Fall 2021 - MATH 1101 Discrete Structures
Lecture 14-15 Languages and Automata.

PART 1. Languages and Regular Expressions.

PART 2. Deterministic Finite State Automata (DFSA) and Language Recognition Problem.

PART 3. Nondeterministic Finite State Automata (NDFSA) and Kleene's Theorem. Pumping Lemma.

PART 4. Finite-State Machines with Outputs (Transducers) (*For Additional Reading*).

INTRODUCTION.

Computers can perform many tasks. Given a task, two questions arise. The first is: Can it be carried out using a computer? If the answer is "YES", the second question arises: how can the task be carried out? Models of computation are used to help answer these questions.

There are three types of structures used in models of computation, namely:

- Finite-state machines
- Grammars
- Turing machines.

1. Various types of finite-state machines are used in modeling. All finite-state machines have ***a set of states,*** including a starting state, ***an input alphabet,*** and ***a transition function*** that assigns a next state to every pair of a state and an input. The states of a finite-state machine give it limited memory capabilities. Some finite-state machines produce an output symbol for each transition; these machines (named also as transducers) can be used to model many kinds of machines, including vending machines, delay machines, binary adders, and language recognizers. There are also finite-state machines that have no output but do have final states. Such machines are extensively used in language recognition. The strings that are recognized are those that take the starting state to a final state.
2. Grammars are used to generate the words of a language and to determine whether a word is in a language. Formal languages, which are generated by grammars, provide models both for natural languages, such as English, and for programming languages, such as Python, C, Java etc. Grammars are extremely important in the construction and theory of compilers.
3. Turing machines can be used to recognize sets and compute number-theoretic functions. There is the Church–Turing thesis, which states that every effective computation can be carried out using a Turing machine. Turing machines can be used to study the difficulty of solving certain classes of problems.

In our course, we study various types of finite-state machines and some of their applications. The concepts of grammars and finite-state machines can be tied together. Grammars, as well as relationship between grammars and finite-state machines and details of Turing machines are studied in the frame of separate course, for example: Theory of Computation or Complexity Theory, or the similar titles. All of them are continuations of the current course.

PART 1. LANGUAGES AND REGULAR EXPRESSIONS.

In Part 1 we discuss *languages*, and *regular expressions*. These topics are closely related to each other. Languages use mostly the letters *a, b, . . .* to code data. Sometimes digits 0 and 1 are used.

Alphabets and words (or strings).

Definition 1. Given a nonempty set A of symbols a *word* or *string w* on the set A is a finite sequence of its elements.

For example, suppose A={*a, b, c*}. Then the following sequences are strings on A: u=*ababb,* v=*accbaaa*

When discussing strings on A, we frequently call A the *alphabet*, and its elements are called *letters*. We also abbreviate our notation and write a^2 for *aa,* a^3 for *aaa*, and so on. Thus, for the above words, u=$abab^2$ and v=ac^2bc^3.

The empty sequence of letters, denoted by λ (Greek letter lambda) or ε (Greek letter epsilon), is also considered to be a word on A, called the *empty word*. The set of all words on A is denoted by A^* (read: "A star"). The *length* of a word u, written |u| or l(u), is the number of elements in its sequence of letters. For the above words u and v, we have l(u)=5 and l(v)=7. Also, l(λ)=0, where λ is the empty word. ■

Remark: Unless otherwise stated, the alphabet A will be finite, the symbols u, v, w will be reserved for words on A, and the elements of A will come from the letters *a, b, c.*

Definition 2. Let A^* be the set of all words on an alphabet A. Define the binary operation $A^*\times A^*\rightarrow A^*$ which is called a concatenation as following:
- Let u and v be words from A*. The *concatenation* of u and v, written uv, is the new word from A* obtained by writing down the letters of u followed by the letters of v. For example, for the above words u and v, we have uv=ababbaccbaaa = $abab^2ac^2ba^3$. As with letters, for any word u, we define u^2=uu, u^3=uuu, and, in general, u^{n+1}=uu^n. ■

Clear, for any words u, v, w, the words (uv)w and u(vw) are identical, they simply consist of the letters of u, v, w written down one after the other. Also, adjoining the empty word before or after a word u does not change the word u. That is:

Theorem 1. The concatenation operation for words on an alphabet A is associative. The empty word λ is an identity element for the operation. ■

In general, the operation is not commutative, e.g., uv≠vu.

Languages

Definition 3. A *language* L over an alphabet A is a collection of words on A, that is, a language L is a subset of A*. ■

EXAMPLE 1. Let A={a, b}. The following are languages over A.
(a) L_1={a, ab, ab^2,...} (c) L_3={a^mb^m| m>0}
(b) L_2={a^mb^n| m>0, n>0} (d) L_4=b^mab^n| m≥0, n≥0}

One may verbally describe these languages as follows.

(a) L_1 consists of all words beginning with an a and followed by zero or more *b*'s.

(b) L_2 consists of all words beginning with one or more a's followed by one or more *b*'s.

(c) L_3 consists of all words beginning with one or more *a*'s and followed by the same number of *b*'s.

(d) L_4 consists of all words with exactly one a. ■

Operations over Languages

Definition 4. Suppose L and M are languages over an alphabet A. The **concatenation of L and M,** denoted by LM, is the language (=subset of A*) defined as follows:

$$LM=\{uv \mid u\in L, v\in M\}$$

That is, LM denotes the set of all words which come from the concatenation of a word from L with a word from M.

Powers of a language L are defined as follows: $L^0=\{\lambda\}$, $L^1=L$, $L^2=LL$, $L^{m+1}=L^mL$.

Given language L the *Kleene* closure of L is a new language L^* which is defined as:

$L^*=L^0\cup L^1\cup L^2\cup\ldots$ ■

It is clear, the concatenation of languages is associative since the concatenation of words is associative.

Suppose $L_1=\{a, b^2\}$, $L_2=\{a^2, ab, b^3\}$, then:

$L_1L_1=\{a^2, ab^2, b^2a, b^4\}$, $L_1L_2=\{a^3, a^2b, ab^3, b^2a^2, b^2ab, b^5\}$

Regular Expressions and Regular Languages (= Regular Sets)

Let A be a nonempty alphabet. This section defines a regular expression **r** over A and a regular language L(**r**) over A associated with the regular expression **r** (in some sources L(**r**) is also called as **regular set**).

The *regular expression* **r** and corresponding *regular language* (=*set*) L(**r**) are defined recursively (Definitions 5 and 6 below).

Definition 5. Each of the following is a *regular expression* over an alphabet *A*.	**Definition 6.** Each regular expression **r** generates (=represents) the *regular language* L(**r**) over A is as follows:
the symbol ∅ is a regular expression;	L(∅)=∅, that is, the symbol ∅ generates the empty language (set), that is, the language with no strings;
the symbol λ (or ε) is a regular expression;	L(λ)={λ}, that is, λ generates the language {λ}, which is the set containing the empty string;
the symbol **a** is a regular expression whenever **a**∈A;	L(**a**)={**a**}, that is letter **a**∈A generates the language {**a**} containing the string with one symbol **a**;
If $\mathbf{r}_1$ and $\mathbf{r}_2$ are regular expressions, then $(\mathbf{r}_1\cup\mathbf{r}_2)$ is regular expression	$L(\mathbf{r}_1\cup\mathbf{r}_2)=L(\mathbf{r}_1)\cup L(\mathbf{r}_2)$, that is, $\mathbf{r}_1\cup\mathbf{r}_2$ generates the union of the languages $L(\mathbf{r}_1)$ and $L(\mathbf{r}_2)$,.
If $\mathbf{r}_1$ and $\mathbf{r}_2$ are regular expressions, then the concatenation $(\mathbf{r}_1\mathbf{r}_2)$ is regular expression	$L(\mathbf{r}_1\mathbf{r}_2)=L(\mathbf{r}_1)L(\mathbf{r}_2)$, that is, the concatenation $\mathbf{r}_1\mathbf{r}_2$ of regular expressions generates the concatenation of the corresponding languages $L(\mathbf{r}_1)$ and $L(\mathbf{r}_2)$.
If **r** is a regular expression, then the Kleene closure **r*** is the regular expression	L(**r***)=(L(**r**))*, that is, the Kleene closure **r*** of regular expression **r** generates the Kleene closure of the language generated by **r**.

Remark: Parentheses will be omitted from regular expressions when possible. Since the concatenation of languages and the union of languages are associative, many of the parentheses may be omitted. Also, by adopting the convention that the star (*) operation takes precedence over concatenation, and concatenation takes precedence over the union (∪) other parentheses may be omitted.

Definition 7. Let L be a language over A. Then L is called **a *regular language*** over A if there

exists a regular expression **r** over A such that L=L(**r**). ■

EXAMPLE 2. Let A={a, b}. Each of the following is an expression r and its corresponding language L(r):

(a) Let r=a*. Then L(r) consists of all powers of a including the empty word λ.

(b) Let r=aa*. Then L(r) consists of all positive powers of a excluding the empty word λ.

(c) Let r=a∪b*. Then L(r) consists of a or any word in b, that is, L(r)={a, λ, b, b^2,⋯}.

(d) Let r=(a∪b)*. Note L(a∪b)={a}∪{b}=A; hence L(r)=A*, all words over A.

(e) Let r=(a∪b)*bb. Then L(r) consists of the concatenation of any word in A with bb, that is, all words ending in b^2.

(f) Let r=a∩b*. L(r) does not exist since r is not a regular expression. (Specifically, ∩ is not one of the symbols used for regular expressions.) ■

Next example solves the **inverse problem** for some cases:

Given language L find (if possible) a regular expression r such that L=L(r).

EXAMPLE 3. Consider the following languages over A={a, b}:

(a) $L_1=\{a^m b^n \mid m>0, n>0\}$; (b) $L_2=\{b^m a b^n \mid m>0, n>0\}$; (c) $L_3=\{a^m b^m \mid m>0\}$.

Find a regular expression **r** over A={a, b} such that L_i=L(**r**) for i=1, 2, 3.

Solution.

(a) L_1 consists of those words beginning with one or more *a*'s followed by one or more *b*'s. Thus, we can set r=aa*bb*. Note that r is not unique; for example, r=a*abb* is another solution.

(b) L_2 consists of all words which begin with one or more *b*'s followed by a single *a* which is then followed by one or more *b*'s, that is, all words with exactly one *a* which is neither the first nor the last letter. Hence r=bb*abb* is a solution. Really, based on rules from Definition 6 we have:

L(r)=L(bb*abb*)=L(b)L(b*)L(a)L(b)L(b*)={b}{b}*{a}{b}{b}*=

={b}{λ=b^0, b, b^2, }{a}{b}{λ=b^0, b, b^2, }={bab, bbab, babb, …}

(c) L_3 consists of all words beginning with one or more *a*'s followed by the same number of *b*'s. There exists no regular expression r such that L_3=L(r); that is, L_3 is not a regular language. The proof of this fact will be provided later. ■

EXERCISES SET 1.

1.1. Let A={a, b}. Describe verbally the following languages over A (which are subsets of A*):
(*a*) $L_1=\{(ab)^m \mid m>0\}$; (*b*) $L_2=\{a^r b a^s b a^t \mid r, s, t \geq 0\}$; (*c*) $L_3=\{a^2 b^m a^3 \mid m > 0\}$.

1.2. Let K={a, ab, a^2} and L={b^2, aba} be languages over A={a, b}. Find: (*a*) KL; (*b*) LL.

1.3. Consider the language L={ab, c}over A={a, b, c}. Find: (*a*) L^0; (*b*) L^3; (*c*) L^{-2};

1.4. Let A={a, b, c}. Find L* where: (*a*) L={b^2}; (*b*) L={a, b}; (*c*) L={a, b, c^3}.

1.5. Let A={a, b}. Describe the language L(r) where:
(a) r=abb*a; (b) r=b*ab*ab*; (c) r=a*∪b*; (d) r=ab*∩a*.

1.6. Let A={a, b, c} and let w=abc. Whether w belongs to L(r) where:
(a) r=a*∪(b∪c)*; (b) r=a*(b∪c)*.

1.7. Let A={a, b}. Find a regular expression r such that L(r) consists of all words w where:
(*a*) w begins with a^2 and ends with b^2; (*b*) w contains an even number of *a*'s.

1.8. Let L={a^2, ab} and K={a, ab, b^2}. Find: (a) LK; (b) KL; (c) L∪K; (d) K∩L.

1.9. Let L={a^2, ab}. Find: (a) L^0; (b) L^2; (c) L^3.

1.10. Let A={a, b, c}. Describe L* if: (a) L={a^2}; (b) L={a, b^2}; (c) L={a, b^2, c^3}.

1.11. Does $(L^2)^*=(L^*)^2$? If not, how are they related?

1.12. Let A={a, b, c}. Describe the language L(r) for each of the following regular expressions:

(a) r=ab*c; (b) r=(ab∪c)*; (c) r=ab∪c*.

1.13. Let A={a, b, c} and let w=ac. Whether w belongs to L(r) where:

(a) r=a*bc*; (b) r=a*b*c; (c) r=(ab∪c)*

1.14. Let A={a, b, c} and let w=abc. Whether w belongs to L(r) where:

(a) r=ab*(bc)*; (b) r=a*∪(b∪c)*; (c) r=a*b(bc∪c^2)*.

1.15. Let A={a, b}. Find a regular expression r such that L(r) consists of all words w where:

(a) w contains exactly three a's. (b) the number of *a*'s is divisible by 3.

Answers (SET 1).

A1.1. (*a*) L_1 consists of words w=ababab ⋯ab, that is, beginning with a, alternating with b, and ending with b.

(*b*) L_2 consists of all words with exactly two *b*'s.

(*c*) L_3 consists of all words beginning with a^2 and ending with a^3 with one or more *b*'s between them.

A1.2. (*a*) Concatenate words in K with words in L to obtain KL={ab^2, a^2ba, ab^3, ababa, a^2b^2, a^3ba}.

(*b*) Concatenate words in L with words in L to obtain LL={b^4, b^2aba, $abab^2$, aba^2ba}.

A1.3. (*a*) L^0={λ}, by definition.

(*b*) Form all three-word sequences from L to obtain:

L^3={abababb, ababc, abcab, abc^2, cabab, cabc, c^2ab, c^3}

(*c*) The negative power of a language is not defined.

A1.4. (*a*) L^* consists of all words b^n where n is even (including the empty word λ).

(*b*) L^* consists of words in a and b.

(*c*) L^* consists of all words from A with the property that the length of each maximal subword composed entirely of *c*'s is divisible by 3.

A1.5. (*a*) L(r) consists of all words beginning and ending in *a* and enclosing one or more *b*'s.

(*b*) L(r) consists of all words with exactly two *a*'s.

(*c*) L(r) consists of all words only in *a* or only in b, that is, L(r)={λ, a, a^2,⋯ , b, b^2,⋯}

(*d*) Here r is not a regular expression since Λ is not one of the symbols used in forming regular expressions.

A1.6. (*a*) No. Here L(r) consists of word in *a* or words in b and c.

(*b*) Yes, since a∈L(a)* and bc∈(b∪c)*.

A1.7. (*a*) $r=a^2(a\cup b)^*b^2$. (Note $(a\cup b)^*$ consists of all words on A.)

(*b*) Note $s=b^*ab^*ab^*$ consists of all words with exactly two *a*'s. Then $r=s^*=(b^*ab^*ab^*)^*$ is a seeking regular expression.

A1.8. (a) LK={a^3, a^3b, a^2b^2, aba, abab, ab^3}; (b) KL = {a^3, a^2b, aba^2, abab, b^2a^2, b^2ab};

(c) L∪K={a^2, ab, a, b^2}; (d) K∩L not defined.

A1.9. (a) L^0={λ}; (b) L^2={a^4, a^3b, aba^2, abab};

(c) L^3={a^6, a^5b, a^3ba^2, a^3bab, aba^4, aba^3b, $ababa^2$, abababb}.

A1.10. (a) L^*={a^n| n is even}. (b) All words w in a and b with only even powers of b.

(c) All words in a, b, c with each power of b even and each power of c a multiple of 3.

A1.11. No. $(L^2)^* \subseteq (L^*)^2$.

A1.12. a) $L(r) = \{ab^n c \mid n \geq 0\}$. (b) All words in x and c where x=ab. (c) $L(r)=ab \cup \{c^n \mid n \geq 0\}$.

A1.13. (a) No; (b) yes; (c) no.

A1.14. (a) Yes; (b) no; (c) no.

A1.15. a) $r=b^*ab^*ab^*ab^*$; (b) $r=(b^*ab^*ab^*ab^*)^*$.

EXERCISES. (SET 2).

