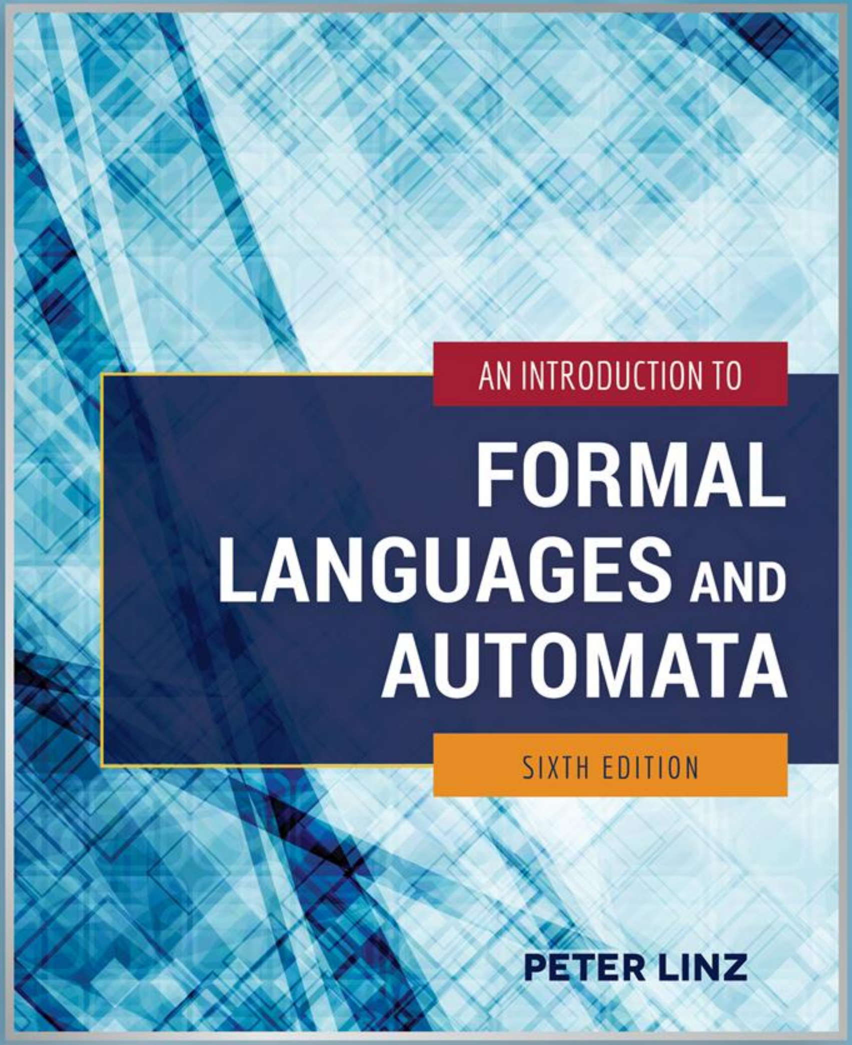
Chapter 3

REGULAR LANGUAGES AND REGULAR GRAMMARS



Learning Objectives

At the conclusion of the chapter, the student will be able to:

- Identify the language associated with a regular expression
- Find a regular expression to describe a given language
- Construct a nondeterministic finite automaton to accept the language denoted by a regular expression
- Use generalized transition graphs to construct a regular expression that denotes the language accepted by a given finite automaton
- Identify whether a particular grammar is regular
- Construct regular grammars for simple languages
- Construct a nfa that accepts the language generated by a regular grammar
- Construct a regular grammar that generates the language accepted by a finite automaton

Regular Expressions

- Regular Expressions provide a concise way to describe some languages
- Regular Expressions are defined recursively. For any alphabet:
  - the empty set, the empty string, or any symbol from the alphabet are *primitive regular expressions*
  - the union (+), concatenation (·), and star closure (*) of regular expressions is also a regular expression
  - any string resulting from a finite number of these operations on primitive regular expressions is also a regular expression

Languages Associated with Regular Expressions

- A regular expression r denotes a language L(r)
- Assuming that r_1 and r_2 are regular expressions:
  1. The regular expression $\varnothing$ denotes the empty set
  2. The regular expression λ denotes the set $\{ \lambda \}$
  3. For any a in the alphabet, the regular expression a denotes the set { a }
  4. The regular expression $r_1 + r_2$ denotes $L(r_1) \cup L(r_2)$
  5. The regular expression $r_1 \cdot r_2$ denotes $L(r_1)\, L(r_2)$
  6. The regular expression (r_1) denotes $L(r_1)$
  7. The regular expression $r_1{}^*$ denotes $(L(r_1))^*$

