Chapter 1 - Regular Languages

Sipser, M. (2013). *Introduction to the Theory of Computation* (3rd ed.). Cengage Learning.

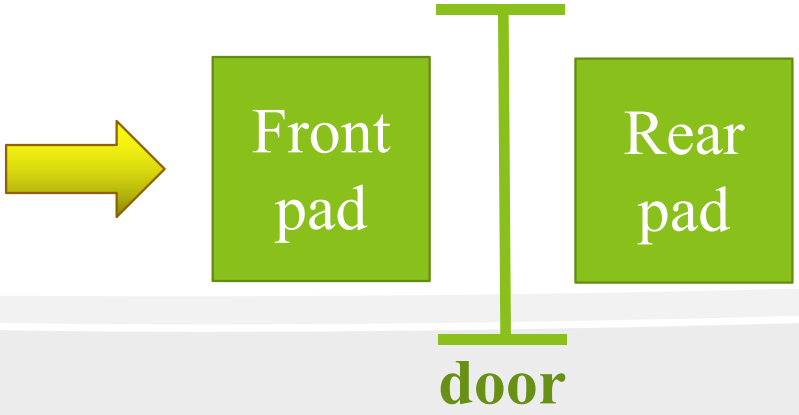
Contents

Chapter 1 - Regular Languages

- The theory of computation starts by asking what a computer is.
  - Real computers are too complex to study directly.
  - It uses simplified *computational models* that capture the features we care about, even if they aren't perfect.
- Different models are used for different purposes, starting with the simplest: the ***finite automaton*** (***finite state machine***).
- A regular language is a set of strings over an alphabet that can be recognized by a finite automaton.

1. Finite Automata

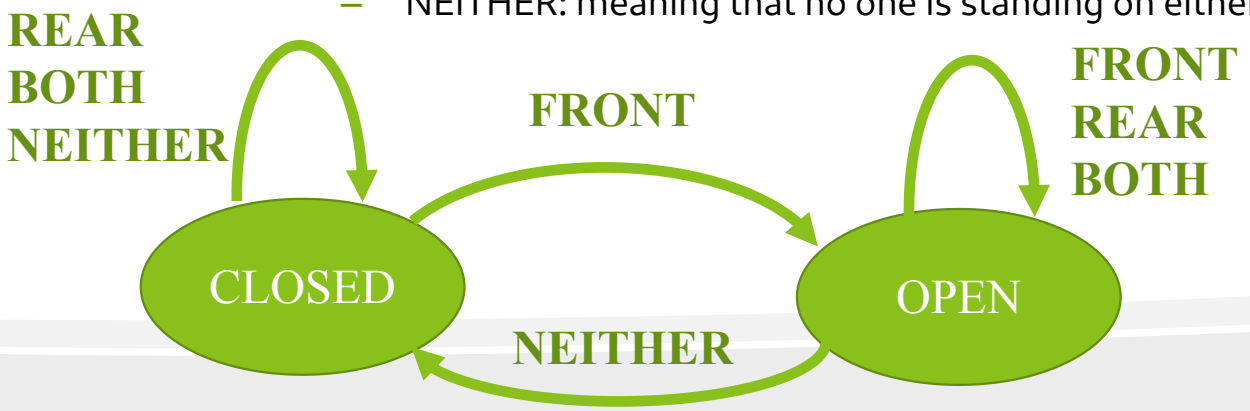
- Finite automata are good models for computers with an extremely limited amount of memory.
  - What can a computer do with such a small memory?
    - Many useful things!
    - The controller for an automatic door is one example of such a device. Often found at supermarket entrances and exits, automatic doors swing open when the controller senses that a person is approaching.



- An automatic door has a pad in the front to detect the presence of a person about to wolk through the doorway.
- Another pad is located to the rear so that controller can hold the door open long enough for the person to pass all the way.

1. Finite Automata

- The controller is in either of two states representing the corresponding condition of the door:
  - OPEN: The door is open.
  - CLOSED: The door is closed.
- There are four possible input conditions:
  - FRONT: meaning that a person is standing on the pad in front of the doorway
  - REAR: meaning that a person is standing on the pad to the rear of the doorway
  - BOTH: meaning that people are standing on both pads
  - NEITHER: meaning that no one is standing on either pad



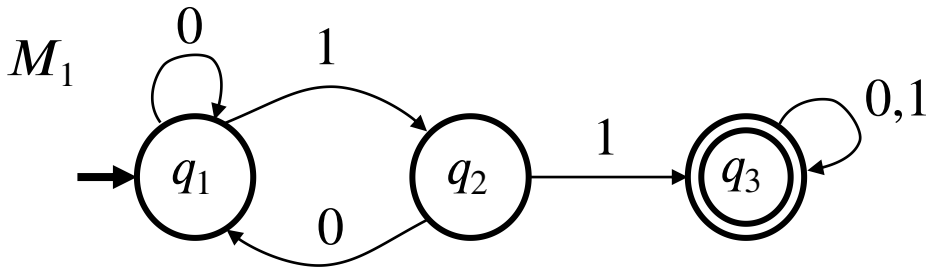
State diagram for an automatic door controller

		Input Signal			
		NEITHER	FRONT	REAR	BOTH
State	CLOSED	CLOSED	OPEN	CLOSED	CLOSED
	OPEN	CLOSED	OPEN	OPEN	OPEN

State transition table for an automatic door controller

1. Finite Automata

- The following depicts a finite automation M_1.

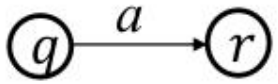


- States: q_1, q_2, q_3
- Transitions: $\xrightarrow{1}$
- Start State: →○
- Accept States: ◎

- **Input:** finite string
- **Output:** Accept or Reject
- **Computation process:** Begin at start state, read input symbols, follow corresponding transitions,
  - Accept if end with accept state,
  - Reject if not.

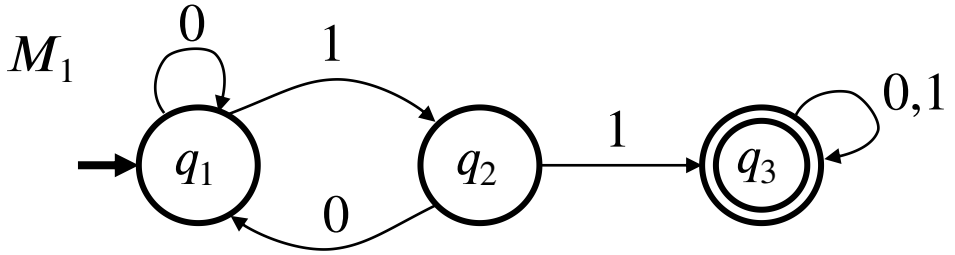
- **Examples:** 01101 → Accept, 00101 → Reject
  - M_1 accepts exactly those string in A where $A = \{w | \ w \text{ contains substring } 11\}$.
  - Say that A is the language of M_1 and that M_1 recognizes A and that $A = L(M_1)$.

1. Finite Automata
Fomal Definition of a Finite Automaton

- Definition: A ***finite automation*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where
  - 1. Q is a finite set called the ***states***.
  - 2. Σ is a finite set called the ***alphabet***.
  - 3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***.
    - $\delta(q, a) = r$ means 

  - 4. $q_0 \in Q$ is the ***start state***.
  - 5. $F \subseteq Q$ is the set of ***accept states*** (***final states***).

- Example:



- $M_1 = (Q, \Sigma, \delta, q_1, F)$
  1. $Q = \{q_1, q_2, q_3\}$
  2. $\Sigma = \{0, 1\}$
  3. δ is described by
  4. q_1 is the start state
  5. $F = \{q_3\}$

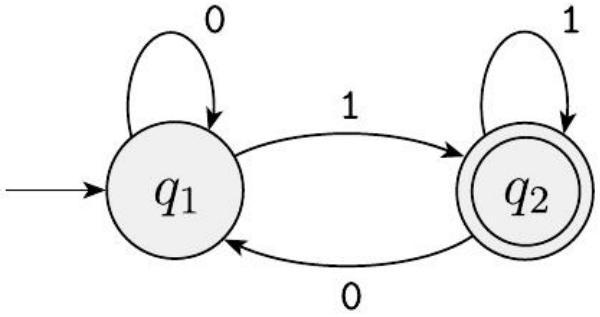
δ	0	1
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_3	q_3

1. Finite Automata
Fomal Definition of a Finite Automaton

- A string is a finite sequence of symbols in Σ.
  - The empty string ε is the string of length 0.
- A language is a set of strings (finite or infinite)
  - The empty language $\varnothing$ is the set with no strings.
- If *A* is the set of all strings that machine *M* accepts, we say that *A* is the ***language of machine M*** and write $L(M) = A$. We say that ***M recognizes A*** (or that ***M accepts A***).
- If the machine accepts no strings, it still recognizes one language—namely, the empty language $\varnothing$.

1. Finite Automata
Examples of Finite Automata

- Example 1.7: Consider the finite automaton M_2.



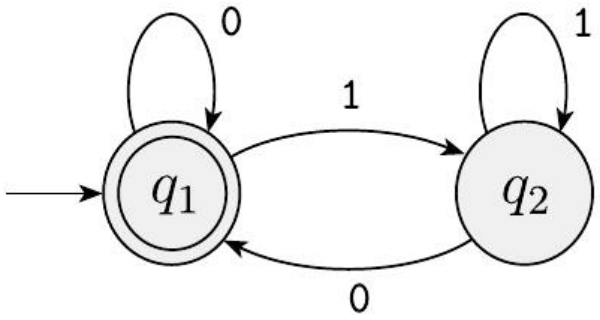
- $M_2 = (\{q_1, q_2\}, \{0, 1\}, \delta, q_1, \{q_2\})$
  - The transition function δ is

	0	1
q_1	q_1	q_2
q_2	q_1	q_2.