2. Let A={**a, b, c**}. Find a regular expression **r** such that L(**r**) consists of all words w where:
   - 2.1. Each word $w \in L(\mathbf{r})$ contains the subword **abb** or **aac**.
   - 2.2. Each word $w \in L(\mathbf{r})$ has a length $l(w) \geq 3$ and do not start with the letter **c**.
   - 2.3. Each word $w \in L(\mathbf{r})$ has a length $l(w) \geq 2$ and the second letter is always **b**.
   - 2.4. If $\mathbf{c} \in w$ then **c** is preceded by an **a**.
   - 2.5. If $\mathbf{c} \in w$ then $\mathbf{acb} \in w$. Equivalent description: the letter **c** can appear in a word w only as part of the subword **acb** of the word w.
   - 2.6. Each word contains no more than two letters **c**.
   - 2.7. Given a word $w \in L(\mathbf{r})$, if $\mathbf{b} \in w$ then $\mathbf{cb} \notin w$. Equivalent description: the letter **c** cannot be a predecessor of the letter **b** in any word w.
   - 2.8. There is no $w \in L(\mathbf{r})$ with two or more consecutive **a**'s.
   - 2.9. If $w \in L(r)$ then neither **ba** nor **bb** is a subword of w.
   - 2.10. Given a word $w \in L(\mathbf{r})$, if $\mathbf{a} \in w$ then $\mathbf{abc} \in w$. Equivalent description: the letter **a** can appear in a word w only as part of the subword **abc** of the word w.
   - 2.11. Each word $w \in L(\mathbf{r})$ has a length $l(w) \geq 2$ and the letter before last one is exactly **c**.
   - 2.12. If $w \in L(\mathbf{r})$ then w does not contain the sequence **ab** as a subword.
   - 2.13. Let $w \in L(\mathbf{r})$ and $w=x_1x_2\ldots x_n$. If $x_k=\mathbf{c}$ then there exists an index $m<k$ such that $x_m=\mathbf{a}$. Equivalent description: if $\mathbf{c} \in w$ then there exists **a** which is "lefter" than **c**.
   - 2.14. Each word $w=x_1x_2\ldots x_n \in L(r)$ has a property: $x_k=\mathbf{a}$ if $k=2m+1$.
   - 2.15. If $w \in L(\mathbf{r})$ starts with **ab** then w ends with **c**.
   - 2.16. If $w=x_1x_2\ldots x_n \in L(\mathbf{r})$ and $x_k=\mathbf{a}$ then at least $x_{k-1}=\mathbf{a}$ or $x_{k+1}=\mathbf{a}$. Equivalent description: if a word contains the letter **a**, then **a** is written at least twice consecutively.
   - 2.17. Each word $w=x_1x_2\ldots x_n \in L(\mathbf{r})$ has a property: $x_k=\mathbf{b}$ if $k=2m$. Moreover L(**r**) does not contain empty word.
   - 2.18. If $\mathbf{a} \in w$ then **a** is followed by two consecutive **b**'s.
   - 2.19. If $\mathbf{c} \in w$ then **c** is preceded by **aa**.
   - 2.20. If $w \in L(\mathbf{r})$ then w does not contain the **cc**. Equivalent description: there is no word in L(**r**) containing **cc**.
   - 2.21. Let $w \in L(\mathbf{r})$. If $\mathbf{b} \in w$ then $\mathbf{a} \in w$.
   - 2.22. If $w \in L(\mathbf{r})$ then the length $l(w)=2k$. Equivalent description: All words in L(**r**) have even length.
   - 2.23. If $w \in L(\mathbf{r})$ then w contains three **c**'s and one of **c**'s is the last letter in w. Equivalent description: each word w in L(**r**) contains three **c**'s and one of **c**'s is the last letter in w.
   - 2.24. $L(\mathbf{r})=\{w \in A^* \mid l(w)=2k+1\}$. Equivalent description: L(**r**) consists of all words from A^* having odd length.
   - 2.25. Each word $w \in L(\mathbf{r})$ contains at least three **a**'s.
   - 2.26. There is no word in L(**r**) containing **cb**. Equivalent description: L(**r**) consists of all words w which do not contain the subword **cb**.
   - 2.27. Let $w \in L(\mathbf{r})$. If $\mathbf{a} \in w$ then w does not contain **b** righter than **a**.

2.28. Each word $w=x_1x_2\ldots x_n \in L(\mathbf{r})$ has a property: $x_k=\mathbf{c}$ if $k=2m$.
2.29. Each word $w \in L(\mathbf{r})$ contains at least four **b**'s.
2.30. If $w \in L(\mathbf{r})$ then w starts and end with the same letter.

PART 2. DETERMINISTIC FINITE STATE AUTOMATA

Definition 8. A *deterministic finite state automaton* (DFSA) or, simply, an *automaton* M, is a structure which consists of five parts:

(1) A finite set (=alphabet) A of inputs. In many examples, instead of A the symbol I is used (first letter of the word "input").

(2) A finite set S of states (or internal states).

(3) A subset $Y \subset S$ (called as accepting, or "yes", or final states).

(4) An initial state $s_0 \in S$.

(5) A next-state function $F: S \times A \to S$ (single-valued function of two variables). ■

Such an automaton M is denoted by $M=(A, S, Y, s_0, F)$ when we want to indicate its five parts. (The plural form of the word automaton is "automata".)

The finite-state automaton defined above is also called a **deterministic finite state automaton with no outputs (briefly, DFSA with no outputs)**, *because for each pair (state, input symbol) the next-state function returns only one state (we move from one state to another state using transition on an input symbol and do not produce any other output).*

Some texts define the next-state function F in (5) by means of a collection of functions $f_a: S \to S$, one for each $a \in A$. Setting $F(s, a)=f_a(s)$ shows that both definitions are equivalent.

We can represent finite-state automaton using one of two forms:

- state table which is another form of next-state function,
- state diagram (below, next section).

EXAMPLE 4. The following defines an automaton M with two input symbols and three states:

(1) $A=\{a, b\}$, input symbols.

(2) $S=\{s_0, s_1, s_2\}$, internal states.

(3) $Y=\{s_0, s_1\}$, "yes" states.

(4) s_0, initial state.

(5) Next-state function $F: S \times A \to S$ defined explicitly by the table 1 in Figure 1. ■

F	a	b
s_0	s_0	s_1
s_1	s_0	s_2
s_2	s_2	s_2

Figure 1. Next State Function

State Diagram of an Automaton M.

The state diagram D=D(M) of an automaton M is a labeled directed graph as follows.

(1) The vertices of D(M) are the states in S and an accepting state is denoted as double circled.

(2) There is an arrow in D(M) from state s_j to state s_k labeled by an input *a* if $F(s_j, a)=s_k$ or, equivalently, if $f_a(s_j)=s_k$. In this case we also say that there is a transition on input *a* from s_j to s_k.

(3) The initial state s_0 is indicated by means of a special arrow which terminates at s_0 but has no initial vertex.

For each vertex s_j and each letter *a* in the alphabet A, there will be an arrow leaving s_j which is labeled by a; hence the outdegree of each vertex is equal to number of elements in A. For notational convenience, we label a single arrow by all the inputs which cause the same change of state rather than having an arrow for each such input.

The state diagram D=D(M) of the automaton M in Example 4 appears in Figure 2. Note that both *a* and *b* label the arrow from s_2 to s_2 since $F(s_2, a)=s_2$ and $F(s_2, b)=s_2$. Note also that the outdegree of each vertex is 2, the number of elements in A.

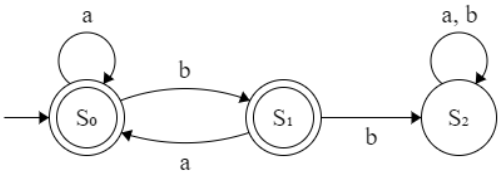


Figure 2. The state diagram D=D(M) of the automaton M in Example 4

Language L(M) Determined by an Automaton M

Each automaton **M** with alphabet A defines a language over A, denoted by L(M) as follows.

Definition 9. Let $w=a_1a_2 \dots a_m$ be *a* word on A. Then w determines the following path in the state diagram graph D(**M**) where s_0 is the initial state and $F(s_{i-1}, a_j)=s_i$ for $i\geq 1$:

$$P=(s_0, a_1, s_1, a_2, s_2, \cdots, a_m, s_m)$$

We say that **M *recognizes* the word w** if the final state s_m is an accepting state in Y. The set of all words, recognizable by M, is called **a language generated by automaton M (=language accepting by M, =language recognizable by M).** Denotation: **L(M).** Two finite-state automata are called ***equivalent*** if they recognize the same language. ■

EXAMPLE 5. Determine whether the automaton M in Figure 2 accepts the words:

$$w_1=ababba; \quad w_2=baab; \quad w_3=\lambda \text{ (the empty word).}$$

Solution. Use Figure 2 and the words w_1 and w_2 to obtain the following respective paths:

$$P_1=s_0 \xrightarrow{a} s_0 \xrightarrow{b} s_1 \xrightarrow{a} s_0 \xrightarrow{b} s_1 \xrightarrow{b} s_2 \xrightarrow{a} s_2 \quad \text{and} \quad P_2=s_0 \xrightarrow{b} s_1 \xrightarrow{a} s_0 \xrightarrow{a} s_0 \xrightarrow{b} s_1$$

The final state in P_1 is s_2 which is not in Y; hence w_1 is not accepted by M. On the other hand, the final state in P_2 is s_1 which is in Y; hence w_2 is accepted by M. The final state determined by w_3 is the initial state s_0 since $w_3=\lambda$ is the empty word. Thus, w_3 is accepted by M since $s_0 \in Y$. ■

Important Observation.

The natural question that arises immediately after Definition 9 is the following:

Problem 1. Given an DFSA **M,** construct a language L which is accepted by an automaton **M**, that is, L=L(M).

Problem 2 (inverse problem). Given a language L, construct a DFSA **M** that accepts L, that is, L(M)=L.

In the next part of the notes we offer the conditions (Kleene's theorem) which guarantee both:

a) the existence of solutions to the problems 1 and 2;
b) the method for solving problems 1 and 2.

Sometimes, it is possible to solve problems 1 and 2 based on an appropriate analysis without Kleene's theorem. Below are examples and some techniques for solving both problems (without Kleene's theorem).

The solution to the **Problem 1** in most cases comes down to building a regular expression (if possible).

The solution to the **Problem 2 (inverse problem)** is usually divided into 4 stages:

- language analysis.

- the construction of the so-called skeleton of an automaton, that is, the construction of a set of states and transitions that provide acceptance of simple basic strings of the language on which other longer words are built up.

- addition to the automaton new transitions that will ensure the reception of "long words".

- addition to the automaton new states (as a rule, this is one state - "dead state" = "black hole" = "recycle bin") and transitions that will complete the structure of DFSA (finalizes the next-state table); usually such a process implements the words which are rejecting by the state-diagram of the automaton.

EXAMPLE 6. (Problem 1). Describe the language L(M) of the automaton M in Figure 2.

***Solution.* L(M) will consist of all words *w* on A which do not have two successive *b*'s.**

This comes from the following facts:

(1) We can enter the state s_2 if and only if there are two successive *b*'s.

(2) We can never leave s_2

(3) The state s_2 is the only rejecting (nonaccepting) state. ■

EXAMPLE 7. (Problem 1). Describe the languages over the alphabet A={0, 1} recognized by the finite-state automata M_1, M_2, and M_3 in Figure 3 below.

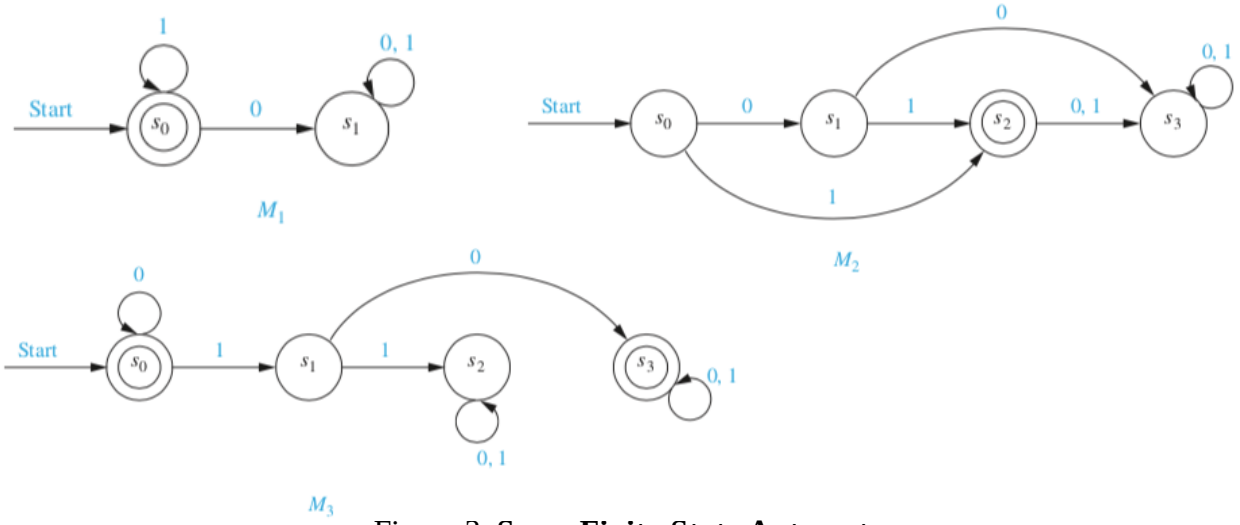


Figure 3. **Some Finite-State Automata.**

Solution: The only final state of M_1 is s_0. The strings (=words) that take s_0 to itself are those consisting of zero or more consecutive 1s. Hence, $L(M_1)=\{1^n | n=0, 1, 2, ...\}$.

The only final state of M_2 is s_2. The only words that take s_0 to s_2 are 1 and 01. Hence, $L(M_2)=\{1,01\}$.

The final states of M_3 are s_0 and s_3. The only words that take s_0 to itself are λ, 0, 00, 000, ..., that is, any word of zero or more consecutive 0s. The only words that take s_0 to s_3 are a string of zero or more consecutive 0s, followed by 10, followed by any word. Hence, $L(M_3)=\{0^m, 0^n10x \mid m, n \in \{0, 1\ 2, ..., \text{and x is any string}\}$. ■

Inverse problem: How to construct an automaton M that recognizes a given language L?

Below we suggest a technique that can be used to construct finite-state automata that recognize a language. (There is a general powerful result (Kleene's theorem): FSA M that recognizes a given language L exists iff L is regular. We will discuss this result later at Part 3).

FSA DESIGN. (Problem 2). We can often construct a finite-state automaton that recognizes a given language by carefully adding states and transitions and determining which of these states should be final states. When appropriate we include states that can keep track of some of the properties of the input string, providing the finite-state automaton with limited memory. Examples 8-10 illustrate some of the techniques that can be used to construct finite-state automata that recognize certain types of languages.

EXAMPLE 8. (Problem 2). Let A={a, b}. Construct an automaton M which will precisely accept those words over A which end in two *b*'s. [corresponding Language L=L(r) is regular because is generated by the regular expression r=(a∪b)*bb. Really, L(r)=(L(a∪b))*L(b)L(b)]=x{b}{b}, x is any word (string) in a and b.

Solution: Since b^2 is accepted, but not λ or b, we need three states, s_0, the initial state, and s_1 and s_2 with an arrow labeled b going from s_0 to s_1 and one from s_1 to s_2. Also, s_2 is an accepting state, but not s_0 nor s_1. This gives the graph in Figure 4(a). On the other hand, if there is an *a*, then we want to go back to s_0, and if we are in s_2 and there is ab, then we want to stay in s_2 These additional conditions give the required M which is shown in Figure 4(b).

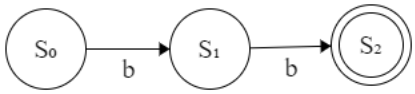


Figure 4 (a).

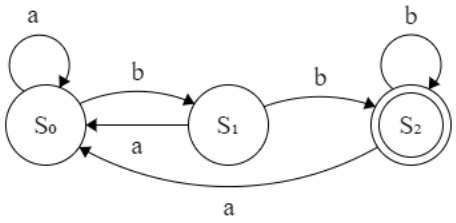


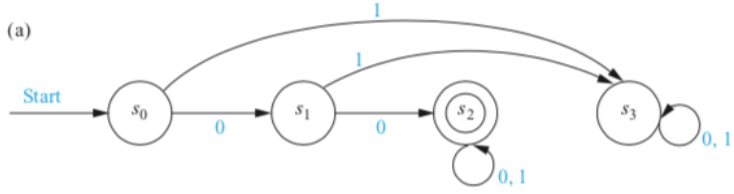
Figure 4 (b)

EXAMPLE 9. (Problem 2). Construct deterministic finite-state automata that recognize each of these languages.

a. the set of bit strings that begin with two 0s
b. the set of bit strings that contain two consecutive 0s
c. the set of bit strings that do not contain two consecutive 0s
d. the set of bit strings that end with two 0s
e. the set of bit strings that contain at least two 0s

Solution: **(a).** Our goal is to construct a DFSA that recognizes the set of words that begin with two 0s, that is, DFSA which recognizes the language L={00**x** | **x** – arbitrary word in 0 and 1, including λ}. Therefore, acceptable string with minimal length (=2) is 00. It is a main observation to start to build up a required automaton. Besides the start state s_0, we include a nonfinal (nonacceptable) state s_1; we move to s_1 from s_0 if the first bit is a 0. Next, we add a final state s_2, which we move to from s_1 if the second bit is a 0. When we have reached s_2 we know that the first two input bits are both 0s, so we stay in the state s_2 no matter what the succeeding bits (if any) are. That is, on both bits 0 an1 we must move from s_2 to itself. This case corresponds to the symbol **x** in description of the language L. All other moves from s_0 and s_1 on bit 1 must be nonacceptable. To implement that we organize a special nonfinal state s_3 which we call as a "black hole or recycle bin", and move to s_3 from s_0 if the first bit is of a string is a 1 and from s_1 if the second bit of a string is a 1 and stay there forever (black hole). This is done by transitions from s3 to itself (a loop in s3) on edges 0 and 1. It is easy to check that the FSA in Figure 5(a) recognizes the set of words that begin with two 0s.