Determining the Language Denoted by a Regular Expression

- By combining regular expressions using the given rules, arbitrarily complex expressions can be constructed
- The concatenation symbol (·) is usually omitted
- In applying operations, we observe the following precedence rules:
  - star closure precedes concatenation
  - concatenation precedes union
- Parentheses are used to override the normal precedence of operators

Sample Regular Expressions and Associated Languages

Regular Expression	Language
(ab)*	$\{ (ab)^n, n \geq 0 \}$
a + b	{ a, b }
(a + b)*	{ a, b }* (in other words, any string formed with a and b)
a(bb)*	{ a, abb, abbbb, abbbbbb, … }
a*(a + b)	{ a, aa, aaa, …, b, ab, aab, … } (Example 3.2)
(aa)*(bb)*b	{ b, aab, aaaab, …, bbb, aabbb, … } (Example 3.4)
(0 + 1)*00(0 + 1)*	Binary strings containing at least one pair of consecutive zeros

Two regular expressions are equivalent if they denote the same language. Consider, for example, (a + b)* and (a*b*)*

Regular Expressions and Regular Languages

- Theorem 3.1: For any regular expression r, there is a nondeterministic finite automaton that accepts the language denoted by r
- Since nondeterministic and deterministic accepters are equivalent, regular expressions are associated precisely with regular languages
- A constructive proof of theorem 3.1 provides a systematic procedure for constructing a nfa that accepts the language denoted by any regular expression

Construction of a nondeterministic fa to accept a language L(r)

We can construct simple automata that accept the languages associated with the empty set, the empty string, and any individual symbol.

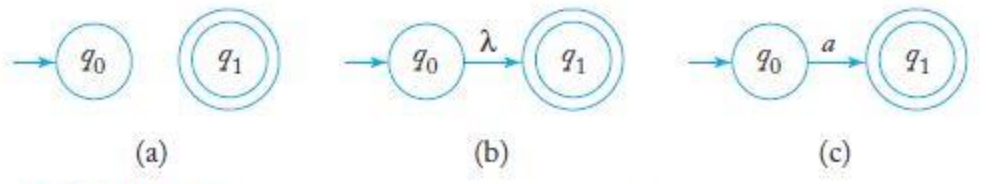


FIGURE 3.1 (a) nfa accepts $\varnothing$. (b) nfa accepts $\{\lambda\}$. (c) nfa accepts $\{a\}$.

Construction of a nondeterministic fa to accept a language L(r) (cont.)

Given schematic representations for automata designed to accept L(r_1) and (r_2), an automaton to accept L(r_1 + r_2) can be constructed as shown in Figure 3.3

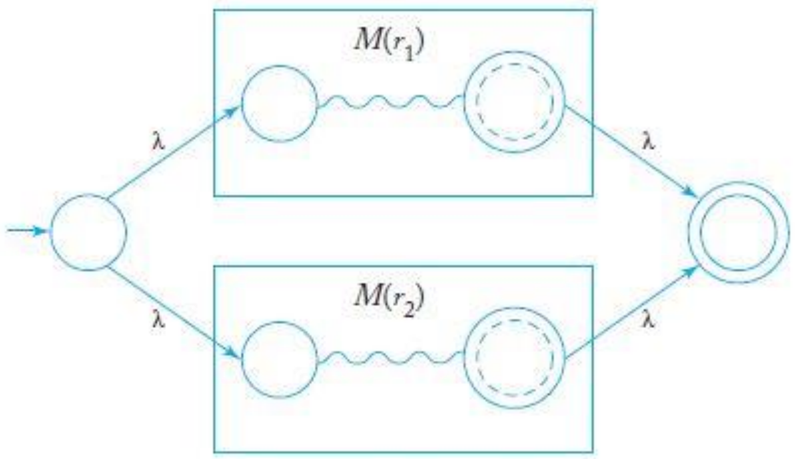


FIGURE 3.3 Automaton for $L\,(r_1 + r_2)$.

Construction of a nondeterministic fa to accept a language L(r) (cont.)

Given schematic representations for automata designed to accept L(r_1) and (r_2), an automaton to accept L(r_1r_2) can be constructed as shown in Figure 3.4

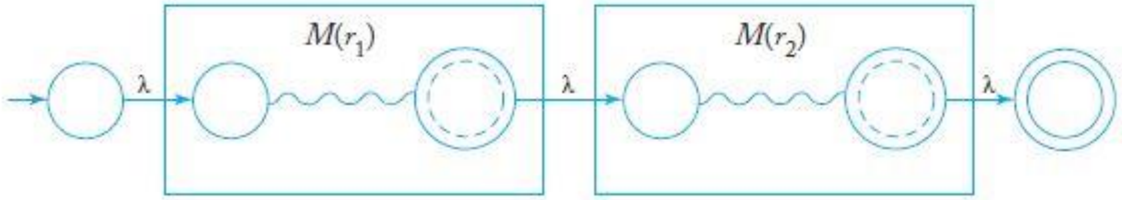


FIGURE 3.4 Automaton for $L(r_1r_2)$.