$L(M_2) = \{w | w \text{ ends in a } 1\}$

1. Finite Automata
Examples of Finite Automata

- Example 1.9: Consider the finite automaton M_3.



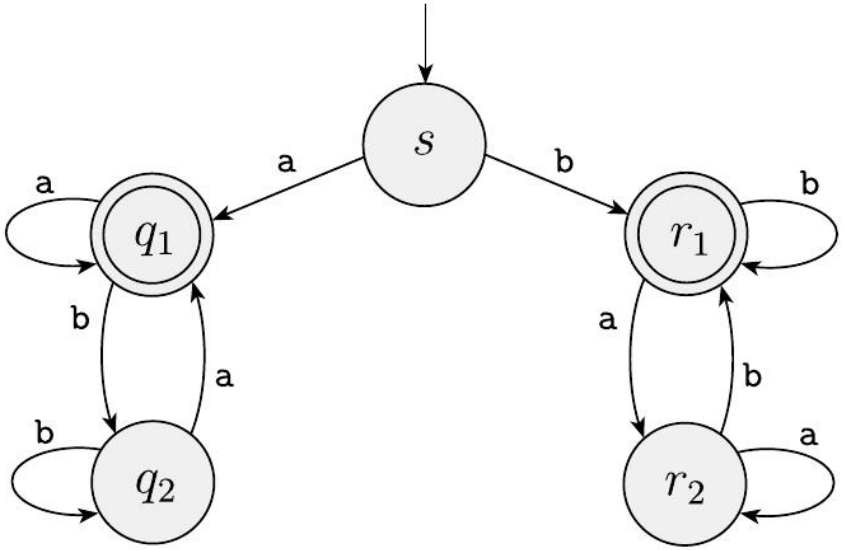
- $M_3 = (\{q_1, q_2\}, \{0, 1\}, \delta, q_1, \{q_1\})$
  - The transition function δ is

	0	1
q_1	q_1	q_2
q_2	q_1	q_2.

$L(M_3) = \{w | w$ is the empty string ε or ends in a $0\}$

1. Finite Automata
Examples of Finite Automata

- Example 1.11: Consider the five-state machine M_4.

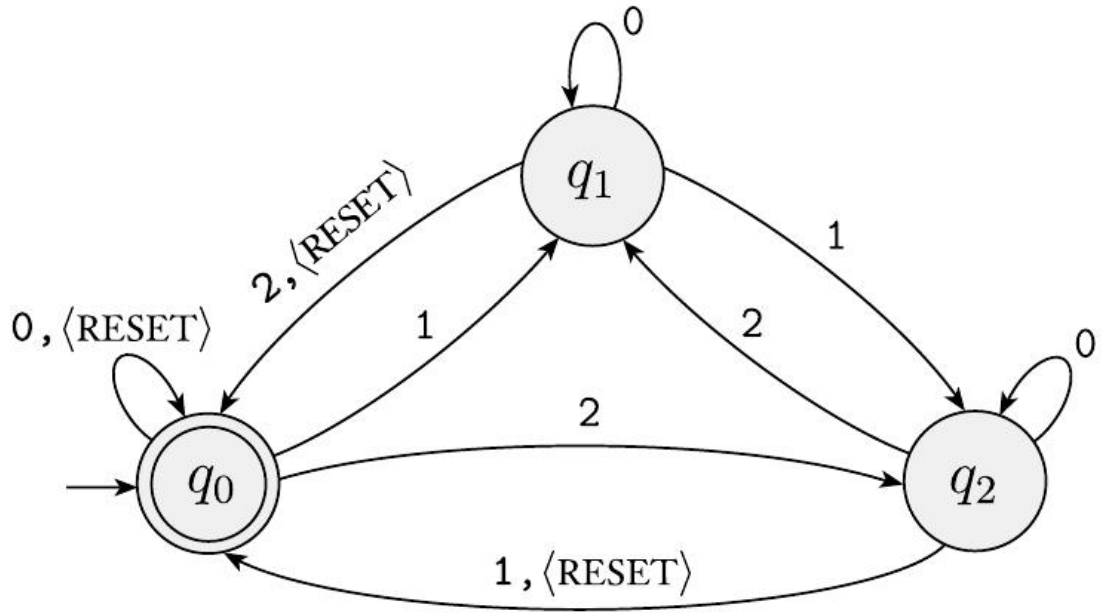


- $M_4 = (\{s, q_1, q_2, r_1, r_2\}, \{a, b\}, \delta, s, \{q_1, r_1\})$

$$L(M_4) = \{w | w \text{ is a string start and end with } a \text{ or that start and end with } b.\}$$

1. Finite Automata
Examples of Finite Automata

- Example 1.13: Consider the three-state machine M_5 which has a four-symbol input alphabet, Σ = {<RESET>, 0, 1, 2}. We treat <RESET> as a single symbol.



- $M_5 = (\{q_0, q_1, q_2\}, \{\langle\text{RESET}\rangle, 0, 1, 2\}, \delta, q_0, \{q_0\})$

- Machine M_5 keeps a running count of the sum of the numerical input symbols it reads, modulo 3.
- Every time it receives the <RESET> symbol, it resets the count to 0.
- It accepts if the sum is 0 modulo 3, or in other words, if the sum is a multiple of 3.

1. Finite Automata
Formal Definition of Computation

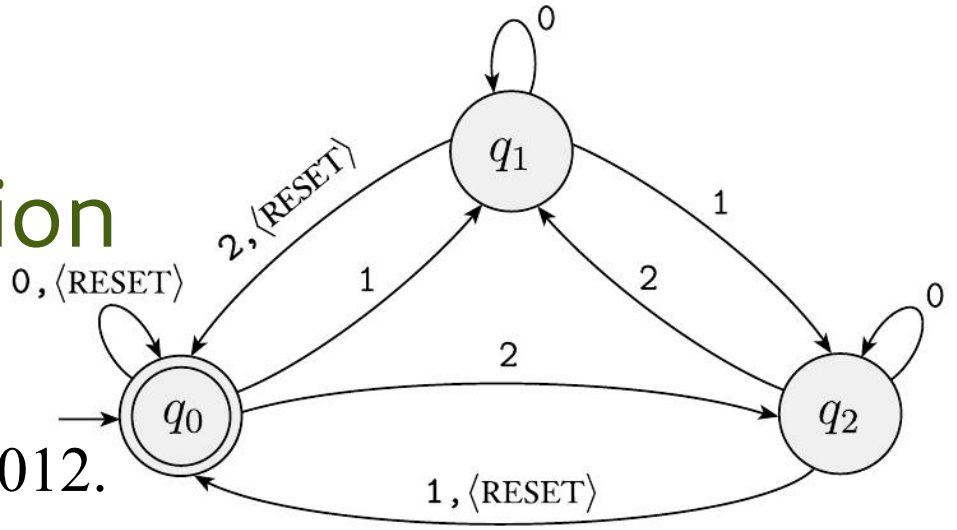
- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \ldots w_n$ be a string where each w_i is a member of the alphabet Σ. Then M accepts w if a sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with three conditions:
  - 1. $r_0 = q_0$,
  - 2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \ldots, n-1$, and
  - 3. $r_n \in F$
- We say that M recognizes language A if $A = \{w \mid M \text{ accepts } w\}$.

1. Finite Automata
Formal Definition of Computation

Definition 1.16:

- *A language aw* is called a ***regular language*** if some finite automaton recognizes it.

1. Finite Automata
Formal Definition of Computation



- Exmaple 1.17: Consider the machine M_5.
  - Let w be the string 10<RESET>22<RESET>012.
  - M_5 accpets w because there exists a sequence of states it enters when computing on w is
    - $q_0, q_1, q_1, q_0, q_2, q_1, q_0, q_0$, q1, q0

  which satisfies the three conditions.
  - The language of M_5 is

  $L(M_5) = \{w |$ The sum of the symbols in w is 0 modulo 3, except that <RESET> resets the count to 0$\}$.
  - $L(M_5)$ is a regular language.

1. Finite Automata
The Regular Operations

Definition 1.23

- Let A and B be languages. We define the regular operations ***union***, ***concatenation***, and ***star*** as follows:
  - **Union:** $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
  - **Concatenation:** $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$
  - **Star:** $A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$
    - Note: $\varepsilon \in A^*$ always

1. Finite Automata
The Regular Operations

- Example 1.24: Let the alphabet Σ be the standard 26 letters {*a*, *b*, . . . , *z*}. If A = {good, bad} and B = {*boy*, *girl*}, then
  - $A \cup B = \{\text{good, bad, boy, girl}\}$
  - $A \circ B = \{\text{goodboy, goodgirl, badboy, badgirl}\}$
  - $A^* = \{\varepsilon, \text{good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, goodbadgood, goodbadbad, . . . }\}$

The Regular Operations

Closure Properties

- A collection of objects is ***closed*** under some operation if applying that operation to members of the collection returns an object still in the collection.
- Let $\mathcal{N} = \{1, 2, 3, \ldots\}$ be the set of natural numbers.
  - $\mathcal{N}$ is *closed under multiplication*, we mean that for any x and y in $\mathcal{N}$, the product $x \times y$ also is in $\mathcal{N}$.
  - In contrast, $\mathcal{N}$ is not closed under division, as 1 and 2 are in $\mathcal{N}$ but 1/2 is not.

Closure Properties

- **Theorem 1.25**: The class of regular languages is closed under the union operation.
  - In other words, if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Closure Properties

- **Theorem 1.26**: The class of regular languages is closed under the concatenation operation.
  - In other words, if A_1 and A_2 are regular languages, so is $A_1 \circ A_2$.

