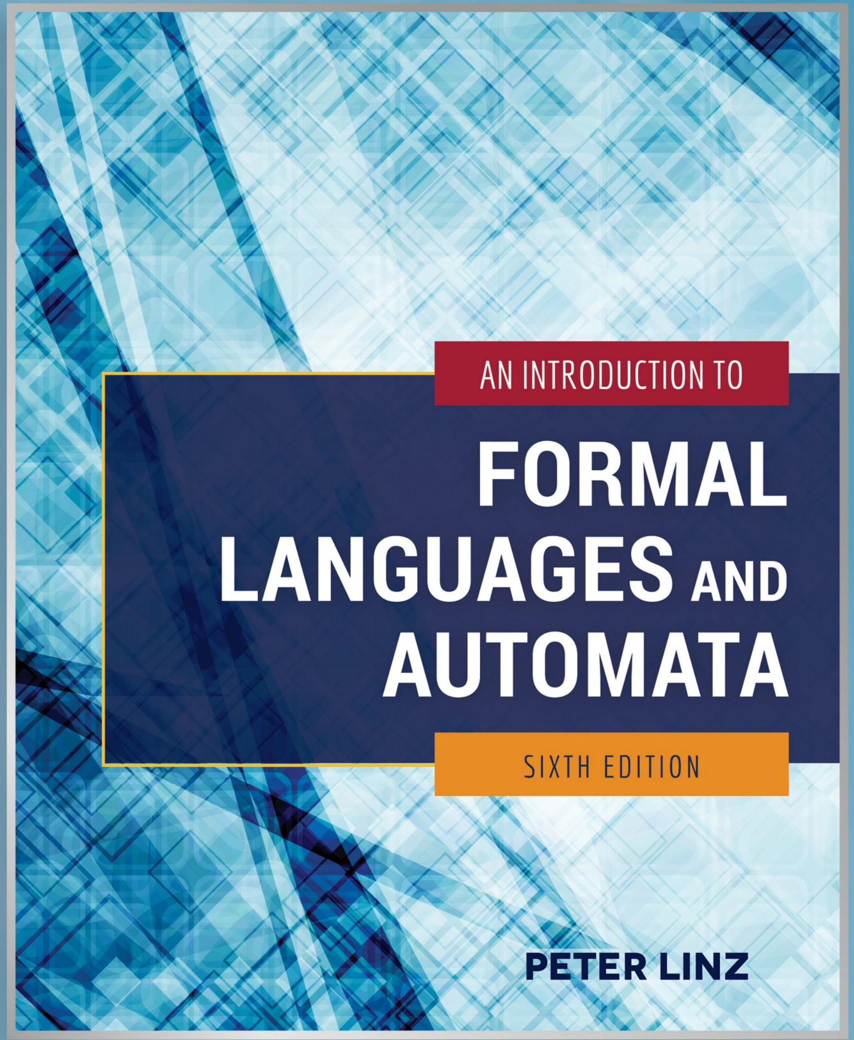
Chapter 2

FINITE AUTOMATA



Learning Objectives

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

- Describe the components of a deterministic finite accepter (dfa)
- State whether an input string is accepted by a dfa
- Describe the language accepted by a dfa
- Construct a dfa to accept a specific language
- Show that a particular language is regular
- Describe the differences between deterministic and nondeterministic finite automata (nfa)
- State whether an input string is accepted by a nfa
- Construct a nfa to accept a specific language
- Transform an arbitrary nfa to an equivalent dfa

Deterministic Finite Accepters

- Def 2.1: A *deterministic finite accepter* is defined by

  Q: a finite set of *internal states*

  Σ: a set of symbols called the *input alphabet*

  δ: a *transition function* from Q X Σ to Q

  q_0: the *initial state*

  F: a subset of Q representing the *final states*

- Example 2.1: Consider the dfa

  $Q = \{ q_0, q_1, q_2 \}$ $\Sigma = \{ 0, 1 \}$ $F = \{ q_1 \}$

  where the transition function is given by

  $\delta(q_0, 0) = q_0$ $\delta(q_0, 1) = q_1$ $\delta(q_1, 0) = q_0$

  $\delta(q_1, 1) = q_2$ $\delta(q_2, 0) = q_2$ $\delta(q_2, 1) = q_1$

Transition Graphs

- A DFA can be visualized with a *Transition Graph*
- The graph below represents the dfa in Example 2.1:

