcc \Rightarrow aabAbcc \Rightarrow aabbAcc \Rightarrow aabbBbccc \Rightarrow aabBbbccc \Rightarrow aaBbbbccc \Rightarrow aaabbbccc$

Exercise: For each step of the above derivation, identify which production is used.

Exercise: Find a CSG for the language $\{a^n : n \text{ is a power of } 2\}$

General Grammars

Every grammar is a general grammar.

The Chomsky Hierarchy

The Chomsky hierarchy names three types of languages:

- Type 0: Languages generated by any grammar. These languages are *recursively enumerable* languages, including some undecidable languages. These are precisely the languages which are accepted by Turing machines.
- Type 1: Context-sensitive languages. CSLs are the languages which are accepted by *linear bounded automata*.[4]
- Type 2: Context-free languages. CFLs are the languages which are accepted by *Push-down automata* (PDAs).
- Type 3: Regular languages. Regular languages are those generated by regular grammars. These languages are accepted by finite automata, either DFAs or NFAs. (But remember, a DFA is also an NFA.)
- The types are nested. Every type 3 language, is also type 2, while every type 2 is also type 1, and every type 1 is type 0.

Exercise: Find a recursively enumerable language which is not context-sensitive.

[4] I will not expect you to learn what a linear bounded automaton is in this course.