2. Nondeterminism

- Nondeterminism as a significant concept in the theory of computation.
- **Deterministic Computation**: When the machine is in a given state and reads the next input symbol, we know what the next state will be—it is determined.
- **Nondeterministic Computation**: The process allows for branching. At any given point, the machine may have several choices for the next state based on the same input.
- Nondeterminism is a generalization of determinism.
  - Every deterministic finite automaton (DFA) is automatically a nondeterministic finite automaton (NFA).

2. Nondeterminism

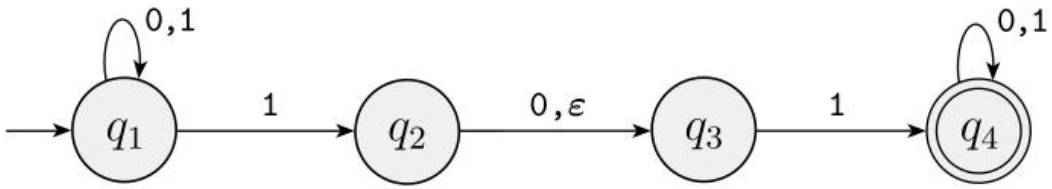


FIGURE **1.27**
The nondeterministic finite automaton N_1

Nondeterminism doesn't correspond to a physical machine we can build. However, it is useful mathematically.

- The difference between a DFA and a NFA
  - 1. Every state of a DFA always has exactly one exiting transition arrow for **each** symbol in the alphabet.
    - In an NFA, a state may have zero, one, or many exiting arrows for each alphabet symbol.
  - 2. In a DFA, labels on the transition arrows are symbols from the alphabet. This NFA has an arrow with the label ε.
    - In general, an NFA may have arrows labeled with members of the alphabet or ε. Zero, one, or many arrows may exit from each state with the label ε.

2. Nondeterminism

- **New features of nondeterminism:**
  - multiple paths possible (0, 1 or many at each step)
  - ε-transition is a "free" move without reading input
  - Accept input if some path leads to accept

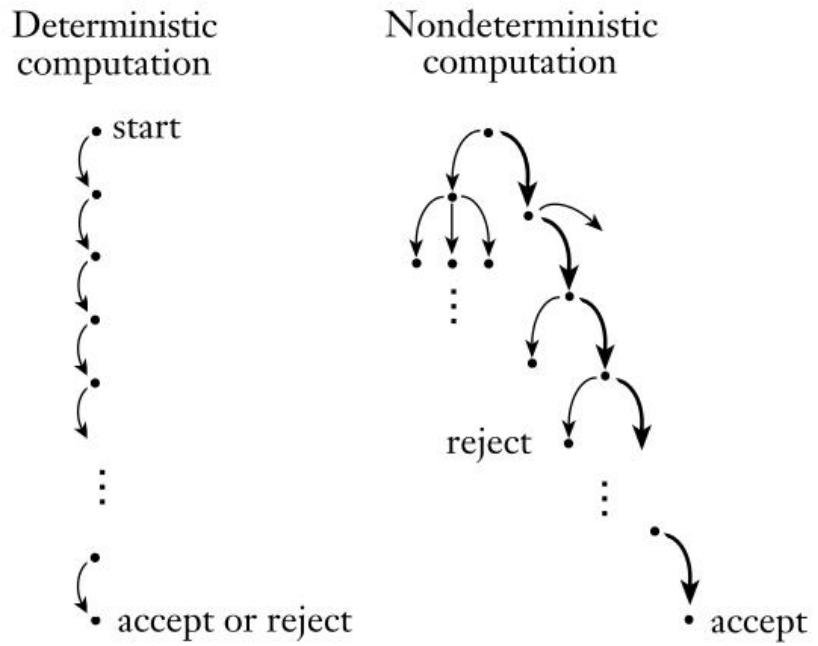


FIGURE **1.28**
Deterministic and nondeterministic computations with an accepting branch

2. Nondeterminism

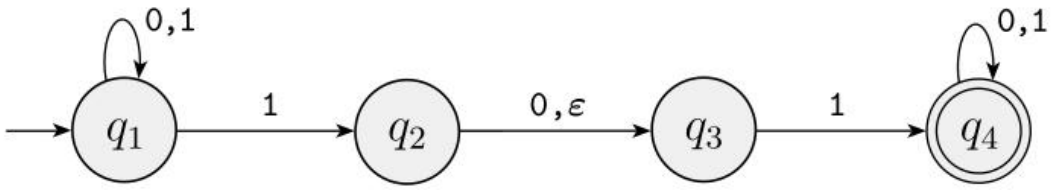


FIGURE **1.27**
The nondeterministic finite automaton N_1

- The computation of N_1 on input 010110 is depicted in the following figure.

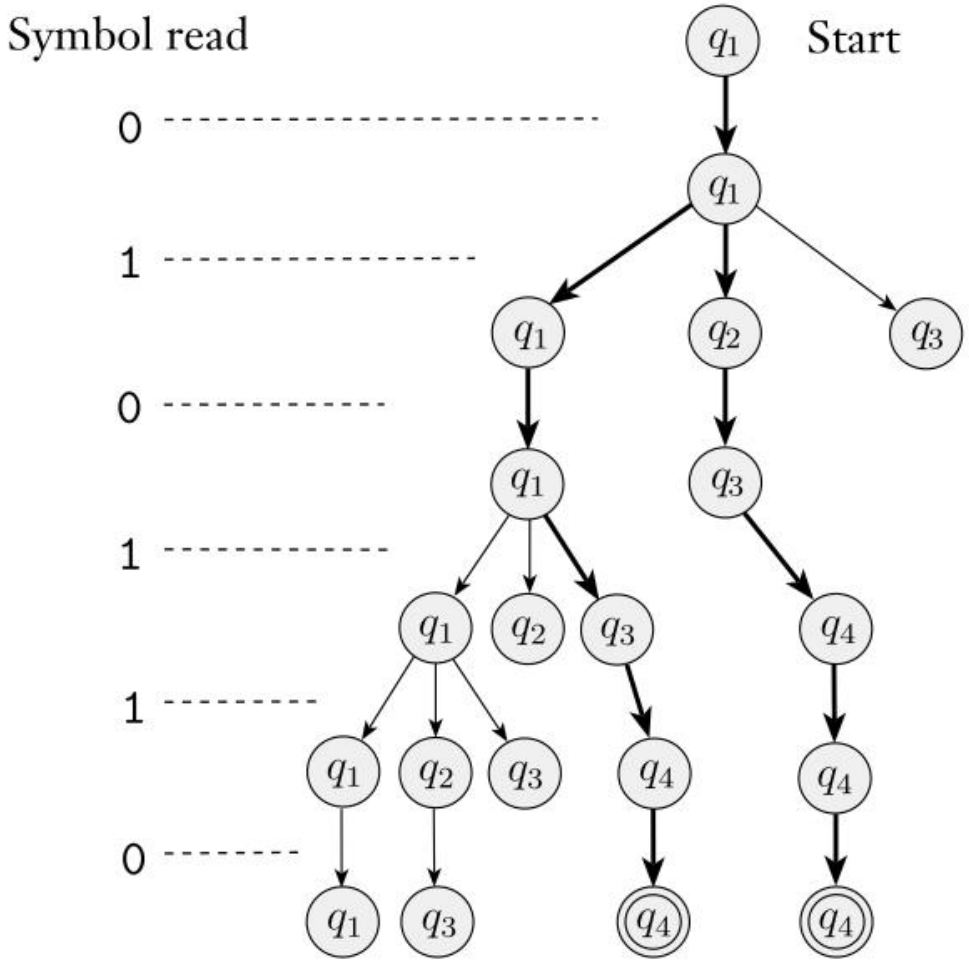


FIGURE **1.29**
The computation of N_1 on input 010110

2. Nondeterminism

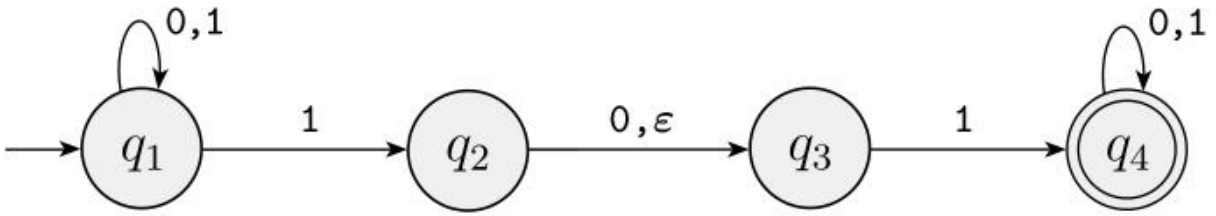


FIGURE **1.27**
The nondeterministic finite automaton N_1

- **New features of nondeterminism:**
  - multiple paths possible (0, 1 or many at each step)
  - ε-transition is a "free" move without reading input
  - Accept input if some path leads to ◎ accept

- Example inputs:
  - 11: Accept
  - 01: Reject
  - 101: Accept
  - 100: Reject

2. Nondeterminism

- Every NFA can be converted into an equivalent DFA.
- Constructing NFAs is sometimes easier than directly constructing DFAs.
- An NFA may be much smaller than its deterministic counterpart, or its functioning may be easier to understand.

2. Nondeterminism

- **Example 1.30.** Let A be the language consisting of all strings over {0,1} containing a 1 in the third position from the end (e.g., 000100 is in A but 0011 is not). The following four-state NFA N_2 recognizes A.