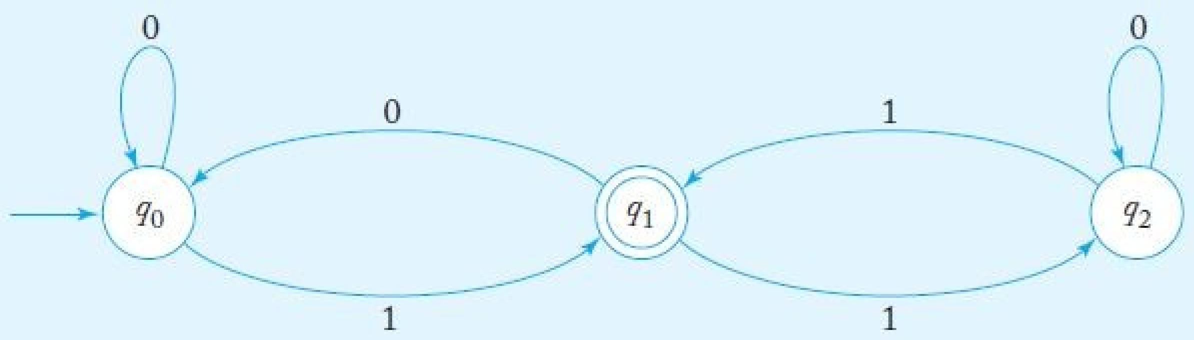


FIGURE 2.1

Processing Input with a DFA

- A DFA starts by processing the leftmost input symbol with its control in state $q_{0.}$ The transition function determines the next state, based on current state and input symbol
- The DFA continues processing input symbols until the end of the input string is reached
- The input string is *accepted* if the automaton is in a final state after the last symbol is processed. Otherwise, the string is *rejected*.
- For example, the dfa in example 2.1 accepts the string 111 but rejects the string 110

The Language Accepted by a DFA

- For a given dfa, the extended transition function δ^* accepts as input a dfa state and an input string. The value of the function is the state of the automaton after the string is processed.
- Sample values of δ^* for the dfa in example 2.1,

$$\delta^*(q_0, 1001) = q_1 \qquad \delta^*(q_1, 000) = q_0$$

- *The language accepted by a dfa* M is the set of all strings accepted by M. More precisely, the set of all strings w such that $\delta^*(q_0, w)$ results in a final state.

A Sample Deterministic Finite Accepter

- Example 2.3 shows a dfa to accept the set of all strings on { a, b } that start with the prefix ab.

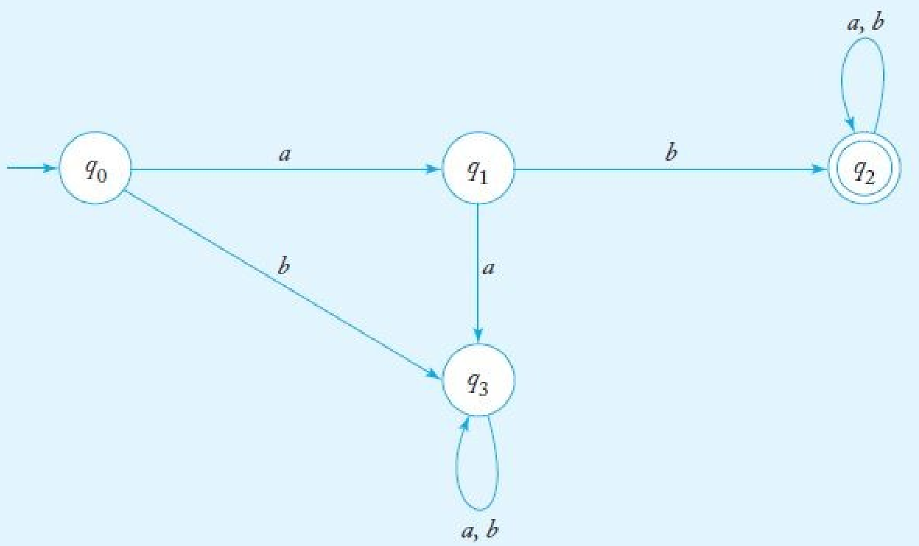


FIGURE 2.4

Regular Languages

- Finite automata accept a family of languages collectively known as *regular languages*.
- A language L is *regular* if and only if there is a DFA that accepts L. Therefore, to show that a language is regular, one must construct a DFA to accept it.
- Practice: show that $L = \{(ab)^n a, n > 0\}$ is regular.
- Regular languages have wide applicability in problems that involve scanning input strings in search of specific patterns.