Construction of a nondeterministic fa to accept a language L(r) (cont.)

Given a schematic representation for an automaton designed to accept $L(r_1)$, an automaton to accept $L(r_1{}^*)$ can be constructed as shown in Figure 3.5

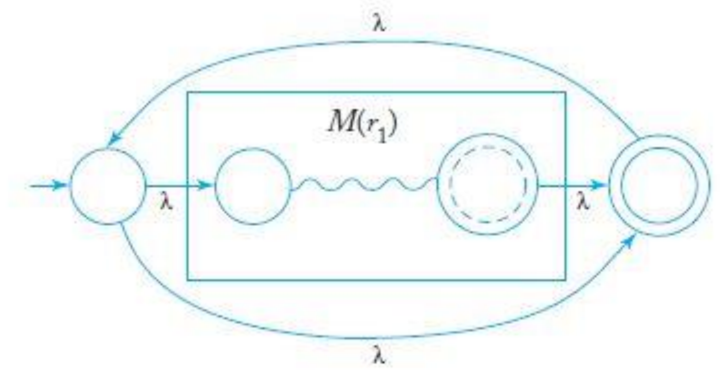


FIGURE 3.5 Automaton for $L(r_1^*)$.

Example: Construction of a nfa to accept a language L(r)

Given the regular expression $r = (a + bb)^* (ba^* + \lambda)$, a nondeterministic fa to accept L(r) can be constructed systematically as shown in Figure 3.7

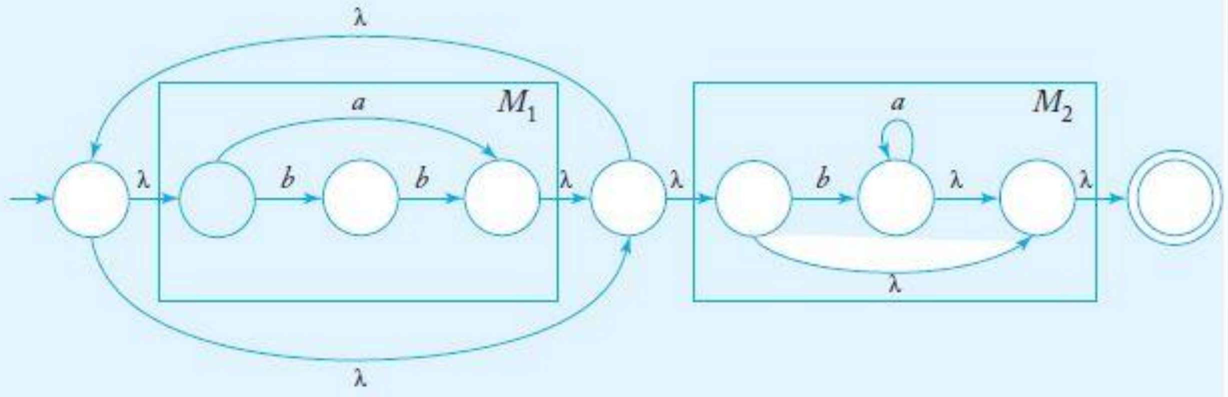


FIGURE 3.7 Automaton accepts $L((a + bb)^* \ (ba^* + \lambda))$.

Regular Expressions for Regular Languages

- Theorem 3.2: For every regular language, it is possible to construct a corresponding r.e.
- The process can be illustrated with a *generalized transition graph (GTG)*
- A GTG for L(a* + a*(a + b)c*) is shown below

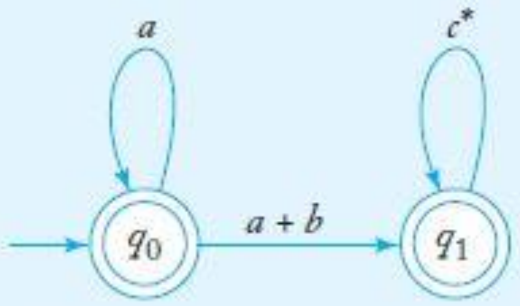


FIGURE 3.8

Regular Grammars

- In a right-linear grammar, at most one variable symbol appears on the right side of any production. If it occurs, it is the rightmost symbol.
- In a left-linear grammar, at most one variable symbol appears on the right side of any production. If it occurs, it is the leftmost symbol.
- A *regular grammar* is either right-linear or left-linear.
- Example 3.13 presents a regular (right-linear) grammar:

  V = { S }, T = { a, b }, and productions $S \rightarrow abS \mid a$

Right-Linear Grammars Generate Regular Languages

Per theorem 3.3, it is always possible to construct a nfa to accept the language generated by a regular grammar G:

- Label the nfa start state with S and a final state V_f
- For every variable symbol V_i in G, create a nfa state and label it V_i
- For each production of the form $A \rightarrow aB$, label a transition from state A to B with symbol a
- For each production of the form $A \rightarrow a$, label a transition from state A to V_f with symbol a (may have to add intermediate states for productions with more than one terminal on RHS)

Example: Construction of a nfa to accept a language L(G)

Given the regular grammar G with productions

$V_0 \to aV_1$

$V_1 \to abV_0 \mid b$

a nondeterministic fa to accept L(G) can be constructed systematically as shown in Figure 3.17

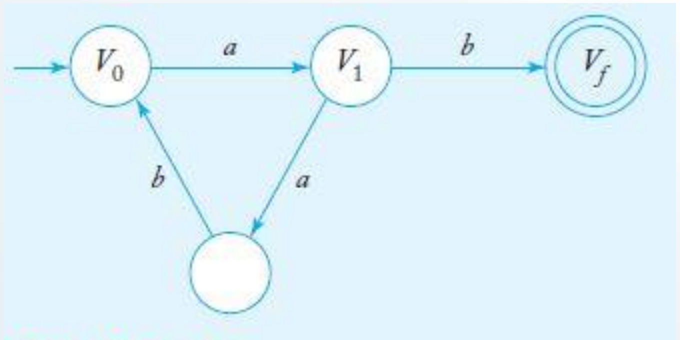


FIGURE 3.17

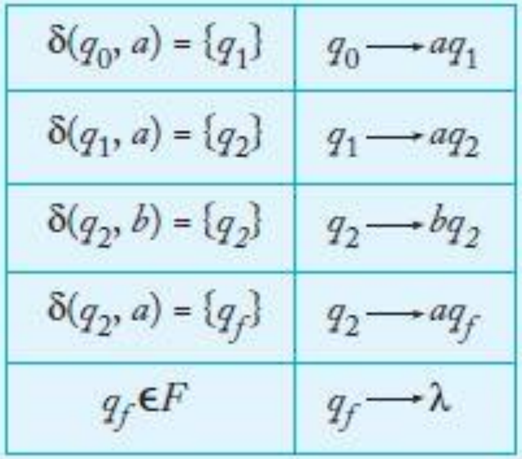
Right-Linear Grammars for Regular Languages

Per theorem 3.4, it is always possible to construct a regular grammar G to generate the language accepted by a dfa M:

- Each state in the dfa corresponds to a variable symbol in G
- For each dfa transition from state A to state B labeled with symbol a, there is a production of the form $A \rightarrow aB$ in G
- For each final state F_i in the dfa, there is a corresponding production $F_i \rightarrow \lambda$ in G

Example: Construction of a regular grammar G to generate a language L(M)

Given the language L(aab*a), Figure 3.18 shows the transition function for a dfa that accepts the language and the productions for the corresponding regular grammar.



$\delta(q_0, a) = \{q_1\}$	$q_0 \longrightarrow aq_1$
$\delta(q_1, a) = \{q_2\}$	$q_1 \longrightarrow aq_2$
$\delta(q_2, b) = \{q_2\}$	$q_2 \longrightarrow bq_2$
$\delta(q_2, a) = \{q_f\}$	$q_2 \longrightarrow aq_f$
$q_f \in F$	$q_f \longrightarrow \lambda$

FIGURE 3.18

Equivalence of Regular Languages and Regular Grammars

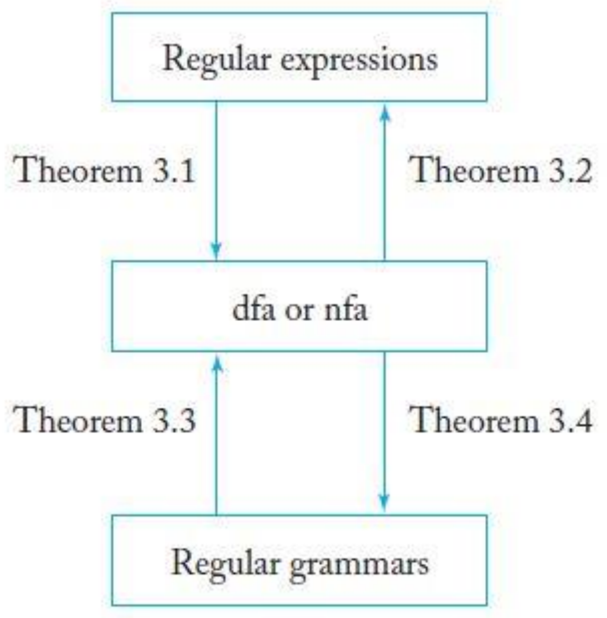


FIGURE 3.19