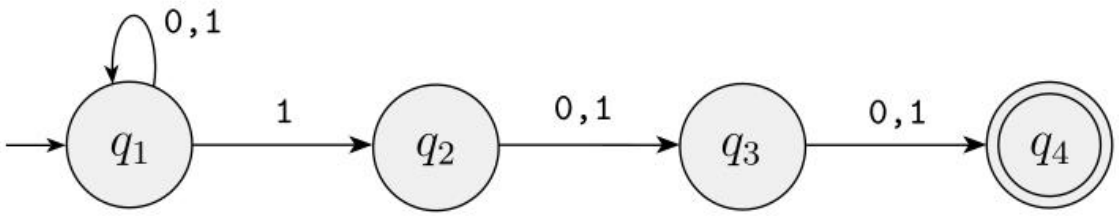


FIGURE **1.31**
The NFA N_2 recognizing A

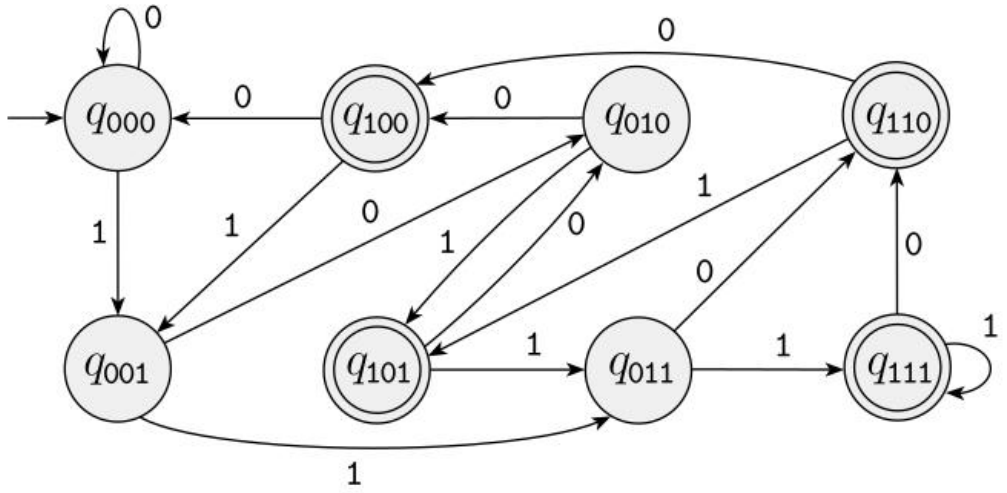


FIGURE **1.32**
A DFA recognizing A

2. Nondeterminism

- Example 1.33. The following NFA N_3 has an input alphabet {0} consisting of a single symbol. An alphabet containing only one symbol is called a ***unary alphabet***.

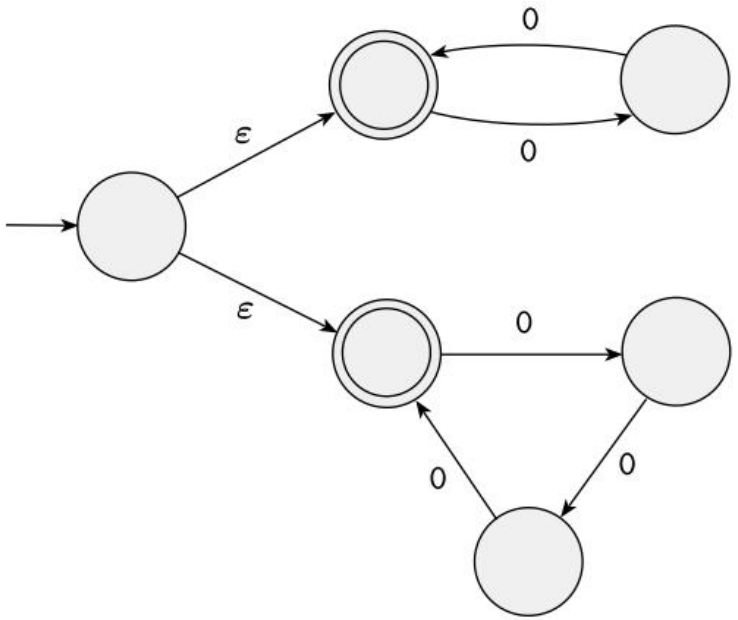


FIGURE **1.34**
The NFA N_3

This machine demonstrates the convenience of having ε arrows. It accepts all strings of the form 0^k where k is a multiple of 2 or 3. (Remember that the superscript denotes repetition, not numerical exponentiation.)

2. Nondeterminism

- Example 1.35. Practice with N_4 to satisfy yourself that it accepts the strings ε, a, baba, and baa, but that it doesn't accept the strings b, bb, and babba.

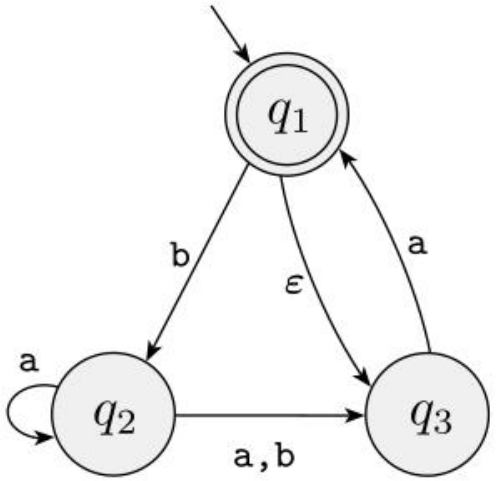


FIGURE **1.36**
The NFA N_4

2. Nondeterminism
Formal Definition of a Nondeterministic Finite Automaton

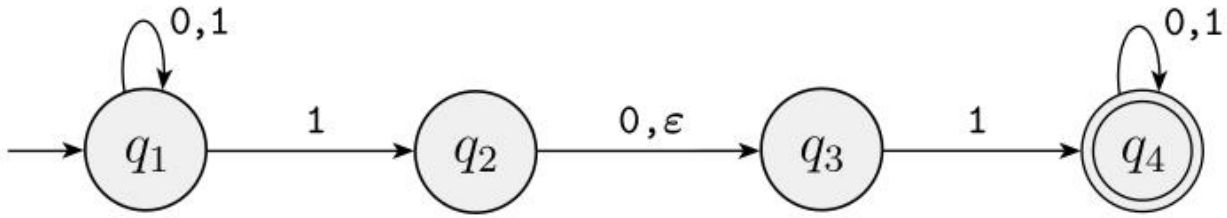
Definition 1.37.

- A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where
  - 1. Q is a finite set of states,
  - 2. Σ is a finite alphabet,
  - 3. $\delta: Q \times \Sigma_\varepsilon \to P(Q)$ is the transition function,
  - 4. $q_0 \in Q$ is the start state, and
  - 5. $F \subseteq Q$ is the set of accept states.

- $P(Q)$: Power set of Q (the collection of all subsets of Q)
- $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$

2. Nondeterminism
Formal Definition of a Nondeterministic Finite Automaton

- **Example 1.38.** Recall the NFA N_1:



  - The formal description of N_1 is $(Q, \Sigma, \delta, q_1, F)$, where
    - 1. $Q = \{q_1, q_2, q_3, q_4\}$,
    - 2. $\Sigma = \{0,1\}$,
    - 3. δ is given as

	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$,

    - 4. q_1 is the start state, and
    - 5. $F = \{q_4\}$.

2. Nondeterminism
Formal Definition of Computation for an NFA

- The formal definition of computation for an NFA is similar to that for a DFA.
- Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and w a string over the alphabet Σ. Then we say that N ***accepts*** w if we can write w as $w = y_1 y_2 \cdots y_m$, where each y_i is a member of Σ_ε and a sequence of states $r_0, r_1, \ldots, r_m$ exists in Q with three conditions:
  - 1. $r_0 = q_0$,
  - 2. $r_{i+1} \in \delta(r_i, y_{i+1})$, for $i = 0, \ldots, m - 1$, and
  - 3. $r_m \in F$.

2. Nondeterminism

Equivalence of NFAs and DFAs

- Deterministic and nondeterministic finite automata recognize the same class of languages.
- Such equivalence is both surprising and useful.
  - It is surprising because NFAs appear to have more power than DFAs, so we might expect that NFAs recognize more languages.
  - It is useful because describing an NFA for a given language sometimes is much easier than describing a DFA for that language.
- Say that two machines are ***equivalent*** if they recognize the same language.

2. Nondeterminism
Equivalence of NFAs and DFAs

- **Theorem 1.39**
  - Every nondeterministic finite automaton has an equivalent deterministic finite automaton.
- The idea is to convert the NFA into an equivalent DFA that simulates the NFA.
  - If k is the number of states of the NFA, it has 2^k subsets of states. Each subset corresponds to one of the possibilities that the DFA must remember, so the DFA simulating the NFA will have 2^k states.

2. Nondeterminism
Equivalence of NFAs and DFAs

- Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A. We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing A.
- Let's first consider the easier case wherein N has no ε arrows. Later we take the ε arrows into account.
  - 1. $Q' = P(Q)$.
    - Every state of M is a set of states of N.
  - 2. For $R \in Q'$ and $a \in \Sigma$, let $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$.
    - Another way to write this expression is $\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$
  - 3. $q_0' = \{q_0\}$
  - 4. $F' = \{R \in Q \mid R \text{ contains an accept state of } N\}$

2. Nondeterminism
Equivalence of NFAs and DFAs

- Let consider the ε arrows
  - For any state R of M, we define $E(R)$ to be the collection of states that can be reached from members of R by going only along ε arrows, including the members of R themselves.

$$E(R) = \{q \mid q \text{ can be reached from } R \text{ by traveling along 0 or more } \varepsilon \text{ arrows}\}.$$

  - Replacing $\delta(r, a)$ by $E(\delta(r, a))$

$$\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a)) \text{ for some } r \in R\}.$$

  - Changing q_0' to be $E(\{q_0\})$

2. Nondeterminism
Equivalence of NFAs and DFAs

- Example 1.41. Convert NFA N_4 to a DFA D.