Figure 5(a)



(b). Our goal is to construct a DFSA that recognizes the set of bit strings that contain two consecutive 0s. Besides the start state s_0, we include a nonfinal state s_1, which tells us that the last input bit seen is a 0, but either the bit before it was a 1, or this bit was the initial bit of the string. We include a final state s_2 that we move to from s_1 when the next input bit after a 0 is also a 0. If a 1 follows a 0 in the string (before we encounter two consecutive 0s), we return to s_0 and begin looking for consecutive 0s all over again. The reader should verify that the FSA in Figure 5(b) recognizes the set of bit strings that contain two consecutive 0s.

(c). Our goal is to construct a DFSA that recognizes the set of words that do not contain two consecutive 0s. Besides the start state s_0, which should be a final state, we include a final state s_1, which we move to from s_0 when 0 is the first input bit. When an input bit is a 1, we return to, or stay in, state s_0. We add a state s_2, which we move to from s_1 when the input bit is a 0. Reaching s_2 tells us that we have seen two consecutive 0s as input bits. We stay in state s_2 once we have reached it; this state is not final. The reader should verify that the FSA in Figure 5(c) recognizes the set of words that do not contain two consecutive 0s.

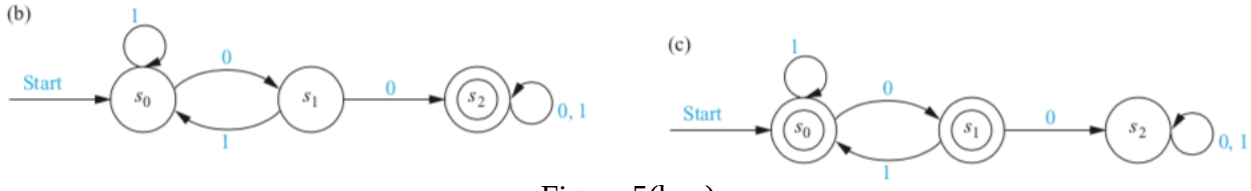


Figure 5(b, c)

(d). Our goal is to construct a DFSA that recognizes the set of bit strings that end with two 0s. Besides the start state s_0, we include a nonfinal state s_1, which we move to if the first bit is 0. We include a final state s_2, which we move to from s_1 if the next input bit after a 0 is also a 0. If an input of 0 follows a previous 0, we stay in state s_2 because the last two input bits are still 0s. Once we are in state s_2, an input bit of 1 sends us back to s_0, and we begin looking for consecutive 0s all over again. We also return to s_0 if the next input is a 1 when we are in state s_1. The reader should verify that the DFSA in Figure 5(d) recognizes the set of words that end with two 0s.

(e). Our goal is to construct a DFSA that recognizes the set of words that contain at least two 0s. Besides the start state, we include a state s_1, which is not final; we stay in s_0 until an input bit is a 0 and we move to s_1 when we encounter the first 0 bit in the input. We add a final state s_2, which we move to from s_1 once we encounter a second 0 bit. Whenever we encounter a 1 as input, we stay in the current state. Once we have reached s_2, we remain there. Here, s_1 and s_2 are used to tell us that we have already seen one or two 0s in the input string so far, respectively. The reader should verify that the DFSA in Figure 5(e) recognizes the set of words that contain two 0s.

00 – acceptable string with minimal length. L=L(M)={1^n01^m0x | m, n≥0, x is any word in 0 and 1.■

Figure 5(d, e)

EXAMPLE 10. (Problem 2). Construct a DFSA that recognizes the set of bit strings that contain an *odd number of 1s* and that *end with at least two consecutive 0s*.

Solution: We can build a DFSA that recognizes the specified set by including states that keep track of both the parity of the number of 1 bits and whether we have seen no, one, or at least two 0s at the end of the input string.

The start state s_0 can be used to tell us that the input read so far contains an even number of 1s and ends with no 0s (that is, is empty or ends with a 1). Besides the start state, we include five more states. We move to states s_1, s_2, s_3, s_4, and s_5, respectively, when the input string read so far contains

an even number of 1s and ends with one 0; when it contains an even number of 1s and ends with at least two 0s; when it contains an odd number of 1s and ends with no 0s; when it contains an odd number of 1s and ends with one 0; and when it contains an odd number of 1s and ends with two 0s. The state s_5 is a final state.

The reader should verify that the FSA in Figure 6 recognizes the set of bit strings that contain an odd number of 1s and end with at least two consecutive 0s. ■

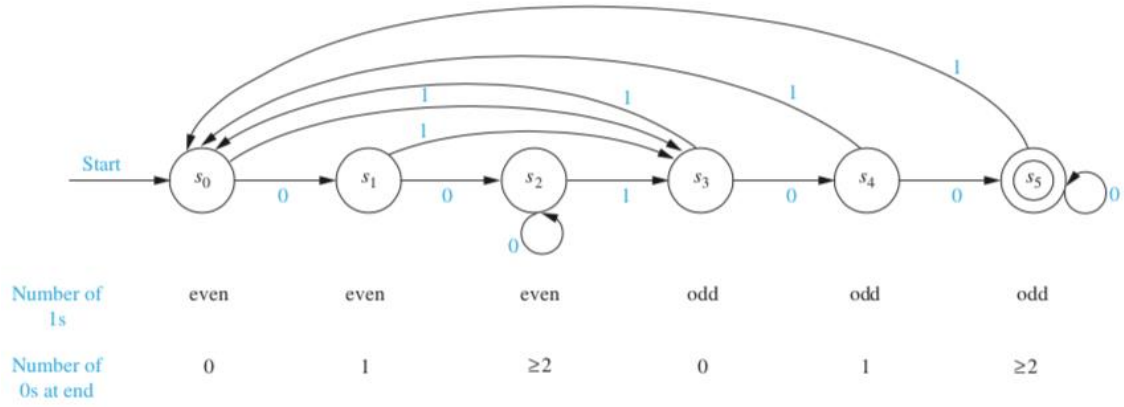


FIGURE 6. A DFSA Recognizing the Set of Bit Strings Containing an Odd Number of 1s and Ending with at Least Two 0s.

Equivalent FSA. In Definition 9 we specified the equivalences of two DFSA. Example 11 provides an example of two equivalent deterministic finite-state machines.

EXAMPLE 11. Show that the two finite-state automata M_0 and M_1 shown in Figure 7 are equivalent.

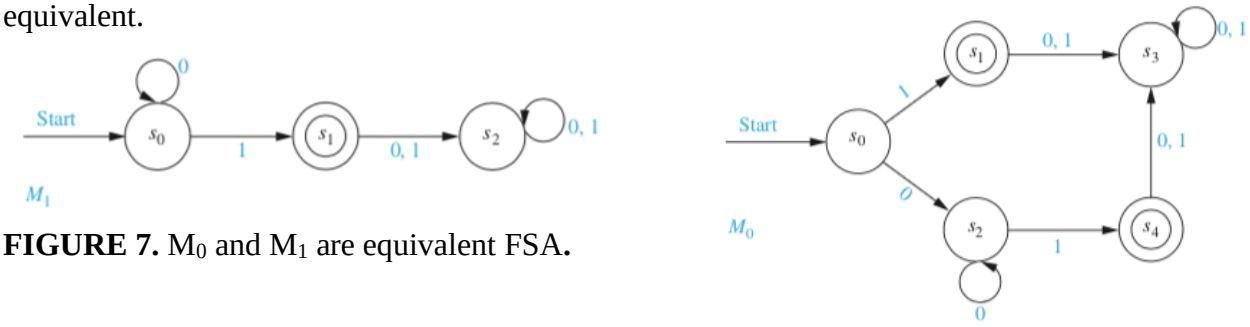


FIGURE 7. M_0 and M_1 are equivalent FSA**.**

Solution:

Below is a description of the languages recognized by M_1 (language L) and M_0 (language K)

$L=\{0^n1 \mid n\geq 0\}=\{1, 01, 001, 0001, \ldots\}$ – recognized by M_1

$K=\{1, 00^m1 \mid m\geq 0\}=\{1, 01, 001, 0001, \ldots\}$ – recognized by M_0

$L=K$

It follows that $L(M_1)=L$ is the same as $L(M_0)=K$. We conclude that M_0 and M_1 are equivalent.

Note that the finite-state machine M_1 only has three states. No finite state machine with fewer than three states can be used to recognize the set of all strings of zero or more 0 bits followed by a 1. ■

As Example 11 shows, a DFSA may have more states than one equivalent to it. In fact, algorithms used to construct finite-state automata to recognize certain languages may have many more states than necessary. Using unnecessarily large finite-state machines to recognize languages can make both hardware and software applications inefficient and costly. This problem arises when finite-state automata are used in compilers, which translate computer programs to a language a computer can understand (object code).

There exists a procedure that constructs a FSA with the fewest states possible among all FSA equivalent to a given FSA. This procedure is known as **machine minimization**. The minimization procedure reduces the number of states by replacing states with equivalence classes of

states with respect to an equivalence relation in which two states are equivalent if every input string either sends both states to a final state or sends both to a state that is not final. Before the minimization procedure begins, all states that cannot be reached from the start state using any input string are first removed; removing these does not change the language recognized.

EXERCISES. SET 3.

3.1. Describe the words *w* in the language L accepted by the automaton M in Figure 8 (a, b).

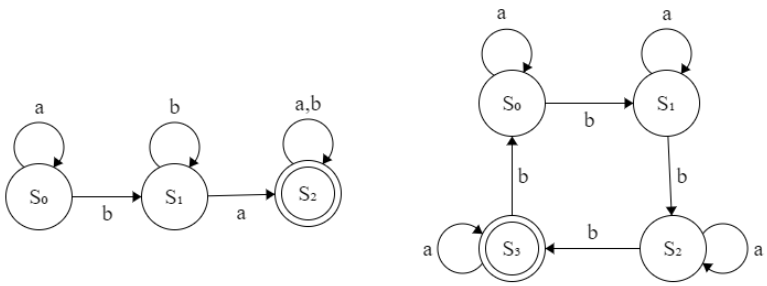


Figure 8 (a, b)

3.2. Find the language L(M) accepted by the automaton M in Figure 9.

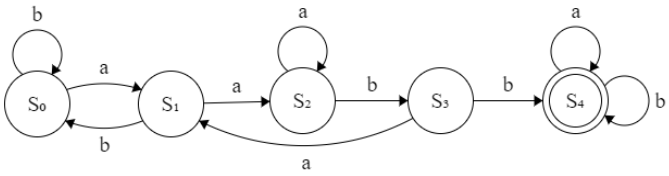


Figure 9.

3.3. Let M be the automaton with the following input set A, state set S with initial state s_0, and accepting ("yes") state set Y:

$$A=\{a, b\},\ S=\{s_0, s_1, s_2\},\ Y=\{s_1\}$$

Suppose next state function F of M is given by the table 2

Table 2.

F	a	b
s_0	s_0	s_1
s_1	s_1	s_2
s_2	s_2	s_2

(*a*) Draw the state diagram D=D(M) of M.

(*b*) Describe the language L=L(M) accepted by M.

3.4. Let A={a, b}. Construct an automaton M which will accept precisely those words from A which have an even number of *a*'s. For example, *aababbab*, *aa*, *bbb*, *ababaa* will be accepted by M, but *ababa*, *aaa*, *bbabb* will be rejected by M.

3.5. Let A={a, b}. Construct an automaton M which will accept those words from A which begin with an a followed by (zero or more) *b*'s.

3.6. Suppose L is a language over A which is accepted by the automaton M = (A, S, Y, s_0, F). Find an automaton N which accepts L′=A*-L (complement of L), that is, those words from A which do not belong to L.

3.7. Let A={a, b}. Construct an automaton M such that L(M) consist of those words w where:

(a) the number of *b*'s is divisible by 3. (b) w begins with a and ends in b.

3.8. Let A={a, b}. Construct an automaton M which accepts the language:

(a) $L(M)=\{b^r a b^s \mid r>0, s>0\}$; (b) $L(M)=\{a^r b^s \mid r>0, s>0\}$.

3.9. Let A={0, 1}. Determine whether each of these strings is recognized by the DFSA in Figure 10:

a) 111 **b)** 0011 **c)** 1010111 **d)** 011011011

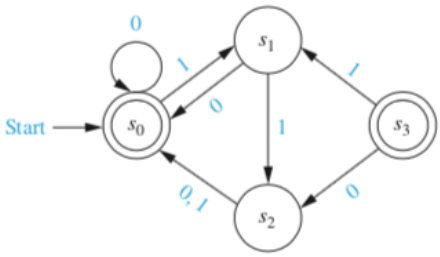


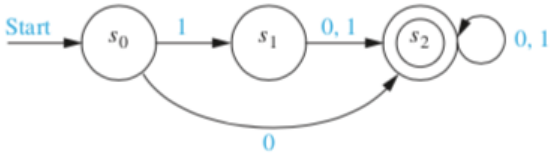
Figure 10

3.10. Determine whether all the strings in each of these sets are recognized by the DFSA in Fig. 10.

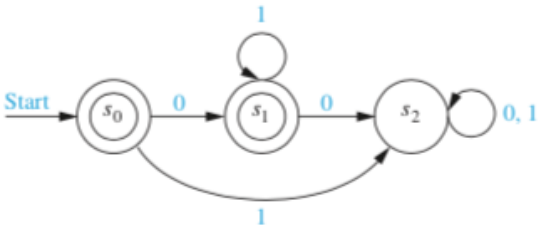
a) $\{0\}^*$ **b)** $\{0\}\{0\}^*$ **c)** $\{1\}\{0\}^*$.

d) $\{01\}^*$ **e)** $\{0\}^*\{1\}^*$ **f)** $\{1\}\{0, 1\}^*$.

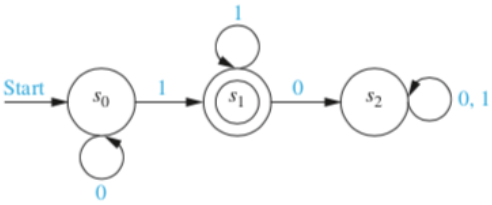
3.11. Find the language recognized by the given DFSA.



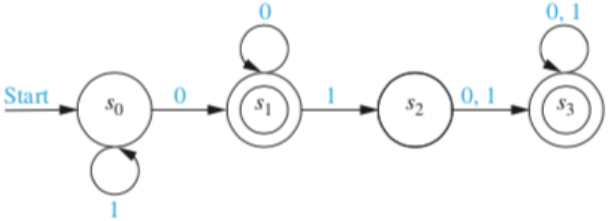
3.12. Find the language recognized by the given DFSA.



3.13. Find the language recognized by the given DFSA.



3.14. Find the language recognized by the given DFSA.



3.15. Show that there is no finite-state automaton with two states that recognizes the set of all bit strings that have one or more 1 bits and end with a 0.

3.16. Show that there is no finite-state automaton with three states that recognizes the set of bit strings containing an even number of 1s and an even number of 0s.

3.17. Construct a deterministic finite-state automaton that recognizes the set of all bit strings beginning with 01.

3.18. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that contain the string 101.

3.19. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that contain exactly three 0s.

3.20. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that contain three consecutive 1s.

3.21. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that contain an odd number of 0s.

3.22. Construct a finite-state automaton that recognizes the set of bit strings consisting of a 0 followed by a string with an odd number of 1s.

3.23. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that begin and end with 11.

Answers (SET 3).

A3.1. Figure 8(a): The system can reach the accepting state s_2 only when there exists an *a* in w which follows a *b*.

Figure 8(b): Each a in w does not change the state of the system, whereas each b in w changes the state from R_i, to s_{i+1} (modulo 4). Thus, w is accepted by M if the number n of *b*'s in w is congruent to 3 modulo 4, that is, where n = 3, 7, 11, ...

A3.2. L(M) (Figure 9) consists of all words w which contain *aabb* as a subword.

A3.3. (*a*) The state diagram D appears in Figure 11. The vertices of D are the states, and a double circle indicates an accepting state. If F (s_j, x)=s_k, then there is a directed edge from s_j to s_k labeled by the input symbol x. Also, there is a special arrow which terminates at the initial state s_0.