Nondeterministic Finite Accepters

- An automaton is nondeterministic if it has a choice of actions for given conditions
- Basic differences between deterministic and nondeterministic finite automata:
  - In an nfa, a (state, symbol) combination may lead to several states <u>simultaneously</u>
  - If a transition is labeled with the empty string as its input symbol, the nfa may change states <u>without consuming input</u>
  - an nfa may have <u>undefined transitions</u>

Nondeterministic FA Example 2.7

- Example 2.7 shows a nondeterministic fa in which there are two transitions labeled a out of state q_0
- More pr

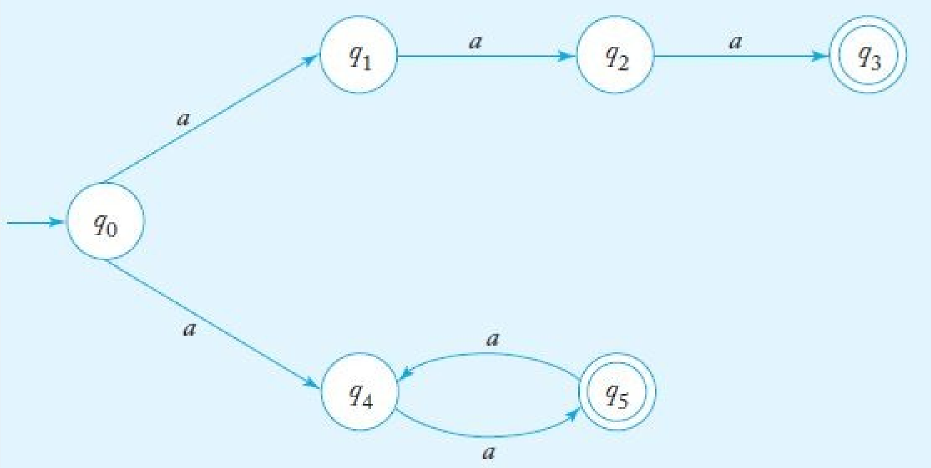


FIGURE 2.8

Nondeterministic FA Example 2.8

- Example 2.8 shows a nondeterministic fa which contains both a λ-transition as well as undefined transitions
- More precisely, $\delta(q_0, \lambda) = \{ q_2 \}$ and $\delta(q_2, 0) = \emptyset$

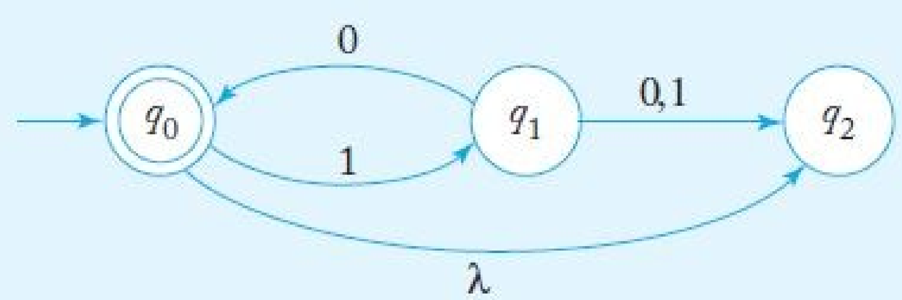


FIGURE 2.9

The Language Accepted by a Nondeterministic FA

- For a given nfa, the value of the extended transition function $\delta^*(q_i, w)$ is the set of all possible states for the control unit after processing w, having started in q_i
- Sample values of δ^* for the nfa in example 2.8:

  $\delta^*(q_0, 10) = \{ q_0, q_2 \}$ $\delta^*(q_0, 101) = \{ q_1 \}$

- A string w is accepted if $\delta^*(q_0, w)$ contains a final state. In the example above, 10 would be accepted but 101 would be rejected.
- As is the case with dfa, *the language accepted by a nondeterministic fa* M is the set of all accepted strings. The machine in example 2.8 accepts $L = \{ (10)^n : n \geq 0 \}$

Equivalence of Deterministic and Nondeterministic Finite Accepters

- Does nondeterminism make it possible to accept languages that deterministic fa cannot recognize?
- As it turns out, per Theorem 2.2: For any nondeterministic finite accepter, there is an equivalent deterministic finite accepter
- Therefore, every language accepted by a nondeterministic finite accepter is regular
- To prove theorem 2.2, a *constructive proof* is given. The algorithm outlines the steps to follow when building a dfa equivalent to a particular nfa

Procedure: nfa-to-dfa Conversion

1. Beginning with the start state, define input transitions for the dfa as follows:
   - If the nfa input transition leads to a single state, replicate for the dfa
   - If the nfa input transition leads to more than one state, create a new state in the dfa labeled $\{ q_i, \ldots, q_j \}$, where $q_i, \ldots, q_j$ are all the states the nfa transition can lead to.
   - If the nfa input transition is not defined, the corresponding dfa transition should lead to a trap state.
2. Repeat step 1 for all newly created dfa states, until no new states are created.
3. Any dfa state containing an nfa final state in its label should be labeled as final.
4. If the nfa accepts the empty string, label the start dfa state a final state.

nfa-to-dfa Conversion Example

- When applying the conversion procedure to the nfa below, we note the following nfa transitions