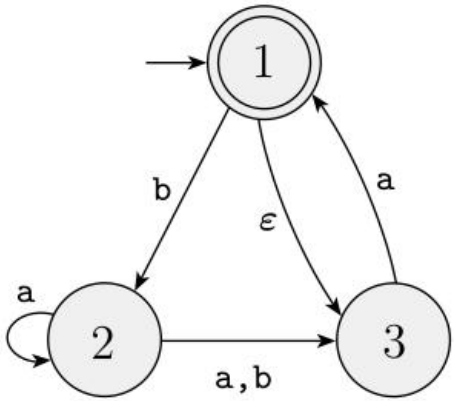


FIGURE **1.42**
The NFA N_4

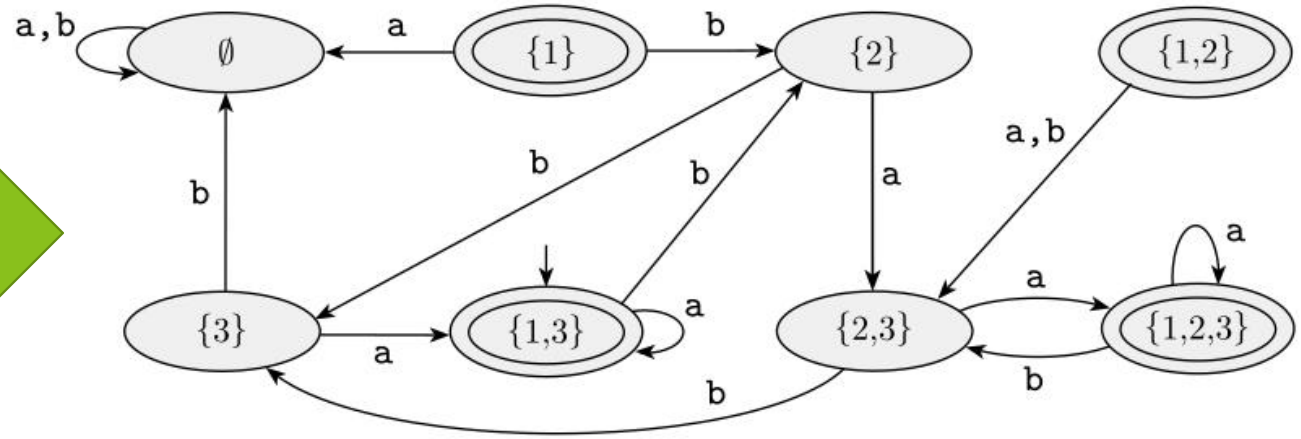


FIGURE **1.43**
A DFA D that is equivalent to the NFA N_4

- D' state set: $\{\varnothing, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$

2. Nondeterminism
Equivalence of NFAs and DFAs

- **Corollary 1.40**
  - A language is regular if and only if some nondeterministic finite automaton recognizes it.

2. Nondeterminism
Closure Under the Regular Operations

- **Theorem 1.45**
  - The class of regular languages is closed under the union operation.

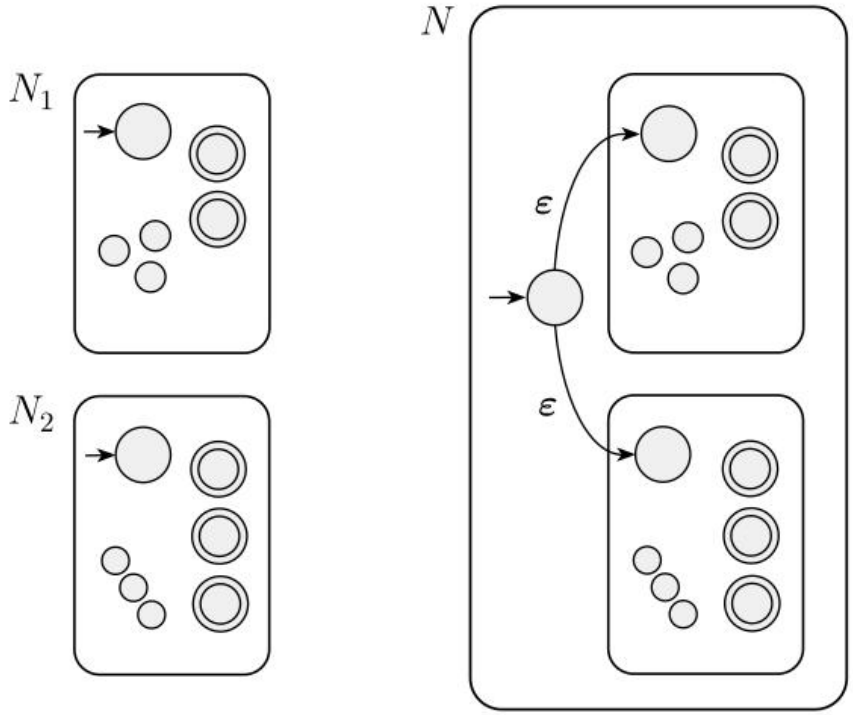


FIGURE **1.46**
Construction of an NFA N to recognize $A_1 \cup A_2$

2. Nondeterminism
Closure Under the Regular Operations

- **Theorem 1.47**
  - The class of regular languages is closed under the concatenation operation.

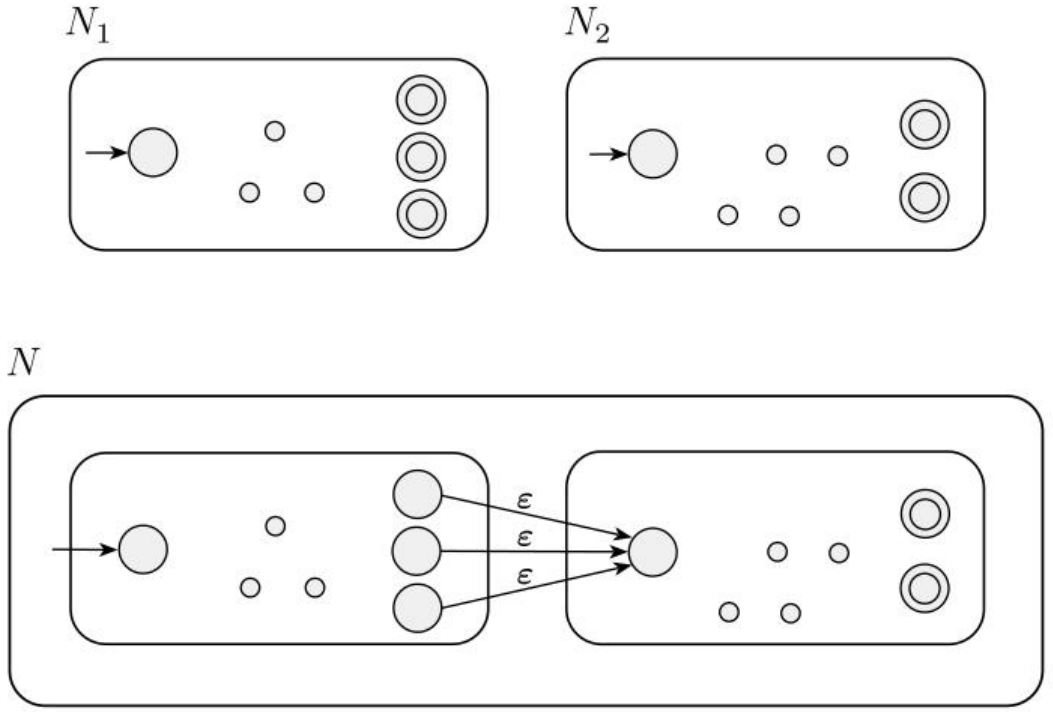


FIGURE **1.48**
Construction of N to recognize $A_1 \circ A_2$

2. Nondeterminism
Closure Under the Regular Operations

- **Theorem 1.49**
  - The class of regular languages is closed under the star operation.