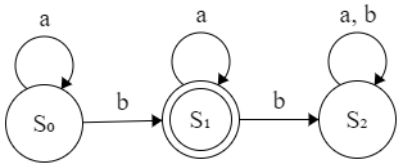
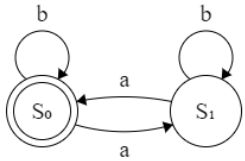


Figure 11.

(*b*) L(M) consists of all words w with exactly one b. Specifically, if an input word *w* has no *b*'s, then it terminates in s_0 and if w has two or more *b*'s then it terminates in s_2. Otherwise w terminates in s_1, which is the only accepting state.

A3.4. We need only two states, s_2 and s_1. We assume that M is in state s_0 or s_1 according as the number of *a*'s up to the given step is even or odd. (Thus, s_0 is an accepting state, but s_1 is a rejecting state.) Then only a will change the state. Also, s_0 is the initial state. The state diagram of M is shown in Figure 12.

Figure 12.



A3.5. The automaton M appears in Figure 13.

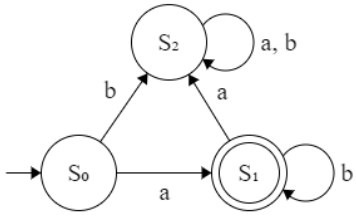


Figure 13.

A3.6. Simply interchange the accepting and rejecting states in M to obtain N. Then w will be accepted in the new machine N if and only if w is rejected in M, that is, if and only if w belongs to L′.

A3.7. See Figures 14 (a, b)

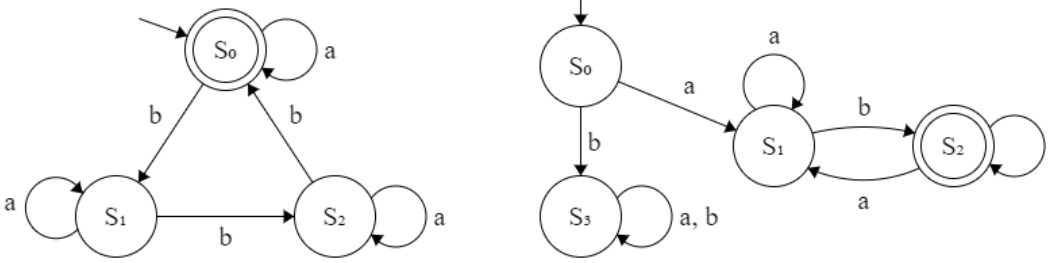


Figure 14 (a, b)

A3.8. See Figures 15 (a, b)

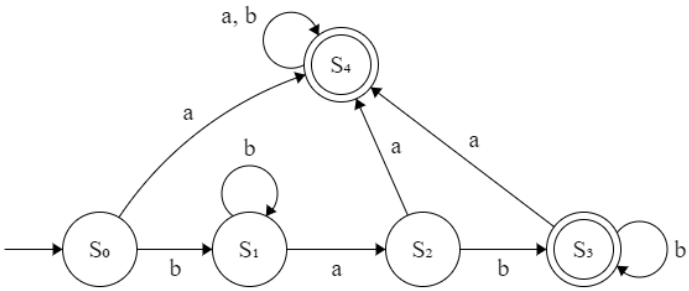


Figure 15 (a)

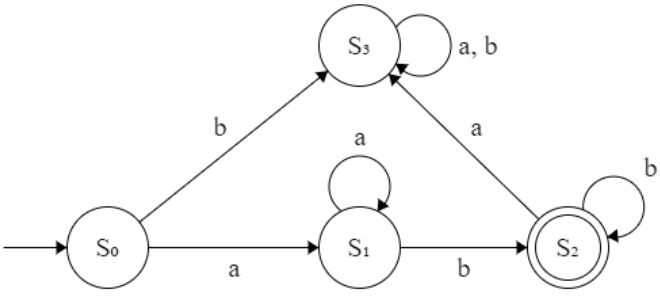


Figure 15 (b)

A3.9. **a)** Yes; **b)** No; **c)** Yes; **d)** No;

A3.10. **a)** Yes; **b)**Yes; **c)** No; **d)** No; **e)** No; **f)** No;

A3.11. $\{0, 10, 11\}\{0,1\}^*$

A3.12. $\{\lambda\}\cup\{01^n|\ n\geq 0\}$

A3.$\{0^m1^n|\ m\geq 0 \text{ and } n\geq 1\}$

A3.14. $\{0\}\{0\}^*\cup\{0\}\{0\}^*\{10\}\{0,1\}^*\cup\{0\}\{0\}^*\{11\}\{0,1\}^*$

A3.15. Suppose that such a machine exists, with start state s_0 and other state s_1. Because the empty string is not in the language, but some strings are accepted, we must have s_1 as the only final state, with at least one transition from s_0 to s_1. Because the string 0 is not in the language, the transition from s_0 on input 0 must be to itself, so the transition from s_0 on input 1 must be to s_1 But this contradicts the fact that 1 is not in the language.

A3.16. Suppose that such a machine M exists, with start state s_0 and other states s_1 and s_2. This must recognize only strings w which contain (2n) 1's, n>0 and (2m) 0's, m>0, (for instance, 1001110110, here 6 times 1 and 4 times 0). In that case s_0 cannot be an acceptable state because λ is not acceptable string. Consider two possible cases:

Case a). s_1 and s_2 both are final states. Then transition from s_0 on input 0 (or 1) generates a contradiction as M accepts a string 0 or 1 (odd numbers of 0s or 1s) which is impossible.

Case b). Only one of s_1 and s_2 is a final state. Let it will be s_1. Any direct transition from s_0 to s_1 by any input symbol 0 or 1 again generates contradiction. Assume that 1 provides the transition from s_0 to s_2. We cannot have a transition from s_2 to s_2 on 0 (loop on s_2 by edge 0) because other outgoing edge from s_2 must be on 1 (to s_0 or s_1) and we obtain a contradiction again (non-acceptable string 10^*1 if end point is s_0 with even number of 1s; or acceptable string 10^*1 if end point is s_1 with odd numbers of 0s). Therefore, we must have transition from s_2 to s_1 on 0. It means that we obtain acceptable path 10 which is a contradiction again.

A3.17. Let s_2 be the only final state. Put transitions from s_2 to itself on either input. Put a transition from the start state s_0 to s_1 on input 0, and a transition from s_1 to s_2 on input 1. Create state s_3, and have the other transitions from s_0 and s_1 (as well as both transitions from s_3) lead to s_3.

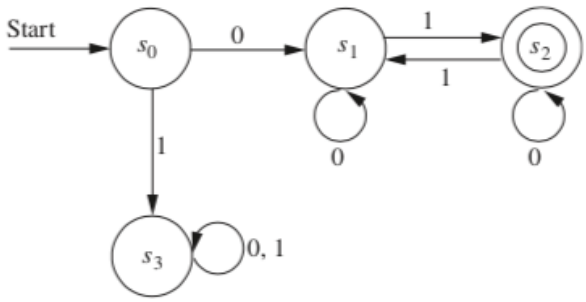
A3.18. Start state s_0, only final state s_3; transitions from s_0 to s_0 on 0, from s_0 to s_1 on 1, from s_1 to s_2 on 0, from s_1 to s_1 on 1, from s_2 to s_0 on 0, from s_2 to s_3 on 1, from s_3 to s_3 on 0, from s_3 to s_3 on 1.

A3.19. Have five states, with only s_3 final. For i=0, 1, 2, 3, transition from s_i to itself on input 1 and to s_{i+1} on input 0. Both transitions from s_4 are to itself.

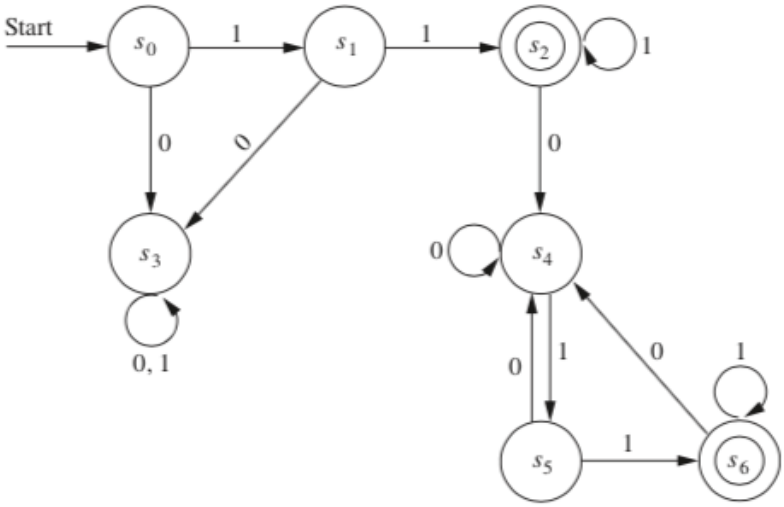
A3.20. Have four states, with only s_3 final. For i=0, 1, 2, transition from s_i to s_{i+1} on input 1 but back to s_0 on input 0. Both transitions from s_3 are to itself.

A3.21. Start state s_0, only final state s_1; transitions from s_0 to s_0 on 1, from s_0 to s_1 on 0, from s_1 to s_1 on 1; from s_1 to s_0 on 0.

A3.22.



A3.23.



PART 3. NONDETERMINISTIC FSA AND KLEENE'S THEOREM. PUMPING LEMMA.

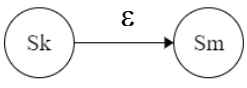
The finite-state automata discussed so far are **deterministic**, because for each pair of state and input value there is a unique next state given by the transition function. There is another important type of finite-state automaton in which there may be several possible next states for each pair of input value and state. Such machines are called **nondeterministic**. Nondeterministic finite-state automata are important in determining which languages can be recognized by a finite-state automaton.

Definition 10. A *nondeterministic finite-state automaton* (NDFSA) $M=(I, S, f, s_0, F)$ consists of an input alphabet I, a set S of states, a transition (next-state) function f that assigns a **set of states** to each pair of state and input (so that $f: S\times I\rightarrow P(S)$), a starting state s_0, and a subset F of S consisting of the final states. ■

Definition 10 says that an NDFSA (the same for DFSA) consumes an input symbol to move from one state to another states (state, in case of DFSA). We say also that an automaton M provides a transition from a state s_k to a state s_m **by consuming an input symbol, say *i*,** (or, **on input symbol, say *i***).

Important Note. In almost all kinds of NDFSA sometimes it is necessary to change a state spontaneously **without consuming an input symbol**. It can be done by using so-called ε (=λ) move (=transition), here ε (or λ) is empty string.

For example, in NDFSA below we change state from s_k to s_m due to ε - move without consuming an input symbol: ε (or λ) is empty string



The using ε-moves technique is very productive in building some new automata. We will demonstrate it in proof of Kleene's Theorem.

NDFSAs can be represented by state tables or state diagrams (similar to DFSA).

- When we use a state table, for each pair of state and input value we give **a list of possible next states**.
- In the state diagram, we include an edge from each state to all possible next states, labeling edges with the input or inputs that lead to this transition.

EXAMPLE 12. Find the state diagram for the NDFSA with the state table shown in Table 3. The final states are s_2 and s_3.

Solution: The state diagram for this automaton is shown in Figure 16. ■

Table 3		
State	f	
	Input	
	0	1
s_0	s_0, s_1	s_3
s_1	s_0	s_1, s_3
s_2		s_0, s_2
s_3	s_0, s_1, s_2	s_1

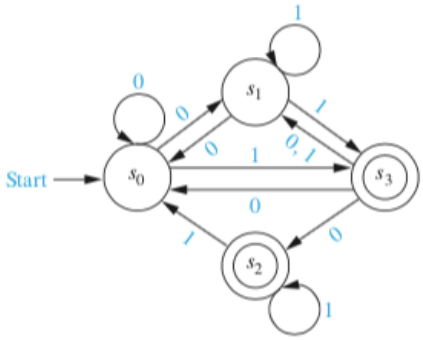


Figure 16. State Diagram for NDFSA given by **Table 3**.

EXAMPLE 13. Find the state table for the NDFSA with the state diagram shown in Figure 17.

Solution: The state table is given as Table 4.

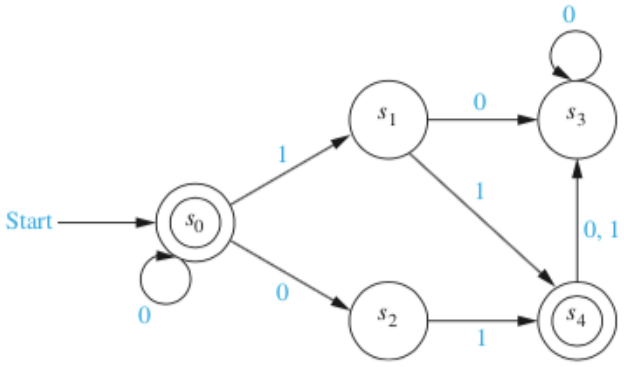


Table 4		
State	f	
	Input	
	0	1
s_0	s_0, s_2	s_1
s_1	s_3	s_4
s_2		s_4
s_3	s_3	
s_4	s_3	s_3

FIGURE 17. A Nondeterministic Finite-State Automaton**.**

EXAMPLE 14. Find the language recognized by the nondeterministic finite-state automaton shown in Figure 17.

Solution: Because s_0 is a final state, and there is a transition from s_0 to itself when 0 is the input, the machine recognizes all strings consisting of zero or more consecutive 0s. Furthermore, because s_4 is a final state, any string that has s_4 in the set of states that can be reached from s_0 with this input string is recognized. The only such strings are strings consisting of zero or more consecutive 0s followed by 01 or 11. Because s_0 and s_4 are the only final states, the language recognized by the machine is $\{0^n, 0^n01, 0^n11 \mid n \geq 0\}$. ■

One important fact is that a language recognized by a nondeterministic finite-state automaton is also recognized by a deterministic finite-state automaton.

Theorem 2. If the language L is recognized by a nondeterministic finite-state automaton M_0, then L is also recognized by a deterministic finite-state automaton M_1.

Proof. We describe how to construct the deterministic finite-state automaton M_1 that recognizes L from M_0, the nondeterministic finite-state automaton that recognizes this language. Each state in M_1 will be made up of a set of states in M_0. The start symbol of M_1 is **a set** $\{s_0\}$, which is the set containing the start state of M_0. The input set of M_1 is the same as the input set of M_0.

Given a state $\{s_{i1}, s_{i2}, ..., s_{ik}\}$ of M_1, the input symbol x takes this state to the union of the sets of

next states for the elements of this set, that is, the union of the sets $f(s_{i1}, x)$, $f(s_{i2}, x)$, ..., $f(s_{ik}, x)$.The states of M_1 are all the subsets of S, the set of states of M_0, that are obtained in this way starting with s_0. (There are as many as 2^n states in the deterministic machine, where n is the number of states in the nondeterministic machine, because all subsets may occur as states, including the empty set, although usually far fewer states occur.) The final states of M_1 are those sets that contain a final state of M_0.

Suppose that an input string is recognized by M_0. Then one of the states that can be reached from s_0 using this input string is a final state (it can be proved by induction). This means that in M_1, this input string leads from $\{s_0\}$ to a set of states of M_0 that contains a final state. This subset is a final state of M_1, so this string is also recognized by M_1. Also, an input string not recognized by M_0 does not lead to any final states in M_0. (The reader should provide the details that prove this statement.) Consequently, this input string does not lead from $\{s_0\}$ to a final state in M_1. ■

EXAMPLE 15.

1. Find a DFSA in "table form" that recognizes the same language as the NDFSA in Example 13.
2. Then build up state diagram for the DFSA created.

Solution:

Part 1. We build up equivalent DFSA in "table form" based on "table form" of NDFSA and algorithm of the Theorem 2 (s_0 and s_4 are final states).

To make the process clearer we repeat below Table 4 (NDFSA) from Example 13 and based on algorithm from Theorem 2 create a new Table 5 (DFSA). We show, step by step, how new states of DFSA are created.

Table 4 (NDFSA)

State	f	
	Input 0	1
s_0	s_0, s_2	s_1
s_1	s_3	s_4
s_2		s_4
s_3	s_3	
s_4	s_3	s_3

Table 5 (DFSA)

State	f		
	Input 0	1	New states produced (theorem 2)
$\{s_0\}$	$s_0 \cup s_2 = \{s_0, s_2\}$	**$\{s_1\}$**	$\{s_0, s_2\}$, $\{s_1\}$
$\{s_1\}$	$s_3 = \{s_3\}$	$s_4 = \{s_4\}$	$\{s_3\}$, $\{s_4\}$
$\{s_0, s_2\}$	$s_0 \cup s_2 \cup \varnothing = \{s_0, s_2\}$	$s_1 \cup s_4 = \{s_1, s_4\}$	$\{s_1, s_4\}$
$\{s_3\}$	$s_3 = \{s_3\}$	$\varnothing = \{\varnothing\}$	$\{\varnothing\}$
$\{s_4\}$	$s_3 = \{s_3\}$	$s_3 = \{s_3\}$	
$\{s_1, s_4\}$	$s_3 \cup s_3 = \{s_3\}$	$s_4 \cup s_3 = \{s_3, s_4\}$	$\{s_3, s_4\}$
$\{\varnothing\}$	$\{\varnothing\}$	$\{\varnothing\}$	
$\{s_3, s_4\}$	$s_3 \cup s_3 = \{s_3\}$	$\varnothing \cup s_3 = \{s_3\}$	

EXPLANATION how to create Table 5.