$\delta(q_0, 0) = \{ q_0, q_1 \}$ $\delta(q_0, 1) = \{ q_1 \}$

$\delta(q_1, 0) = \{ q_2 \}$ $\delta(q_1, 1) = \{ q_2 \}$

$= \{ q_2 \}$

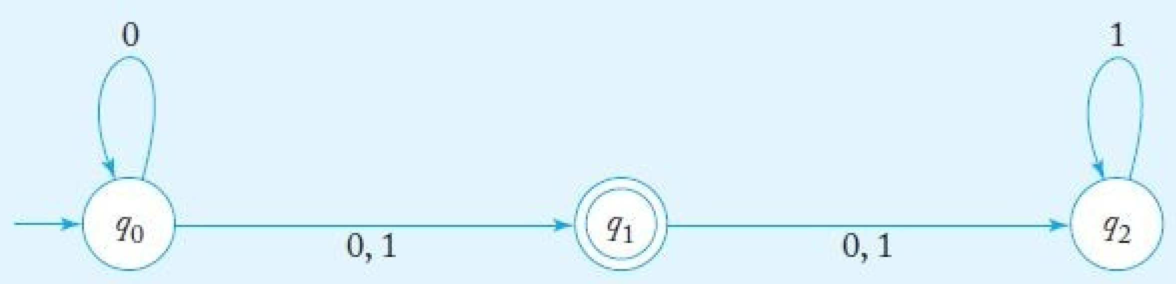


FIGURE 2.14

nfa-to-dfa Conversion Example (cont.)

- We add transitions from q_0 to states $\{ q_0, q_1 \}$ and $\{ q_1 \}$
- Note that $\delta(q_0, 0) \cup \delta(q_1, 0) = \{ q_0, q_1, q_2 \}$ and

  $\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1) = \{ q_1, q_2 \}$
- So we add trans … s $\{ q_0, q_1, q_2 \}$ and $\{ q_1, q_2 \}$

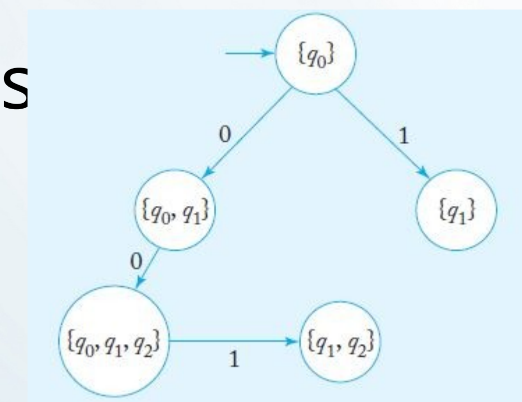


FIGURE 2.15

nfa-to-dfa Conversion Example (cont.)

- Note that $\delta(q_1, 0) \cup \delta(q_2, 0) = \{ q_2 \}$ and

  $\delta(q_1, 1) \cup \delta(q_2, 1) = \{ q_2 \}$
- So we add 0-1 transitions from $\{ q_1, q_2 \}$ to $\{ q_2 \}$
- Similarly, we add 0-1 transitions from $\{ q_1 \}$ to $\{ q_2 \}$
- Since $\delta(q_2, 1) = \{ q_2 \}$, we add the corresponding transition
- Since $\delta(q_2, 0)$ is undefined, we add a trap state (labeled $\varnothing$) as well as the corresponding transitions.

nfa-to-dfa Conversion Example (cont.)

- Since there are no dfa states with undefined transitions, the process stops. All states containing q_1 in their label are designated as final states.

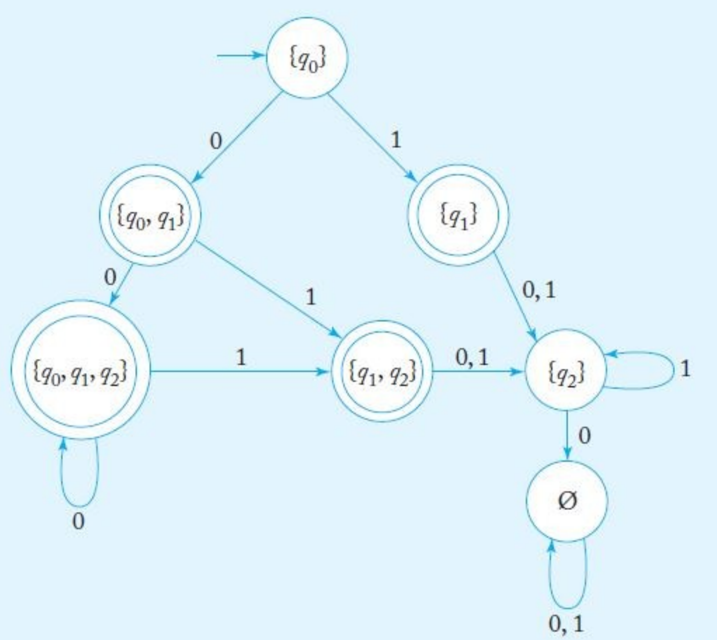


FIGURE 2.16