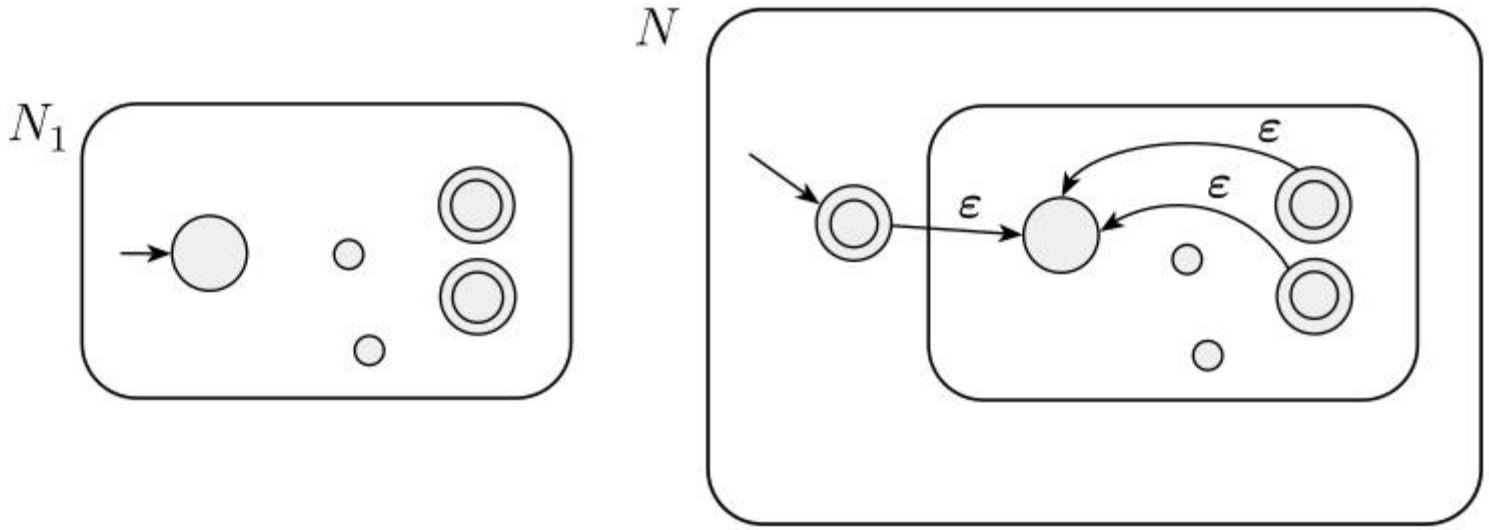


FIGURE 1.50
Construction of N to recognize A^*

3. Regular Expressions

- We can use the regular operations to build up expressions describing languages, which are called ***regular expressions***.
- In regular expressions, the star operation is done first, followed by concatenation, and finally union, unless parentheses change the usual order.
- If $\Sigma = \{0,1\}$, we can write Σ as shorthand for the regular expression $(0 \cup 1)$. If Σ is any alphabet, the regular expression Σ describes the language consisting of all strings of length 1 over this alphabet, and Σ^* describes the language consisting of all strings over that alphabet.

3. Regular Expressions
Formal Definition of a Regular Expression

Definition 1.52

R is a regular expression if R is

- 1. a for some a in the alphabet Σ,
- 2. ε,
- 3. $\varnothing$,
- 4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
- 5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
- 6. (R_1^*), where R_1 is a regular expression.

- In items 1 and 2, the regular expressions a and ε represent the languages $\{a\}$ and $\{\varepsilon\}$, respectively. In item 3, the regular expression $\varnothing$ represents the empty language.
- Let $R^+ = RR^*$, so $R^+ \cup \varepsilon = R^*$.
- $L(R)$ is the language of R.

3. Regular Expressions
Formal Definition of a Regular Expression

- **Example 1.53.** We assume that the alphabet Σ is {0,1}.
  - 1. $0^*10^* = \{w \mid w \text{ contains a single 1}\}$.
  - 2. $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one 1}\}$.
  - 3. $\Sigma^*001\Sigma^* = \{w \mid w \text{ contains the string 001 as a substring}\}$.
  - 4. $1^* (01^+)^* = \{w \mid \text{every 0 in } w \text{ is followed by at least one 1}\}$.
  - 5. $(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$.
  - 6. $(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of 3}\}$.
  - 7. $01 \cup 10 = \{01, 10\}$.

3. Regular Expressions
Formal Definition of a Regular Expression

- **Example 1.53.** We assume that the alphabet Σ is {0,1}.
  - 8. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{w \mid w$ starts and ends with the same symbol$\}$.
  - 9. $(0 \cup \varepsilon)1^* = 01^* \cup 1^*$.
    - The expression $0 \cup \varepsilon$ describes the language $\{0, \varepsilon\}$, so the concatenation operation adds either 0 or ε before every string in 1*.
  - 10. $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$.
  - 11. $1^*\emptyset = \emptyset$.
    - Concatenating the empty set to any set yields the empty set.
  - 12. $\emptyset^* = \{\varepsilon\}$.
    - The star operation puts together any number of strings from the language to get a string in the result. If the language is empty, the star operation can put together 0 strings, giving only the empty string.

3. Regular Expressions
Formal Definition of a Regular Expression

- If we let R be any regular expression, we have the following identities.
  - $R \cup \emptyset = R$.
  - $R \circ \varepsilon = R$.
- Elemental objects in a programming language, called ***tokens***, such as the variable names and constants, may be described with regular expressions.
  - For example, a numerical constant that may include a fractional part and/or a sign may be described as a member of the language

$$(+ \cup - \cup \varepsilon)(D^+ \cup D^+.D^* \cup D^*.D^+)$$

where $D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ is the alphabet of decimal digits. Examples of generated string are 12, 3.14, +7., -.123.

3. Regular Expressions
Equivalence With Finite Automata

- **Theorem 1.54**
  - A language is regular if and only if some regular expression describes it.
- Regular expressions and finite automata are equivalent in their descriptive power.
- Any regular expression can be converted into a finite automaton that recognizes the language it describes, and vice versa.

3. Regular Expressions
Equivalence With Finite Automata
Convert a Regular Expression to an Equivalent NFA

- **Example 1.56.** Convert the regular expression $(ab \cup a)^*$ to an NFA.

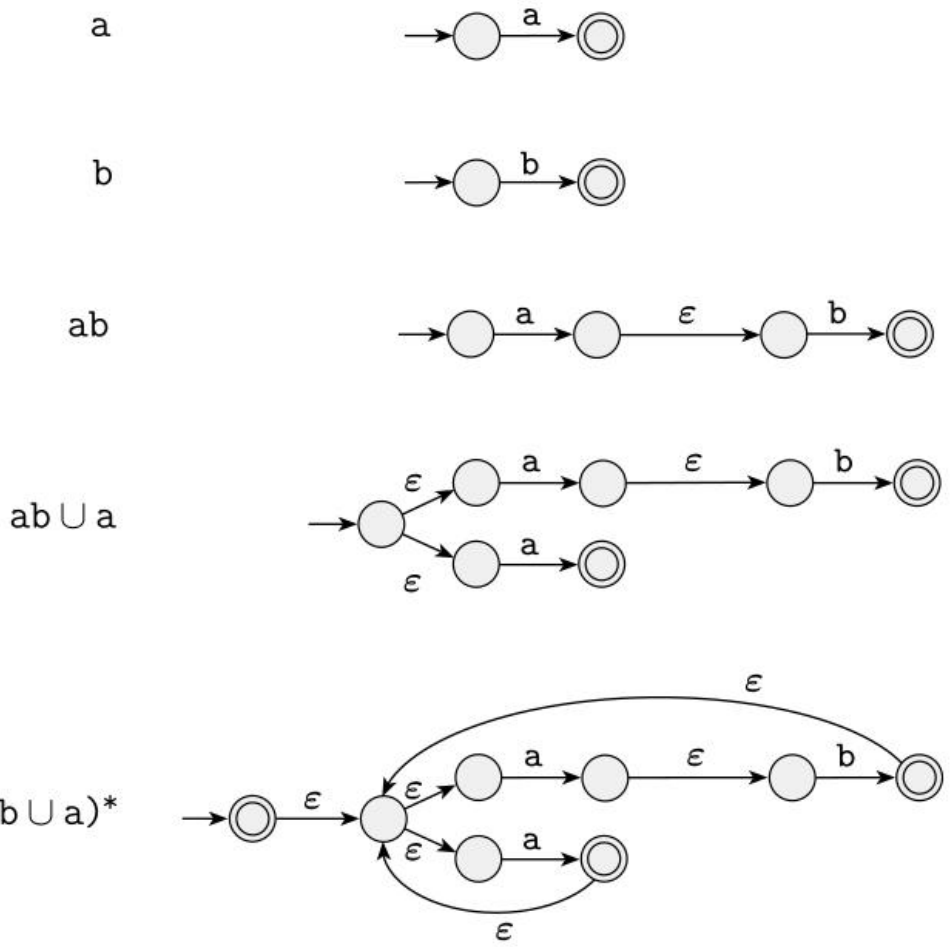


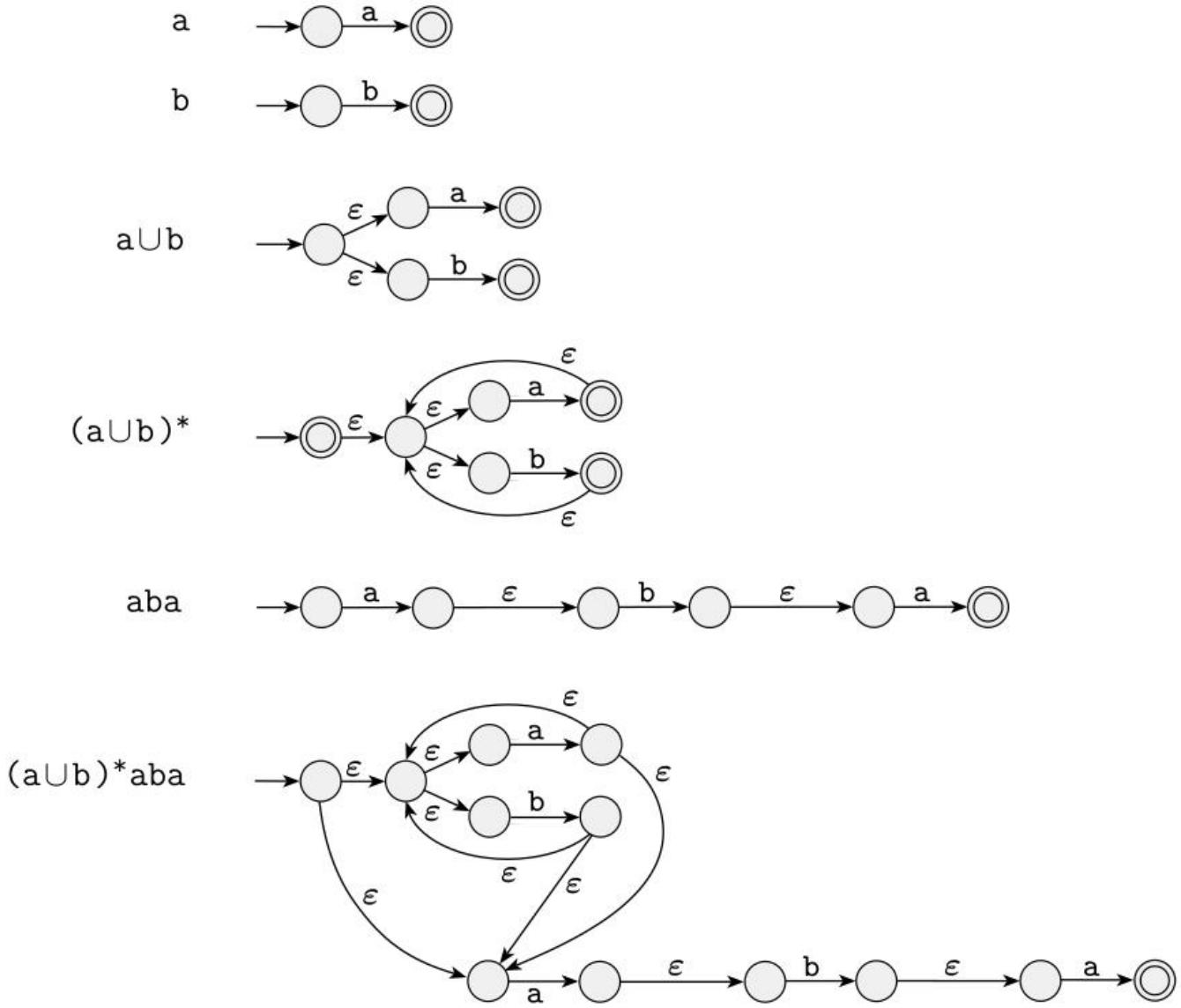
FIGURE **1.57**
Building an NFA from the regular expression $(ab \cup a)^*$