The states of DFSA in Table 5 **are SUBSETS** (according to the Theorem 2) **of the set of all states of the NDFSA** from Table 4**.**

The Rule to create rows of Table 5 is:

- In the Table 5, the next state of any state { , } (*see the most left column of the Table 5*) under an input symbol is the SUBSET consisting of the **next states of ALL elements** forming the state { , }. To determine these **next states** see Table 4**.**

-) The 1st row (in Table 5):

on input symbol 0, the set $\{s_0\}$ goes to $\{s_0, s_2\}$, because s_0 (in Table 4) has transitions to itself and to s_2;

on input symbol 1, the set $\{s_0\}$ goes to $\{s_1\}$, because s_0 (in Table 4) has transitions to s_1;

Therefore, in Table 5, starting from the start state $\{s_0\}$ we produced new states for the seeking DFSA (we use set notation): $\{s_0, s_2\}$, $\{s_1\}$.

At the next iterations we must create new states of DFSA which are images of states already created, namely, **$\{s_0, s_2\}$ and $\{s_1\}$).**

-) The 2nd row (in Table 5):

on input symbol 0, the set $\{s_0, s_2\}$ goes to $\{s_0, s_2\}$ itself (it means that we do not create new state), because:

s_0 (in Table 4) has transitions to s_0 itself and to s_2;

s_2 (in Table 4) has transition to "nothing", that is, $\varnothing$.

Therefore, $\{s_0, s_2\}$ goes to $s_0 \cup s_2 \cup \varnothing$, which is again $\{s_0, s_2\}$ – no new state**.**

on input symbol 1, we find the image of the set $\{s_0, s_2\}$ as following:

s_0 (in Table 4) has transitions to s_1;

s_2 (in Table 4) has transition to s_4;

Therefore, $\{s_0, s_2\}$ goes to $s_1 \cup s_4 = \{s_1, s_4\}$ - **new state.**

We continue row by row. All NEW SUBSETS that are obtained in this way are included in the DFSA (last column of the Table 5). Note that the empty set is one of the states of this machine, because it is the subset containing all the next states of $\{s_3\}$ on input of 1 (5th row of Table 5).

The start state is $\{s_0\}$, and **the set of final states are all those that include s_0 or s_4**, that is, **$\{s_0\}$, $\{s_0, s_2\}$, $\{s_4\}$, $\{s_1, s_4\}$, $\{s_3, s_4\}$**

Part 2. The state diagram for the DFSA created in Table 5 is:

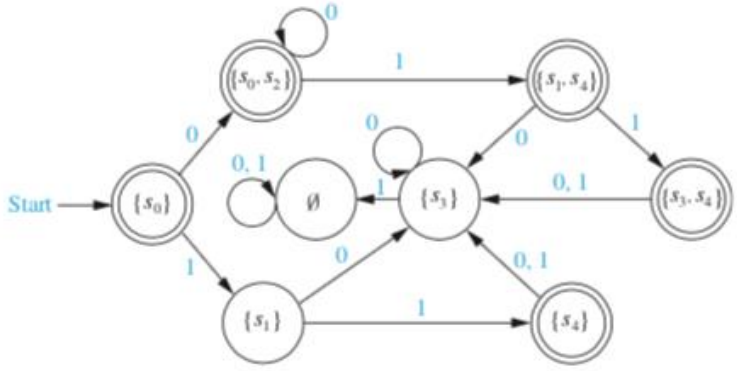


FIGURE 18. NDFSA which is equivalent to DFSA in Figure 17. ■

KLEENE'S THEOREM.

In 1956 Kleene proved that regular languages (=sets) are the sets that are recognized by a FSA. Consequently, this important result is called Kleene's theorem which is one of the central results in automata theory.

Theorem 3. Kleene's Theorem. A language is regular if and only if it is recognized by a FSA .

"Only If" part of Kleene's theorem.

Proof. We repeat the proof from KR textbook for the "Only If" part. Let L be a regular set. We must prove that there exist an automaton M such that L(M)=L.

Recall that a regular set is defined in terms of regular expressions, which are defined recursively. We can prove that every regular set is recognized by a FSA if we can do the following things.

1. Show that ∅ is recognized by an FSA.
2. Show that {λ} is recognized by an FSA.
3. Show that {a} is recognized by an FSA whenever a is a symbol in an alphabet I.
4. Show that AB is recognized by an FSA whenever both A and B are.
5. Show that A∪B is recognized by an FSA whenever both A and B are.
6. Show that A^* is recognized by an FSA whenever A is.

We now consider each of these tasks separately.

1. We show that ∅ is recognized by a nondeterministic finite-state automaton. To do this, all we need is an automaton with no final states. Such an automaton is shown in Figure 19(a).

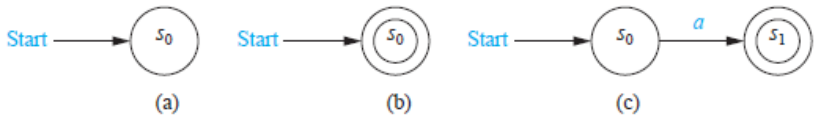


FIGURE 19. NDFSA That Recognize Some Basic Sets.

2. We show that {λ} is recognized by a finite-state automaton. To do this, all we need is an automaton that recognizes λ, the null string, but not any other string. This can be done by making the start state s_0 a final state and having no transitions, so that no other string takes s_0 to a final state. The nondeterministic automaton in Figure 19(b) shows such a machine.
3. We show that {*a*} is recognized by a NDSA. To do this, we can use a machine with a starting state s_0 and a final state s_1. We have a transition from s_0 to s_1 when the input is *a*, and no other transitions. The only string recognized by this machine is *a*. This machine is shown in Figure 19(c).

Below we demonstrate proof of the last three parts of Kleene's theorem.

Suppose that A is recognized by $M_A=(S_A, I, f_A, s_A, F_A)$ and B is recognized by $M_B=(S_B, I, f_B, s_B, F_B)$. Here, S_A – set of states of M_A, I – input set (alphabet), f_A – next-state function, s_A is the start state, and F_A – set of final states in M_A. The similar denotations are valid for machine M_B.

4. **Concatenation of Machines.** We show that AB can be recognized by a finite-state automaton if A and B are languages recognized by finite-state automata.

We begin by constructing a finite-state machine $M_{AB}=(S_{AB}, I, f_{AB}, s_{AB}, F_{AB})$ that recognizes AB, the concatenation of A and B. We build such a machine by combining the machines for A and B in series, so that:

- a string in A takes the combined machine from s_A, the start state of M_A, to s_B, the start state of M_B;
- a string in B should take the combined machine from s_B to a final state of the combined machine.

Consequently, we make the following construction.

4.1. $S_{AB}=S_A \cup S_B$. [Note that we can assume that S_A and S_B are disjoint.]

4.2. The starting state s_{AB} is the same as s_A.

4.3. The set of final states, F_{AB}, is the set of final states of M_B with s_{AB} included if and only if $\lambda \in A \cap B$.

4.4. The transitions in M_{AB} include all transitions in M_A and in M_B, as well as some new transitions: for every transition in M_A that leads to a final state, we form a transition in M_{AB} from the same state to s_B, on the same input. In this way, a string in A takes M_{AB} from s_{AB} to s_B, and then a string in B takes s_B to a final state of M_{AB}. Moreover, if $\lambda \in A$ then for every transition in B from s_B to any state s_i on input symbol *a* we form a transition in M_{AB} from s_{AB} to the same state

s_i on the same input symbol *a*.

Figure 20(a) contains an illustration of this construction.

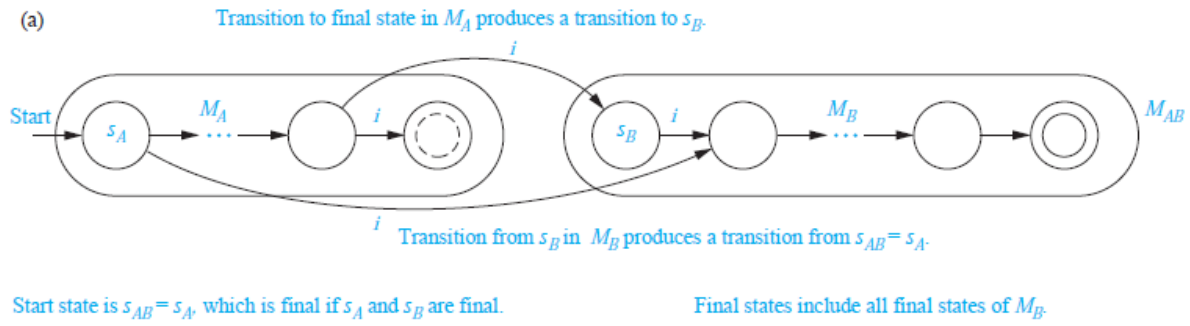


FIGURE 20 (a). Building Automata to Recognize Concatenations (From KR textbook)

5. **Union of machines.** We now construct a machine $M_{AUB}=(S_{AUB}, I, f_{AUB}, s_{AUB}, F_{AUB})$ that recognizes AUB. This automaton can be constructed by combining M_A and M_B in parallel, using a new start state that has the transitions that both s_A and s_B have.

5.1. $S_{AUB}=S_A \cup S_B \cup \{s_{AUB}\}$, where s_{AUB} is a new state that is the start state of M_{AUB}.

5.2. The starting state s_{AUB}, is an additional (external) state considered as the start state for M_{AUB}.

5.3. F_{AUB} be $F_A \cup F_B \cup \{s_{AUB}\}$ if $\lambda \in A \cup B$, and $F_A \cup F_B$ otherwise.

5.4. The transitions in M_{AUB} include all those in M_A and in M_B. Also, for each transition on input *i* from the state s_A to a state *s* we include a transition from s_{AUB} to *s* on input *i*, and for each transition from s_B to a state *s* on input *i* we include a transition from s_{AUB} to *s* on input *i*. In this way, a string in A leads from s_{AUB} to a final state in the new machine, and a string in B leads from s_{AUB} to a final state in the new machine.

Figure 20(b) illustrates the construction of M_{AUB}.

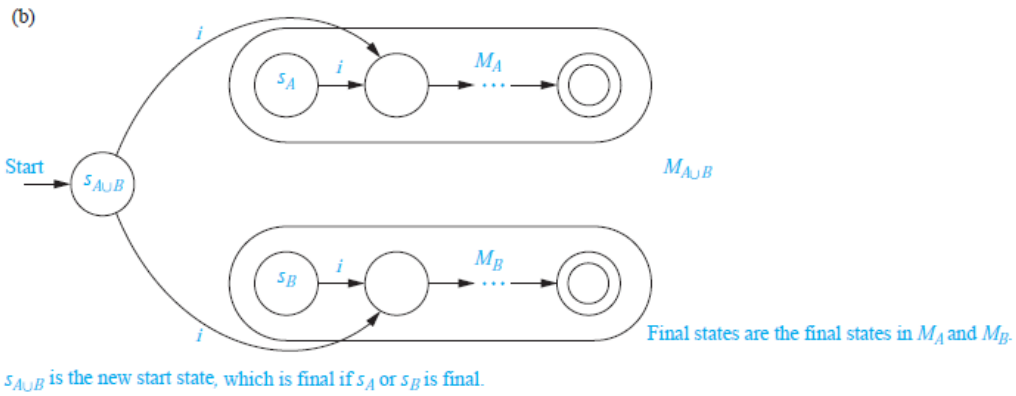


FIGURE 20 (b). Building Automata to Recognize Unions (From KR textbook)

6. **Kleene's closure of a machine.** We construct $M_{A^*}=(S_{A^*}, I, f_{A^*}, s_{A^*}, F_{A^*})$, a machine that recognizes A.

6.1. S_{A^*} include all states in S_A and one additional state s_{A^*}; $S_{A^*}=S_A \cup s_{A^*}$;

6.2. The starting state for the new machine is s_{A^*}.

6.3. The set of final states F_{A^*} includes all states in F_A as well as the start state s_{A^*}, because λ must be recognized.

6.4. To recognize concatenations of arbitrarily many strings from A, we include all the transitions in M_A, as well as transitions from s_{A^*} that match the transitions from s_A, and transitions from each final state that match the transitions from s_{A^*}. With this set of transitions, a string made up of concatenations of strings from A will take s_{A^*} to a final state when the first string in A has been read, returning to a final state when the second string in A has been read, and so on.

Figure 20(c) illustrates the construction we used.

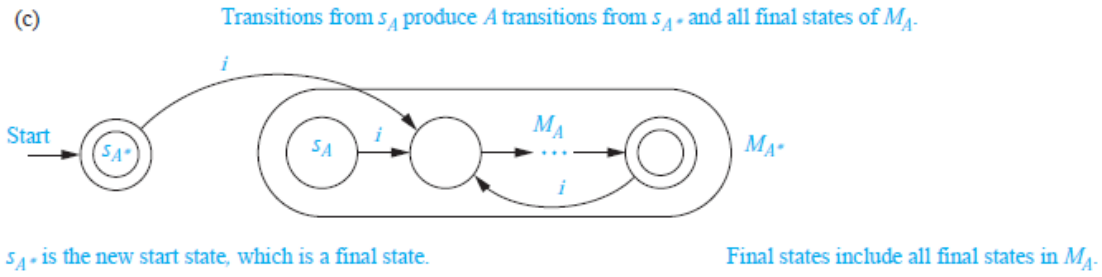


FIGURE 20. Building Automata to Recognize Kleene Closures.

"IF" part of Kleene's theorem.

Proof. Let A be a finite alphabet, $A=\{a_1, ..., a_n\}$. Let $M=(A, S, f, q_1, F)$ be a DFSA, here:

- S – set of states;
- q_1 - start state;
- f – next state function;
- F – set of final states.

Let $L \subseteq A^*$ be the language recognized by M, that is, $L(M)=L$. We must prove that L is a regular set.

Let $S=\{q_1, ..., q_r\}$. We start with hypothesis that F contains only one state, $F=\{q_r\}$. The general case is a trivial consequence of this case.

Define sets $Z^k_{ij} \subseteq A^*$, as following: Z^k_{ij} is the set of strings which take the automaton M from the state q_i to state q_j passing only through states from the set $\{q_1, ..., q_k\}$, that is,

$Z^k_{ij}=\{\alpha \in A^* | P(\alpha, q_i)=q_j, P(\beta, q_i) \in \{q_1, ..., q_k\}, \alpha=\beta\gamma, \beta, \gamma \in A^*, \beta, \gamma \neq \lambda\}$. Here $P(\alpha, q_i)=q_j$ is the path from q_i to q_j implemented by the string α.

Now we prove, by induction over k, that each of Z^k_{ij} is a regular set.

Base step: The set Z^0_{ij} consists of all words in the alphabet A that transfer M from state q_i to state q_j without passing through intermediate state.

- If i=j, then Z^0_{ij} contains λ and the subset $B \subseteq A$, defined by $B = \{a \in A | P(a, q_i)=f(a, q_i)= q_i\}$, in other words, all possible loops at a state q_i determined by elements from B (substrings in L).
- If $i \neq j$, then $Z^0_{ij}=\{a \in A | f(a, q_i)=q_j\}$.

Therefore, all Z^0_{ij} are regular in both cases.

Inductive Step: Let Z^k_{ij} are regular. Consider the set Z^{k+1}_{ij}. It contains all the words from A which take automaton M from state q_i to state q_j passing only through states $q_1, ..., q_k, q_{k+1}$.

Let $\alpha \in Z^{k+1}_{ij}$, т.е. $P(\alpha, q_i)=q_j$. Let us see what intermediate states the automaton M goes through when reading the word α:

1) if the automaton M does not pass through the state q_{k+1}, then $\alpha \in Z^k_{ij}$;

2) if the automaton M passes through the state q_{k+1}, then string α can be written as

$\alpha = \beta\gamma_1 ... \gamma_t\delta$, where $\beta, \gamma_1, ..., \gamma_t, \delta \in A^*$, and

- $\beta \in Z^k_{i(k+1)}$ - path from state q_i to state q_{k+1};
- $\gamma_1 . . . \gamma_t \in Z^k_{(k+1)(k+1)}$ - several paths from state q_{k+1} to itself (cycles);

- $\delta \in Z^k_{(k+1)j}$ – path from q_{k+1} to q_j

Note that in all these paths state q_{k+1} is the start or end state of the corresponding path.

Thus, $Z^{k+1}_{ij}=Z^k_{ij} \cup Z^k_{i(k+1)}(Z^k_{(k+1)(k+1)})^* Z^k_{(k+1)j}$. Therefore Z^{k+1}_{ij} is the regular set.