3. Regular Expressions

Equivalence With Finite Automata

Convert a Regular Expression to an Equivalent NFA

- **Example 1.58.** Convert the regular expression $(a \cup b)^* aba$ to an NFA.



3. Regular Expressions
Equivalence With Finite Automata
Convert a DFA into an Equivalent Regular Expression

- Convert a DFA into an Equivalent Regular Expression
  - 1. Convert DFAs into GNFAs (***General Nondeterministic Finite Automaton***)
  - 2. Convert GNFAs into regular expressions
- General Nondeterministic Finite Automaton
  - GNFAs are simply nondeterministic finite automata wherein the transition arrows may have any regular expressions as labels, instead of only members of the

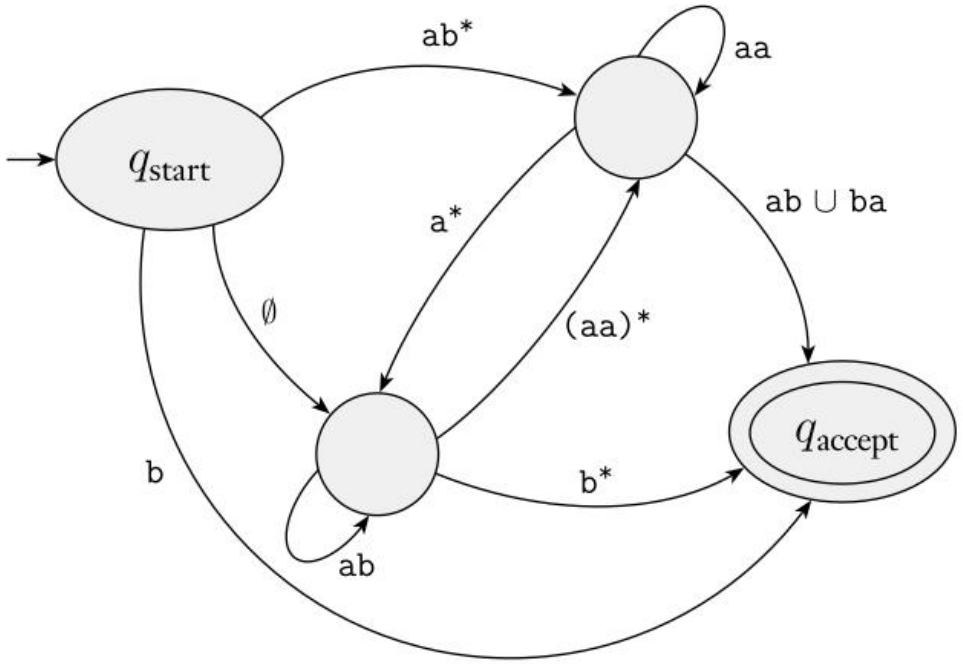


FIGURE 1.61
A generalized nondeterministic finite automaton

3. Regular Expressions
Equivalence With Finite Automata
Convert a DFA into an Equivalent Regular Expression

- For convenience, we require that GNFAs always have a **special form** that meets the following conditions.
  - The start state has transition arrows going to every other state but no arrows coming in from any other state.
  - There is only a single accept state, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.
  - Except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself.

3. Regular Expressions

Equivalence With Finite Automata

Convert a DFA into a GNFA (in the Special Form)

- Add a new start state with an ε arrow to the old start state and a new accept state with ε arrows from the old accept states.
- If any arrows have multiple labels (or if there are multiple arrows going between the same two states in the same direction), replace each with a single arrow whose label is the union of the previous labels.
- Finally, we add arrows labeled $\emptyset$ between states that had no arrows. This last step won't change the language recognized because a transition labeled with $\emptyset$ can never be used.

3. Regular Expressions

Equivalence With Finite Automata

Convert a GNFA into a Regular Expression

- Suppose that the GNFA has k states. Then, because a GNFA must have a start and an accept state and they must be different from each other, we know that $k \geq 2$.
  - If k > 2, we construct an equivalent GNFA with $k - 1$ states. This step can be repeated on the new GNFA until it is reduced to two states.
  - If $k = 2$, the GNFA has a single arrow that goes from the start state to the accept state. The label of this arrow is the equivalent regular expression.

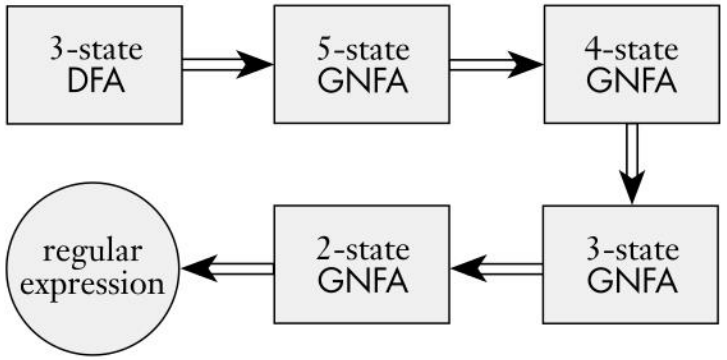


FIGURE 1.62
Typical stages in converting a DFA to a regular expression

3. Regular Expressions
Equivalence With Finite Automata
Convert a GNFA into a Regular Expression

- The crucial step is constructing an equivalent GNFA with one fewer state when $k > 2$.
  - Selecting a state, ripping it out of the machine, and repairing the remainder so that the same language is still recognized. Any state will do, provided that it is not the start or accept state. We are guaranteed that such a state will exist because $k > 2$. Let's call the removed state q_{rip}.

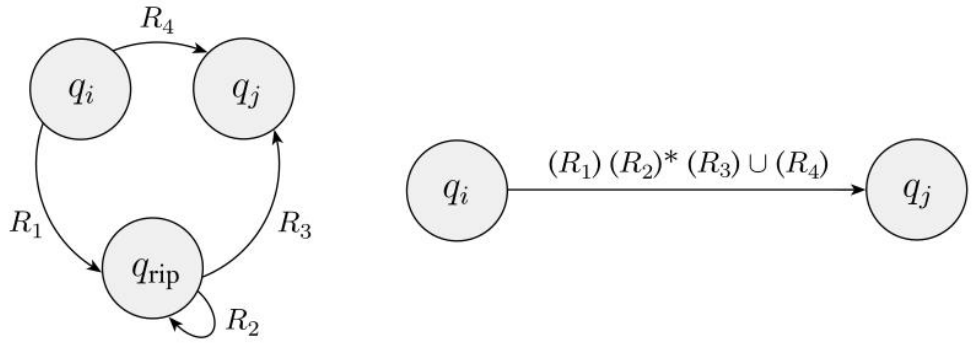
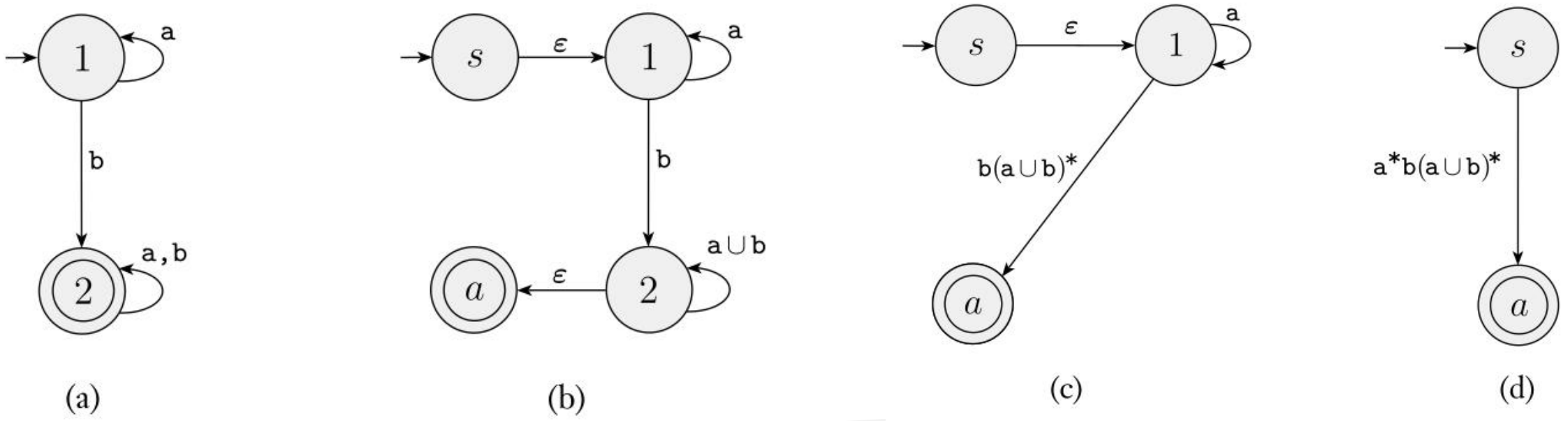


FIGURE 1.63
Constructing an equivalent GNFA with one fewer state

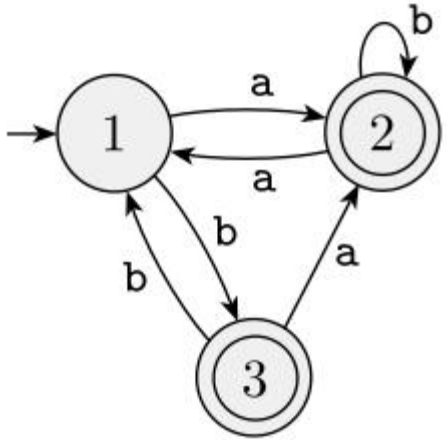
3. Regular Expressions
Equivalence With Finite Automata
Convert a DFA into an Equivalent Regular Expression

- **Example 1.66.** Converting a two-state DFA to an equivalent regular expression

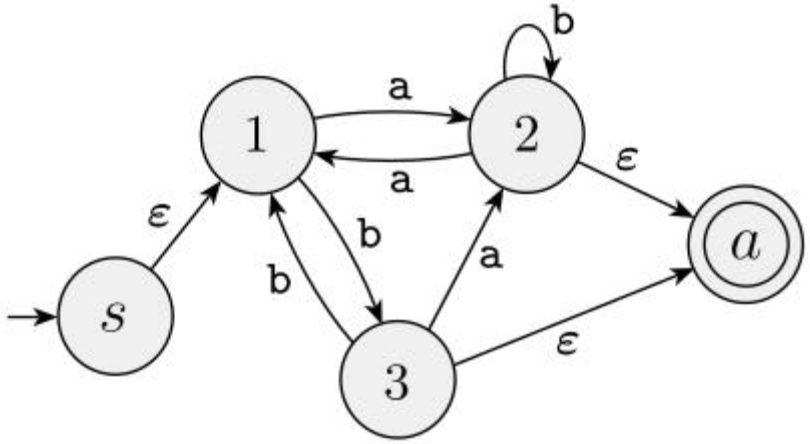


3. Regular Expressions
Equivalence With Finite Automata
Convert a DFA into an Equivalent Regular Expression

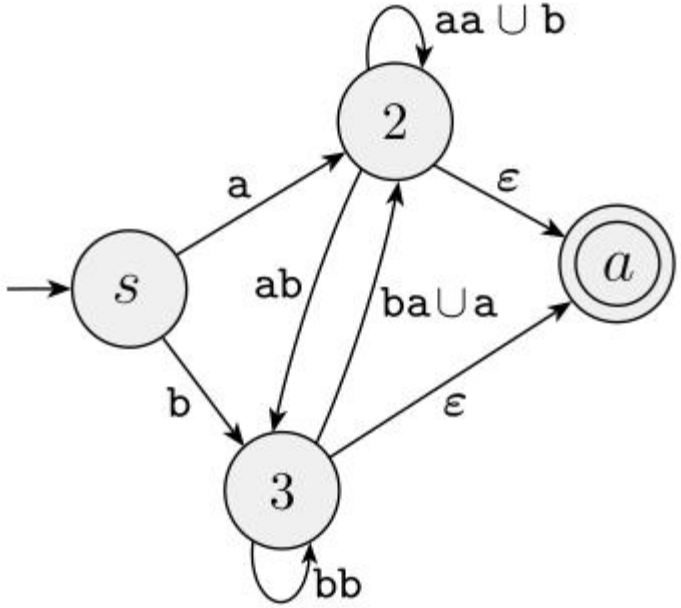
- **Example 1.68.** Converting a three-state DFA to an equivalent regular expression



(a)

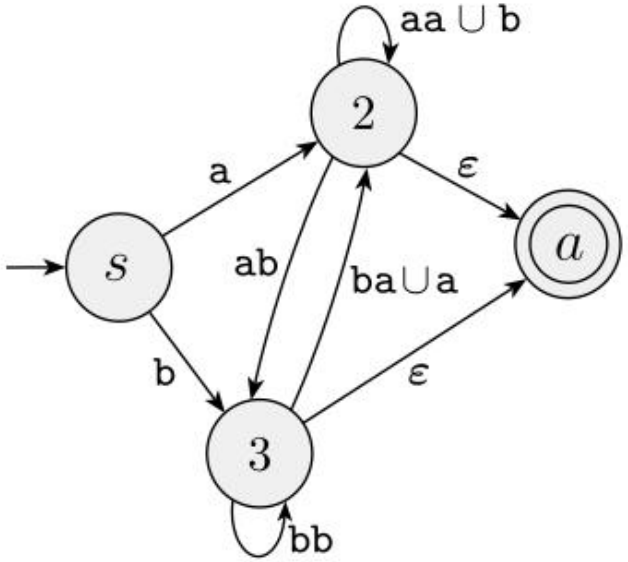


(b)

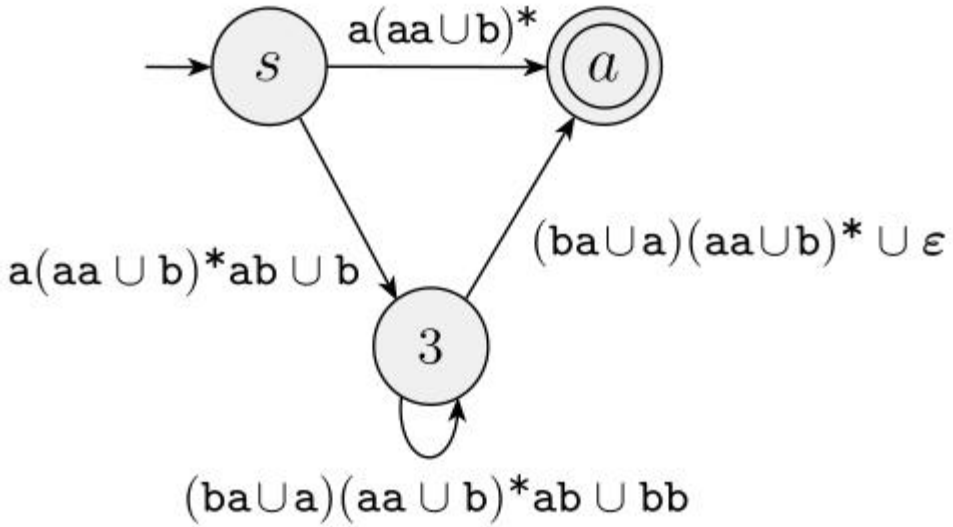


(c)

3. Regular Expressions
Equivalence With Finite Automata
Convert a DFA into an Equivalent Regular Expression



(c)



(d)

(a(aa∪b)*ab∪b)((ba∪a)(aa∪b)*ab∪bb)*((ba∪a)(aa∪b)* ∪ε)∪a(aa∪b)*

(e)

4. Nonregular Languages

- Let's take the language $B = \{0^n1^n \mid n \geq 0\}$. B is not regular.
  - We discover that the machine seems to need to remember how many 0s have been seen so far as it reads the input.
  - Because the number of 0s isn't limited, the machine will have to keep track of an unlimited number of possibilities. But it cannot do so with any finite number of states.

4. Nonregular Languages
The Pumping Lemma for Regular Languages

- **Theorem 1.70**

  Pumping lemma If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

  - 1. for each $i \geq 0$, $xy^iz \in A$,
  - 2. $|y| > 0$, and
  - 3. $|xy| \leq p$.

- When s is divided into xyz, either x or z may be ε, but condition 2 says that
  $y \neq \varepsilon$.

4. Nonregular Languages
The Pumping Lemma for Regular Languages

- **Example 1.73.** Let B be the language $\{0^n1^n \mid n \geq 0\}$. We use the pumping lemma to prove that B is not regular. The proof is by contradiction.
  - Assume to the contrary that B is regular. Let p be the pumping length given by the pumping lemma. Choose s to be the string 0^p1^p. Because s is a member of B and s has length more than p, the pumping lemma guarantees that s can be split into three pieces, $s = xyz$, where for any $i \geq 0$ the string xy^iz is in B. We consider three cases to show that this result is impossible.
    - 1. The string y consists only of 0s. In this case, the string $xyyz$ has more 0s than 1s and so is not a member of B, violating condition 1 of the pumping lemma. This case is a contradiction.
    - 2. The string y consists only of 1s. This case also gives a contradiction.
    - 3. The string y consists of both 0s and 1s. In this case, the string $xyyz$ may have the same number of 0s and 1s, but they will be out of order with some 1s before 0s. Hence it is not a member of B, which is a contradiction.