Note now that $L=Z^r_{1r}$, that is, L is a regular set.

If set F of final states consists of several states q_r, q_v, …, q_w then $L=Z^r_{1r} \cup Z^v_{1v} \cup \ldots \cup Z^w_{1w}$ (union of regular sets is regular). The **"IF" part of Kleene theorem** is proved. ■

Theorem 4. (Alternative proof of "Only If" part of Kleene's Theorem).

Proof. Here we demonstrate alternative proof "Only If" part of Kleene's Theorem for the Concatenation AB, Union A∪B and A^* using λ-moves.

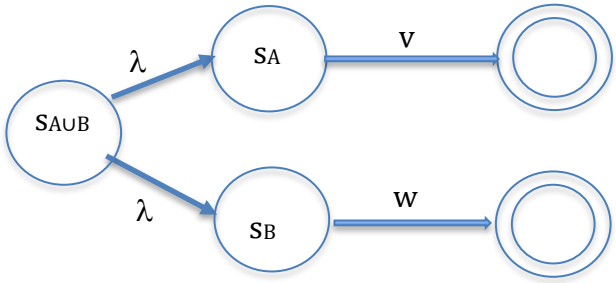
4′. Concatenation of Machines. Sections 4′.1, 4′.2, and 4′.3 are identically the same as sections 4.1, 4.2, and 4.3. We consider section 4′.4, that is, transitions in M_{AB}. Transitions in M_{AB} include all transitions in M_A and in M_B, as well as some new λ-transitions: for every final state *s* in M_A we form λ-transition in M_{AB} from *s* to s_B. In this way, a string in A takes M_{AB} from s_{AB} to s_B, and then a string in B takes s_B to a final state of M_{AB} as follows:

- if $u \in A$ with accepting state in s_m (u defines a path from s_A to s_m in M_A) and
- if $w \in B$ with accepting state in s_k (w defines a path from s_B to s_k in M_B) start state in s_B and final state in s_K
- if λ is considered as λ-transition from s_m to s_B

Then the word uw=uλw belongs, by definition, to AB and generates an accepting path from $s_{AB}=s_A$ to s_k in automaton M_{AB}.

Moreover, empty word belongs to A ($\lambda \in A$) then we define new λ-transition in M_{AB} from $s_{AB}=s_A$ to s_B. Under last λ-transition all concatenations λw, $\lambda \in A$, $w \in B$ are acceptable words in machine M_{AB}.

5′. Union of Machines. Sections 5′.1, 5′.2, and 5′.3 are identically the same as sections 5.1, 5.2, and 5.3. We consider section 5′.4, that is, transitions in $M_{A \cup B}$. The transitions in $M_{A \cup B}$ include all those in M_A and in $M_{(B)}$. Also, include new λ-transitions from the state $s_{A \cup B}$ to s_A and to s_B. In this way, a string v in A considered as λv leads from $s_{A \cup B}$ to a final state in the new machine, and a string w in B considered as λw leads from $s_{A \cup B}$ to a final state in the new machine.



6′. Kleene's closure of a Machine. Sections 6′.1, 6′.2, and 6′.3 are identically the same as sections 6.1, 6.2, and 6.3. We consider section 6′.4, that is, transitions in $M_{A \cup B}$. To recognize concatenations of arbitrarily many strings from A, we include all the transitions in M_A, as well as λ-transition from s_{A^*} to s_A, and transitions from each final state that match the transitions from s_{A^*}. With this set of transitions, a string made up of concatenations of strings from A will take s_{A^*} to a final state when the first string in A has been read, returning to a final state when the second string in A has been read, and so on. ■

NDFSA can be constructed for any regular set using the procedure described in **"Only If"** part of the Theorem 3 (Kleene's method) or in Theorem 4 (ε-moves method).

We illustrate how this is done with Example 16 (Kleene's method).

EXAMPLE 16. Construct a NDFSA that recognizes the regular set $\mathbf{1}^*\cup\mathbf{01}$.

Solution: We begin by building a machine that recognizes $\mathbf{1}^*$. This is done using the machine that recognizes **1** and then using the construction for M_{A^*} described in the proof. Next, we build a machine that recognizes **01**, using machines that recognize **0** and **1** and the construction in the proof for M_{AB}. Finally, using the construction in the proof for $M_{A\cup B}$, we construct the machine for $\mathbf{1}^*\cup\mathbf{01}$. The finite-state automata used in this construction are shown in Figure 21. The states in the successive machines have been labeled using different subscripts, even when a state is formed from one previously used in another machine. Note that the construction given here does not produce the simplest machine that recognizes $\mathbf{1}^*\cup\mathbf{01}$. A much simpler machine that recognizes this set is shown in Figure 21(b). ■

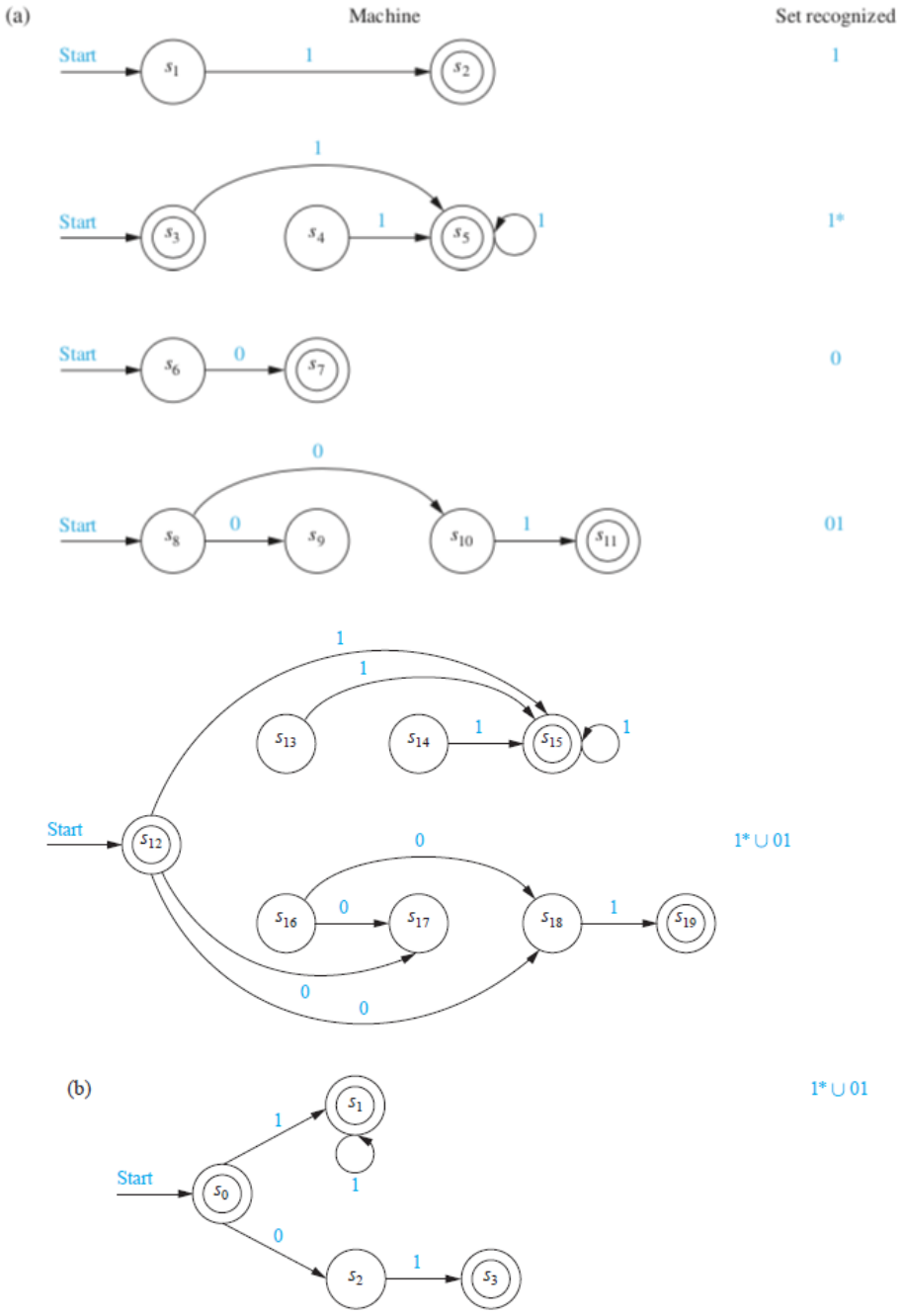


FIGURE 21. Nondeterministic Finite-State Automata Recognizing $\mathbf{1}^*\cup\mathbf{01}$.

A Set Not Recognized by a Finite-State Automaton.

We have seen that a set is recognized by a FSA if and only if it is regular. One important technique used to prove that certain sets are not regular is the pumping lemma.

Theorem 4 (Pumping Lemma). Suppose M = (S, I, f, s_0, F) is an automaton over I such that:

(i) M has k states.

(ii) M accepts a word w where |w|>k.

Then there exist subwords x, y, z such that w=xyz where, for every positive integer m, $w_m=xy^mz$ is accepted by M.

Proof: Suppose $w=a_1a_2 \dots a_n$ is a word over I accepted by M and suppose |w|=n>k, k is the number of states. Let $P=(s_0, s_1, ..., s_n)$ be the sequence of states determined by the word w. Since w is acceptable word so s_n is the accepting state. Since n>k, two of the states in P must be equal, say $s_i=s_j$ where i<j. Divided w into subwords x, y, z as follows: $x=a_1a_2\dots a_i$, $y=a_{i+1}\cdots a_j$, (y is non-empty word due to i<j), $z=a_{j+1}\cdots a_n$. As shown in Figure 22, xy ends in $s_i=s_j$; hence xy^m also ends in s_i. Thus, for every m, $w_m=xy^mz$ ends in s_n, which is an accepting state. Theorem is proved. ■

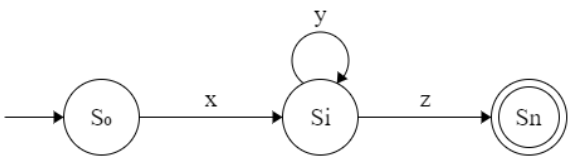


Figure 22.

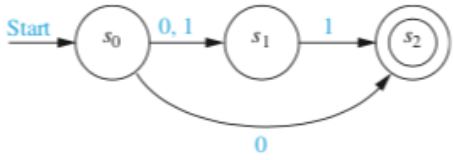
EXAMPLE 17. Show that the language $L=\{a^mb^m \mid m \text{ is positive}\}$ is not regular.

Solution. Assume the contrary. Suppose L is regular. Then, by Theorem 3, there exists a finite state automaton M which accepts L. Suppose M has k states. Let $w=a^kb^k$. Then |w|>k. By Pumping Lemma, w=xyz where y is not empty and $w_2=xy^2z$ is also accepted by M. If *y* consists of only *a*'s or only *b*'s, then w_2 will not have the same number of *a*'s as *b*'s. If y contains both *a*'s and *b*'s, then w_2 will have *a*'s following *b*'s. In either case w_2 does not belong to L, which is a contradiction. Thus, L is not regular. ■

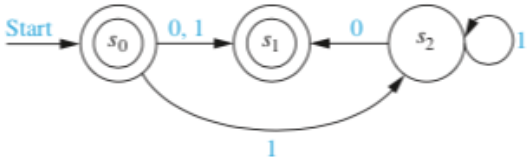
EXERCISES. SET 4 (KR Textbook, selected exercises from chapter 13.3)

In Exercises **4**.1–**4**.7 find the language recognized by the given NDFSA M.

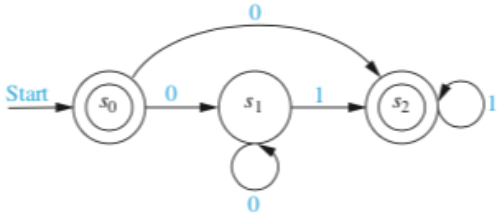
4.1



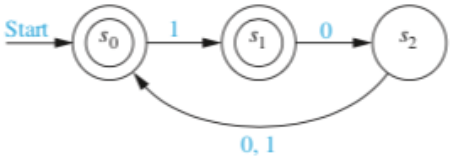
4.2



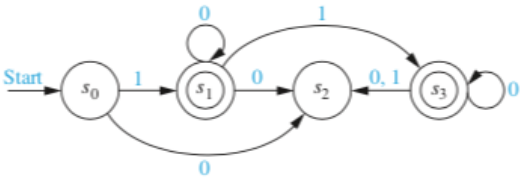
4.3



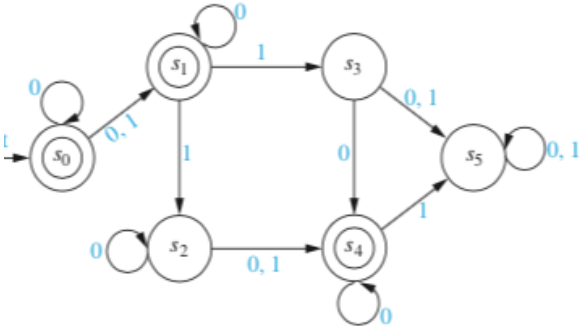
4.4



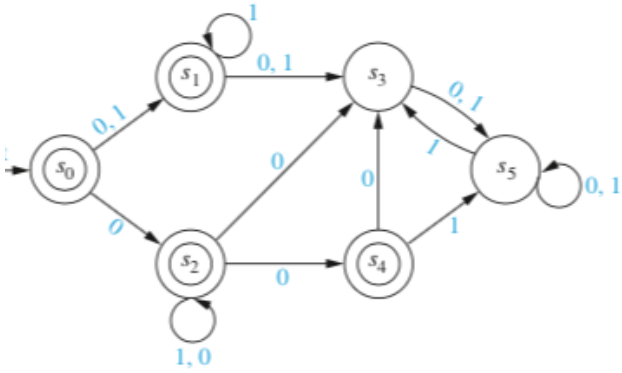
4.5



4.6



4.7



4.8. Find a deterministic finite-state automaton (DFSA) that recognizes the same language as the nondeterministic finite-state automaton (NDFSA) in Exercise **4**.1.

4.9 Find a DFSA that recognizes the same language as the NDFSA in Exercise **4**.2.

4.10 Find a DFSA that recognizes the same language as the NDFSA in Exercise **4**.3.

4.11 Find a DFSA that recognizes the same language as the NDFSA in Exercise **4**.4.

4.12 Find a DFSA that recognizes the same language as the NDFSA in Exercise **4**.5.

4.13 Find a DFSA that recognizes each of these sets:

a) {0} b) {1, 00} c) $\{1^n \mid n=2, 3, 4, ...\}$

4.14 Find a NDFSA that recognizes each of the languages in Exercise **4**.13, and has fewer states, if possible, than the DFSA you found in that exercise.

Answers (SET 4).

A.4.1 L(M)={0, 01, 11}

A.4.2 L(M)=$\{\lambda, 0, 1\} \cup \{1^n 0 \mid n \geq 1\}$

A.4.3 L(M)=$\{\lambda, 0\} \cup \{0^m 1^n \mid m \geq 1, n \geq 1\}$

A.4.4 Let x=1, y=100, z=101. Then

L(M)={w, wx | w∈(y, z)*, that is, w is any word over input symbols y and z}

A.4.5 L(M)=$\{10^n \mid n \geq 0\} \cup \{10^n 10^m \mid n,m \geq 0\}$

A.4.6 Let A=$\{0^n 00^m 100^k \mid n, m, k \geq 0\}$,

B=$\{0^n 10^n 100^k \mid n, m, k \geq 0\}$,

C=$\{0^n 00^m 10^p 00^k \mid n, m, p, k \geq 0\}$,

D=$\{0^n 00^m 10^p 10^k \mid n, m, p, k \geq 0\}$,

E=$\{0^n 10^m 10^p 00^k \mid n, m, p, k \geq 0\}$,

F=$\{0^n 10^m 10^p 10^k \mid n, m, p, k \geq 0\}$

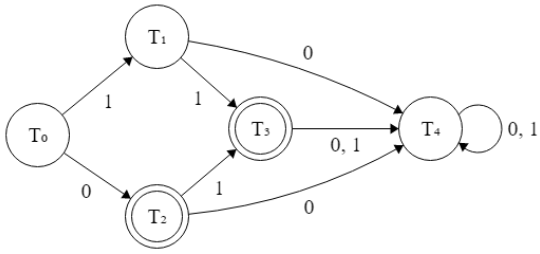
It is easy to see that A=C=$\{0^x 10^y \mid x, y \geq 1\}$=R, B=E=$\{0^n 10^m 10^y \mid n, m \geq 0, y \geq 1\}$=S

$D=\{0^x10^p10^k \mid p, k\geq 0, x\geq 1\}$, $F=\{0^n10^m10^p10^k \mid n, m, p, k\geq 0\}$

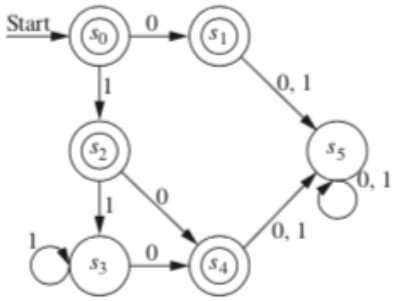
Thus L(M)=RUSUDUF

A.4.7 The union of the set of all strings that start with a 0 and the set of all strings that have no 0s.

A.4.8



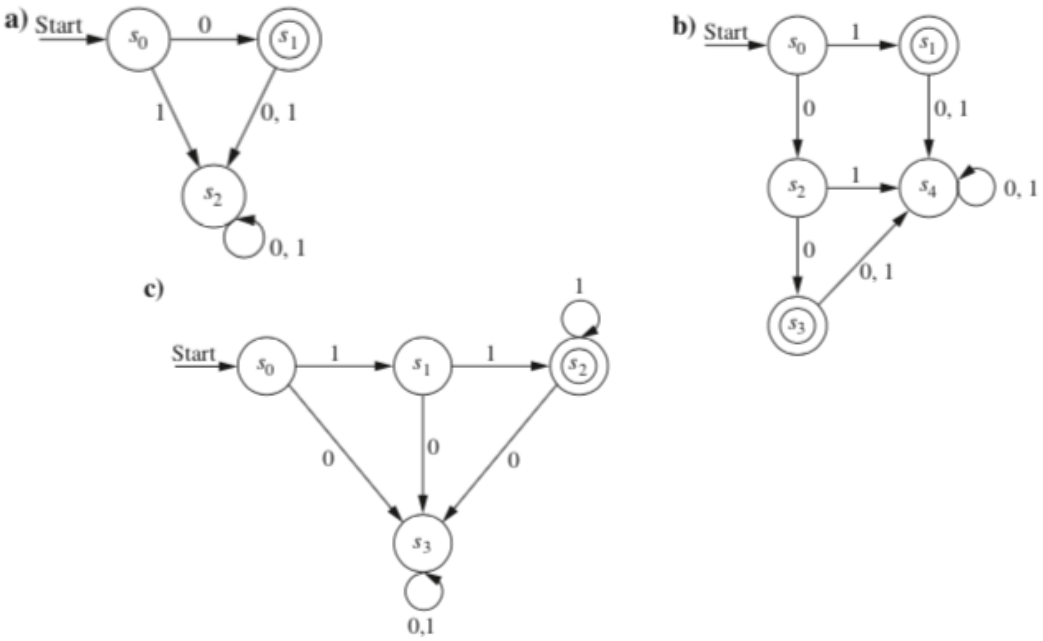
A.4.9.



A.4.11 Add a nonfinal state s_3 with transitions to s_3 from s_0 on input 0, from s_1 on input 1, and from s_3 on input 0 or 1.

Note. There exists another more optimal state diagram for the task **5.9** with 5 states (3 - finals)

A.4.13



A.4.14 NDFSAs with fewer states which recognizes the same languages as related DFSAs in exercises 4.13 are as following:

A4.14 a). s_1 is a final state;

Table 6a (NDFSA)		
State	f	
	Input	
	0	1
s_0	s_1	
s_1		

A4.14 b). s_1 and s_3 are final states;

Table 6b (NDFSA)		
State	f	
	Input	
	0	1
s_0	s_2	s_1
s_1		
s_2	s_3	
s_3		

A4.14 c). s_2 is a final state:

Table 6c (NDFSA)		
State	f	
	Input	
	0	1
s_0		s_1
s_1		s_2
s_2		

EXERCISES. SET 5 (KR Textbook, selected exercises from chapter 13.4)

5.1 Construct nondeterministic finite-state automata that recognize each of these sets.

a) {λ, 0} **b)** {0, 11} **c)** {0, 11, 000}

5.2 **The reversal** of a string is the string consisting of the symbols of the string in reverse order. The reversal of the string w is denoted by w^R. Show that if A is a regular set, then A^R, the set of all reversals of strings in A, is also regular.

5.3 Using the constructions described in the proof of Kleene's theorem, find nondeterministic finite-state automata that recognize each of these sets. **a)** $\mathbf{01}^*$ **b)** $\mathbf{(0 \cup 1)1}^*$ **c)** $\mathbf{00(1}^* \mathbf{\cup 10)}$

5.4 Using the constructions described in the proof of Kleene's theorem, find nondeterministic finite-state automata that recognize each of these sets.

a) $\mathbf{0}^*\mathbf{1}^*$ **b)** $\mathbf{(0 \cup 11)}^*$ **c)** $\mathbf{01}^* \mathbf{\cup 00}^*\mathbf{1}$

5.5 Show that the set $\{0^{2n}1^n \mid n=0, 1, 2, ...\}$ is not regular using the pumping lemma.

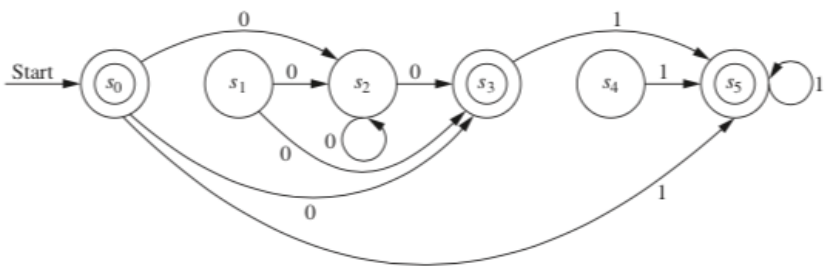
5.6 Show that the set $\{1^{n^2} \mid n = 0, 1, 2, ...\}$ is not regular using the pumping lemma.

5.7 Show that the set of palindromes over {0, 1} is not regular using the pumping lemma. [*Hint:* consider strings of the form 0^N10^N]

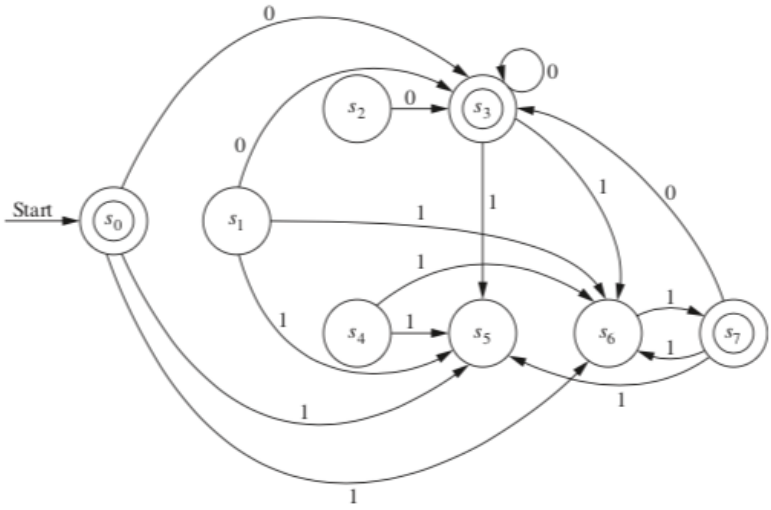
Answers (SET 2).

A5.2 Use an inductive proof. If the regular expression for A is ∅, λ, or x, the result is trivial. Otherwise, suppose the regular expression for A is **BC**. Then A=BC where B is the set generated by **B** and C is the set generated by **C**. By the inductive hypothesis there are regular expressions $\mathbf{B}'$ and $\mathbf{C}'$ that generate B^R and C^R, respectively. Because $A^R=(BC)^R=C^RB^R$, $\mathbf{C}'\mathbf{B}'$ is a regular expression for A^R. If the regular expression for A is $\mathbf{B} \cup \mathbf{C}$, then the regular expression for A^R is $\mathbf{B}' \cup \mathbf{C}'$ because $(B \cup C)^R=(B^R) \cup (C^R)$. Finally, if the regular expression for *A* is $\mathbf{B}^*$, then it is easy to see that $(\mathbf{B}')^*$ is a regular expression for A^R.

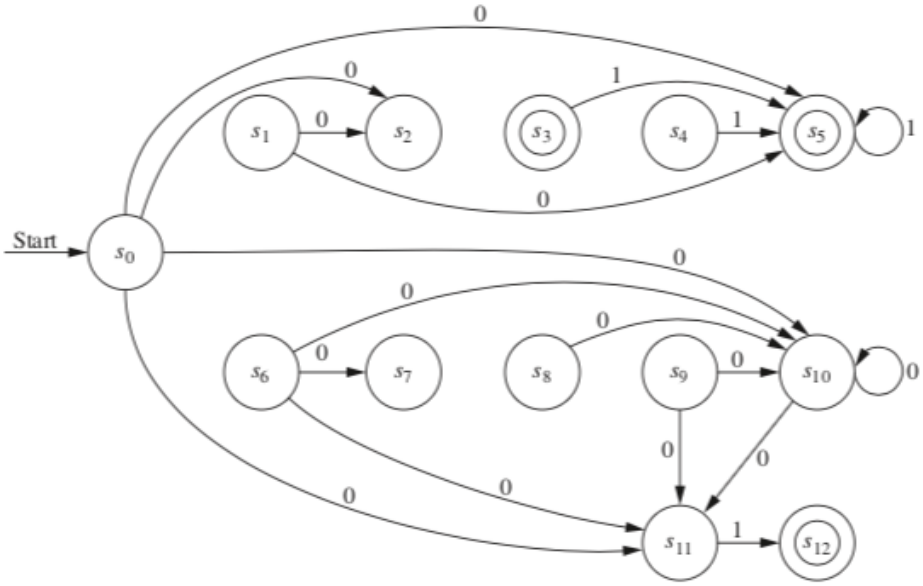
A5.4 a



A5.4 b



A5.4 c



A5.5 Suppose that $L=\{0^{2n}1^n \mid n=0, 1, 2, ...\}$ were regular. Let S be the set of states of a finite-state machine recognizing this set. Let $z=0^{2n}1^n$ where $3n\geq|S|$. Then by Pumping lemma, $z=0^{2n}1^n=uvw$, $l(v)\geq 1$, and $uv^iw\in\{0^{2n}1^n \mid n \geq 0\}$. Obviously v cannot contain both 0 and 1, because v^2 would then contain 10. So, v is all 0s or all 1s, and hence, uv^2w contains too many 0s or too many 1s, so it is not in L. This contradiction shows that L is not regular.

A5.7 Suppose that the set of palindromes over {0, 1} were regular. Let S be the set of states of a finite-state machine recognizing this set. Let $z=0^n10^n$, where $n>|S|$. Apply the pumping lemma to get $uv^iw\in L$ for all nonnegative integers i where $l(v)\geq 1$, and $l(uv)\leq|S|$, and $z=0^n10^n=uvw$. Then v must be a string of 0s (because $n>|S|$), so uv^2w is not a palindrome. Hence, the set of palindromes is not regular.

PART 4. FINITE-STATE MACHINES WITH OUTPUTS (TRANSDUCERS).

In this Part, we study those finite-state machines that produce output. We show how finite-state machines can be used to model a vending machine, a machine that delays input, a machine that adds

integers, and a machine that determines whether a bit string contains a specified pattern.

Before giving formal definitions, we will show (follow to the KR textbook) how a vending machine can be modeled.

EXAMPLE 18 (KR). A vending machine accepts nickels (5 cents), dimes (10 cents), and quarters (25 cents). When a total of 30 cents or more has been deposited, the machine immediately returns the amount in excess of 30 cents. When 30 cents has been deposited and any excess refunded, the customer can push an orange button and receive an orange juice or push a red button and receive an apple juice.

Describe how the machine works by specifying its states, how it changes states when input is received, and the output that is produced for every combination of input and current state.

Solution. The machine can be in any of seven different states s_i, i=0, 1, 2, ..., 6, where s_i is the state where the machine has collected 5i cents. The machine starts in state s_0, with 0 cents received. The possible inputs are 5 cents, 10 cents, 25 cents, the orange button (O), and the red button (R). The possible outputs are nothing (n), 5 cents, 10 cents, 15 cents, 20 cents, 25 cents, an orange juice, and an apple juice.

We illustrate how this model of the machine works with this example. Suppose that a student puts in a dime followed by a quarter, receives 5 cents back, and then pushes the orange button for an orange juice. The machine starts at the state s_0. The first input is 10 cents, which changes the state of the machine to s_2 and gives no output. The second input is 25 cents. This changes the state from s_2 to s_6 and gives 5 cents as output. The next input is the orange button, which changes the state from s_6 back to s_0 (because the machine returns to the start state) and gives an orange juice as its output.

We can display all the state changes and output of this machine in a table. To do this we need to specify for each combination of state and input the next state and the output obtained. Table 7 below shows the transitions and outputs for each pair of a state and an input. ■

Another way to show the actions of a machine is to use a directed graph with labeled edges, where each state is represented by a circle, edges represent the transitions, and edges are labeled with the input and the output for that transition. Figure 23 shows such a directed graph for the vending machine.

TABLE 7. State Table for a Vending Machine

TABLE 7	**State Table for a Vending Machine.**									
	Next State					**Output**				
State	***Input***					***Input***				
	5	**10**	**25**	**O**	**R**	**5**	**10**	**25**	**O**	**R**
s_0	s_1	s_2	s_5	s_0	s_0	n	n	n	n	n
s_1	s_2	s_3	s_6	s_1	s_1	n	n	n	n	n
s_2	s_3	s_4	s_6	s_2	s_2	n	n	5	n	n
s3	s_4	s_5	s_6	s_3	s_3	n	n	10	n	n
s_4	s_5	s_6	s_6	s_4	s_4	n	n	15	n	n
s_5	s_6	s_6	s_6	s_5	s_5	n	5	20	n	n
s_6	s_6	s_6	s_6	s_0	s_0	5	10	25	OJ	AJ

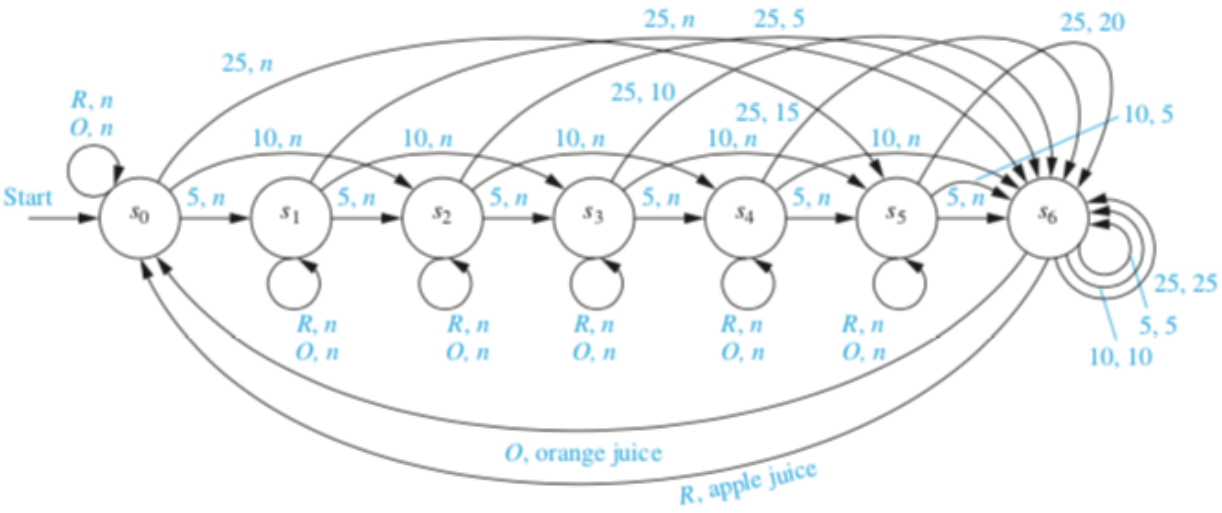


FIGURE 23 **A Vending Machine.**

Definition 11. A *finite-state machine* M = (S, I, O, f, g, s_0) consists of a finite set S of *states*, a finite *input alphabet* I, a finite *output alphabet* O, a *transition function* f that assigns to each state and input pair a new state, an *output function* g that assigns to each state and input pair an output, and an *initial state* s_0. ■

Let M = (S, I, O, f, g, s_0) be a finite-state machine. We can use a **state table** to represent the values of the transition function *f* and the output function *g* for all pairs of states and input. We previously constructed a state table for the vending machine discussed in the introduction to this section.

EXAMPLE 19. The state table shown in Table 8 describes a finite-state machine with S={s_0, s_1, s_2, s_3}, I={0, 1}, and O={0, 1}. The values of the transition function *f* are displayed in the first two columns, and the values of the output function g are displayed in the last two columns. ■

Table 8. State table for Example 19.

State	f - Next State Function		g – Output Function	
	Input		Input	
	0	1	0	1
s_0	s_1	s_0	1	0
s_1	s_3	s_0	1	1
s_2	s_1	s_2	0	1
s_3	s_2	s_1	0	0

Another way to represent a finite-state machine is to use a **state diagram**, which is a directed graph with labeled edges. In this diagram, each state is represented by a circle. Arrows labeled with the input and output pair are shown for each transition.

EXAMPLE 20. Construct the state diagram for the finite-state machine with the state table shown in Table 8.

Solution: The state diagram for this machine is shown in Figure 24. ■

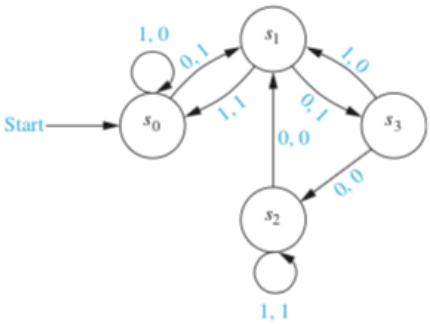


FIGURE 24. **The State Diagram for the Finite-State Machine Shown in Table 8.**

EXAMPLE 21. State Table 9 corresponds to the state diagram for the FSA shown in Figure 25. ■

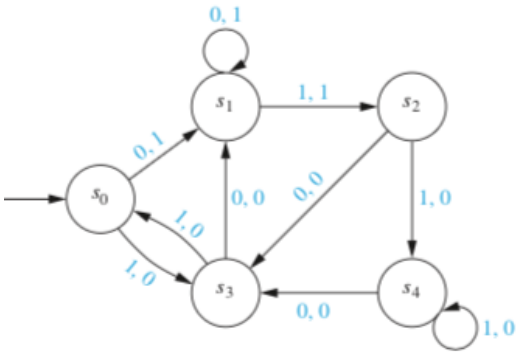


Table 9. State table for Example 21.

State	f - Next State Function		g – Output Function	
	Input		Input	
	0	1	0	1
s_0	s_1	s_3	1	0
s_1	s_1	s_2	1	1
s_2	s_3	s_4	0	0
s_3	s_1	s_0	0	0
s_4	s_3	s_4	0	0

Figure 25. **A Finite-State Machine.**

An input string takes the starting state through a sequence of states, as determined by the transition function. As we read the input string symbol by symbol (from left to right), each input symbol takes the machine from one state to another. Because each transition produces an output, an input string also produces an output string.

Suppose that the input string is $x=x_1x_2 \ldots x_k$. Then, reading this input takes the machine from state s_0 to state s_1, where $s_1=f(s_0, x_1)$, then to state s_2, where $s_2=f(s_1, x_2)$, and so on, with $s_j=f(s_{j-1}, x_j)$ for j=1, 2, ..., k, ending at state $s_k=f(s_{k-1}, x_k)$. This sequence of transitions produces an output string $y_1y_2\ldots y_k$, where $y_1=g(s_0, x_1)$ is the output corresponding to the transition from s_0 to s_1, $y_2=g(s_1, x_2)$ is the output corresponding to the transition from s_1 to s_2, and soon. In general, $y_j=g(s_{j-1}, x_j)$ for j=1, 2, ..., k. Hence, **we can extend the definition of the output function *g* to input strings so that g(x)=y, where y is the output corresponding to the input string *x*.** This notation is useful in many applications.

EXAMPLE 22. Find the output string generated by the finite-state machine in Figure 25 if the input string is 101011.

Solution: The output obtained is 001000. The successive states and outputs are in Table 10.■

Table 10

Input	1	0	1	0	1	1	--
State (the function f works)	s_0	s_3	s_1	s_2	s_3	s_0	s_3
Output (the function g works)	0	0	1	0	0	0	--

We can now look at some examples of useful finite-state machines. Examples 23, 24, and 25 illustrate that the states of a finite-state machine give it limited memory capabilities. The states can be used to remember the properties of the symbols that have been read by the machine. However, because there are only finitely many different states, finite-state machines cannot be used for some important purposes.

EXAMPLE 23. An important element in many electronic devices is a *unit-delay machine*, which produces as output the input string delayed by a specified amount of time. How can a finite-state machine be constructed that delays an input string by one unit of time, that is, produces as output the bit string $0x_1x_2\ldots x_{k-1}$ given the input bit string $x_1x_2\ldots x_{k-1}x_k$?

Solution: A delay machine can be constructed that has two possible inputs, namely, 0 and 1. The machine must have a start state s_0. Because the machine has to remember whether the previous input was a 0 or a 1, two other states s_1 and s_2 are needed, where the machine is in state s_1 if the previous input was 1 and in state s_2 if the previous input was 0. An output of 0 is produced for the initial transition from s_0. Each transition from s_1 gives an output of 1, and each transition from s_2 gives an output of 0. The output corresponding to the input of a string $x_1x_2\ldots x_{k-1}x_k$ is the string that begins with 0, followed by x_1, followed by x_2, ..., ending with x_{k-1}. The state diagram for this machine is shown in Figure 26. ■

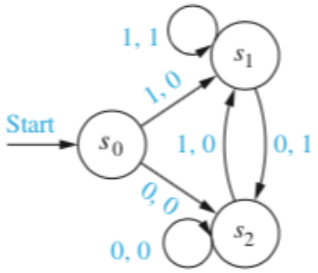


FIGURE 26 A Unit-Delay Machine

EXAMPLE 23A. Find the output string generated by the finite-state machine in Figure 26 if the input string is a) 10101100; b) 00111011.

Solution: The output obtained are: for (a) 01010110, for (b) 00011101 in accordance with general description in Example 23. The successive states and outputs are in Table 11. ■

Table 11

Input (a)	1	0	1	0	1	1	0	0	--
Next-State (the function f works)	s_0	s_1	s_2	s_1	s_2	s_1	s_1	s_2	s_2
Output (the function g works)	0	1	0	1	0	1	1	0	--
Input (b)	0	0	1	1	1	0	1	1	--
Next-State (the function f works)	s_0	s_2	s_2	s_1	s_1	s_1	s_2	s_1	s_1
Output (the function g works)	0	0	0	1	1	1	0	1	--

EXAMPLE 24. Produce a finite-state machine that adds two positive integers using their binary expansions.

Solution: When $(x_n \ldots x_1x_0)_2$ and $(y_n \ldots y_1 y_0)_2$ are added, the following procedure is followed. First, the bits x_0 and y_0 are added, producing a sum bit z_0 and a carry bit c_0. This carry bit is either 0 or 1.

Then, the bits x_1 and y_1 are added, together with the carry c_0. This gives a sum bit z_1 and a carry bit c_1. This procedure is continued until the nth stage, where x_n, y_n, and the previous carry c_{n-1} are added to produce the sum bit z_n and the carry bit c_n, which is equal to the sum bit z_{n+1}.

A finite-state machine to carry out this addition can be constructed using just two states. For simplicity we assume that both the initial bits x_n and y_n are 0 (otherwise we have to make special arrangements concerning the sum bit z_{n+1}). The start state s_0 is used to remember that the previous carry is 0 (or for the addition of the rightmost bits). The other state, s_1, is used to remember that the previous carry is 1.

Because the inputs to the machine are pairs of bits, there are four possible inputs. We represent these possibilities by 00 (when both bits are 0), 01 (when the first bit is 0 and the second is 1), 10 (when the first bit is 1 and the second is 0), and 11 (when both bits are 1). The transitions and the outputs are constructed from the sum of the two bits represented by the input and the carry represented by the state. For instance, when the machine is in state s_1 and receives 01 as input, the next state is s_1 and the output is 0, because the sum that arises is $0+1+1=(10)_2$. The state diagram for this machine is shown in Figure 27. ■

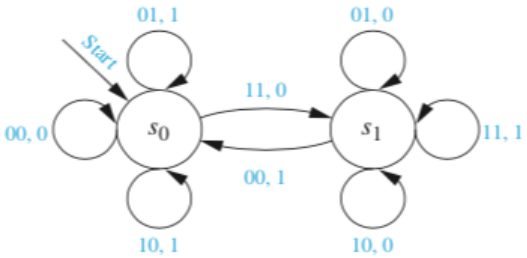


FIGURE 27. **A Finite-State Machine for Addition of Integers in Binary Notation.**

By the way, the algorithm for addition can be described using pseudocode as follows.

ALGORITHM 1. Addition of Integers in Binary Notation.

procedure *add*(*a, b*: positive integers){the binary expansions of a and b are $(a_{n-1}a_{n-2}\ldots a_1a_0)_2$ and $(b_{n-1}b_{n-2}\ldots b_1b_0)_2$, respectively}

c:=0

for j:=0 **to** n−1

d:=⌊$(a_j+b_j+c)/2$⌋

$s_j:=a_j+b_j+c-2d$

c:=d

next j

s_n:=c

return (s_0, s_1, ..., s_n) {the binary expansion of the sum is $(s_ns_{n-1}\ldots s_0)_2$}

The worst-case complexity of Algorithm 1 to add two n-bit integers in terms of required additions of bits is clearly O(n). ■

EXAMPLE 25. In a certain coding scheme, when three consecutive 1s appear in a message, the receiver of the message knows that there has been a transmission error. Construct a finite-state machine that gives a 1 as its current output bit if and only if the last three bits received are all 1s.

Solution: Three states are needed in this machine. The start state s_0 remembers that the previous input value, if it exists, was not a 1. The state s_1 remembers that the previous input was a 1, but the input before the previous input, if it exists, was not a 1. The state s_2 remembers that the previous two inputs were 1s.

An input of 1 takes s_0 to s_1, because now a 1, and not two consecutive 1s, has been read; it takes s_1 to s_2, because now two consecutive 1s have been read; and it takes s_2 to itself, because at least two consecutive 1s have been read. An input of 0 takes every state to s_0, because this breaks up any string of consecutive 1s. The output for the transition from s_2 to itself when a 1 is read is 1, because this combination of input and state shows that three consecutive 1s have been read. All other outputs are 0. The state diagram of this machine is shown in Figure 28. ■

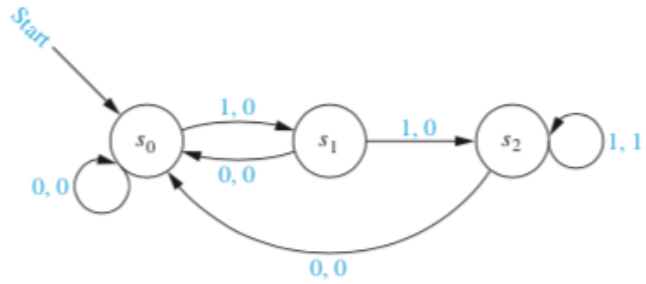


Figure 28. **A Finite-State Machine That Gives an Output of 1 If and Only If the Input String Read So Far Ends with 111.**

The final output bit of the finite-state machine we constructed in Example 25 is 1 if and only if the input string ends with 111. Because of this, we say that this finite-state machine **recognizes** the set of bit strings that end with 111. This leads us to Definition 12.

Definition 12. Let $M = (S, I, O, f, g, s_0)$ be a finite-state machine and $L \subseteq I^*$. We say that M *recognizes* (or *accepts*) L if an input string x belongs to L if and only if the last output bit produced by M when given x as input is a 1. ■

TYPES OF FINITE-STATE MACHINES. Many kinds of finite-state machines have been developed to model computing machines. In this section we have given a definition of one type of finite-state machine. In the type of machine introduced in this section, outputs correspond to transitions between states. Machines of this type are known as **Mealy machines** because they were first studied by G. H. Mealy in 1955. In Example 25 we showed how a Mealy machine can be used for language recognition. For such purposes (language recognition) are usually used, as we have already seen earlier in previous Parts of these Lecture Notes, another type of finite-state machine, giving no output. Remind you that such machines, also known as finite-state automata, have a set of final states and recognize a string if and only if it takes the start state to a final state.

There is another important type of finite-state machine with output, where the output is determined only by the state. This type of finite-state machine is known as a **Moore machine** (E.F. Moore introduced this type of machine in 1956).

Definition 13. A **Moore machine** $M=(S, I, O, f, g, s_0)$ consists of a finite set of states, an input alphabet I, an output alphabet O, a transition function f that assigns a next state to every pair of a state and an input, an output function g that assigns an output to every state, and a starting state s_0 ■

A Moore machine can be represented either by a table listing the transitions for each pair of state and input and the outputs for each state, or by a state diagram that displays the states, the transitions between states, and the output for each state. In the diagram, transitions are indicated with arrows labeled with the input, and the outputs are shown next to the states.

EXERCISES. SET 6 (KR Textbook, selected exercises from chapter 13.2)

6.1. Draw the state diagrams for the finite-state machines with these state tables.

a)

State	f - Next State Function		g – Output Function	
	Input		Input	
	0	1	0	1
s_0	s_1	s_0	0	1
s_1	s_0	s_2	0	1
s_2	s_1	s_1	0	0

b)

State	f - Next State Function		g – Output Function	
	Input		Input	
	0	1	0	1
s_0	s_1	s_0	0	0
s_1	s_2	s_0	1	1
s_2	s_0	s_3	0	1
s_3	s_1	s_2	1	0

c).

State	f - Next State Function		g – Output Function	
	Input		Input	
	0	1	0	1
s_0	s_0	s_4	1	1
s_1	s_0	s_3	0	1
s_2	s_0	s_2	0	0
s_3	s_1	s_1	1	1
s_4	s_1	s_0	1	0

6.2. Find the output generated from the input string 01110 for the finite-state machine with the state table in

a) Exercise1(a). b) Exercise 1(b). c) Exercise 1(c).

6.3. Find the output for each of these input strings when given as input to the finite-state machine in Example 20 (Figure 24).

a) 0111 b) 11011011 c) 01010101010

6.4. Find the output for each of these input strings when given as input to the finite-state machine in Example 21 (Figure 25).

a) 0000 b) 101010 c) 11011100010

6.5. Construct a finite-state machine that models an old- fashioned soda machine that accepts nickels, dimes, and quarters. The soda machine accepts change until 35 cents has been put in. It gives change back for any amount greater than 35 cents. Then the customer can push but- tons to receive either a cola, a root beer, or a ginger ale.

6.6. Construct a finite-state machine that delays an input string two bits, giving 00 as the first two bits of output.

6.7. Construct a finite-state machine for the log-on procedure for a computer, where the user logs on by entering a user identification number, which is considered to be a single input, and then a password, which is considered to be a single input. If the password is incorrect, the user is asked for the user identification number again.

6.8. Construct a finite-state machine for a toll machine that opens a gate after 25 cents, in nickels, dimes, or quarters, has been deposited. No change is given for overpayment, and no credit is given to the next driver when more than 25 cents has been deposited.

6.9. Construct a finite-state machine for a restricted telephone switching system that implements

these rules. Only calls to the telephone numbers 0, 911, and the digit 1 followed by 10-digit telephone numbers that begin with 212, 800, 866, 877, and 888 are sent to the network. All other strings of digits are blocked by the system and the user hears an error message.

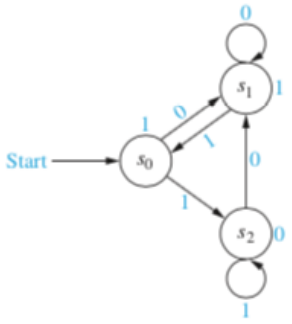
6.10. Construct a finite-state machine that determines whether the input string has a 1 in the last position and a 0 in the third to the last position read so far.

6.11. Construct a finite-state machine that determines whether the word computer has been read as the last eight characters in the input read so far, where the input can be any string of English letters.

6.12. Construct the state diagram for the Moore machine with this state table.

State	f		g
	Input		
	0	1	
s_0	s_0	s_2	0
s_1	s_3	s_0	1
s_2	s_2	s_1	1
s_3	s_2	s_0	1

6.13. Construct the state table for the Moore machine with the state diagram shown here. Each input string to a Moore machine M produces an output string. In particular, the output corresponding to the input string $a_1a_2\ldots a_k$ is the string $g(s_0)g(s_1)...g(s_k)$, where $s_i = f(s_{i-1}, a_i)$ for i=1, 2, ..., k.



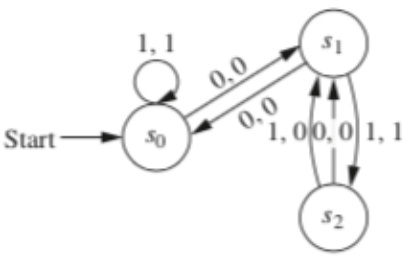
6.14. Find the output string generated by the Moore machine in Exercise 6.12 with each of these input strings. **a)** 0101 **b)** 111111 **c)** 11101110111

6.15. Find the output string generated by the Moore machine in Exercise 6.13 with each of the input strings in Exercise 6.14.

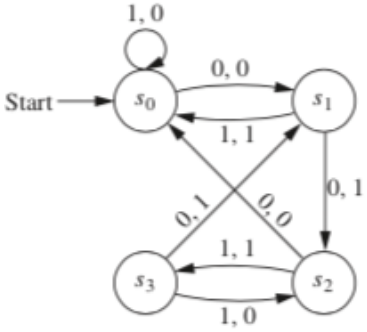
6.16. Construct a Moore machine that determines whether an input string contains an even or odd number of 1s. The machine should give 1 as output if an even number of 1s are in the string and 0 as output if an odd number of 1s are in the string.

Answers (SET 6).

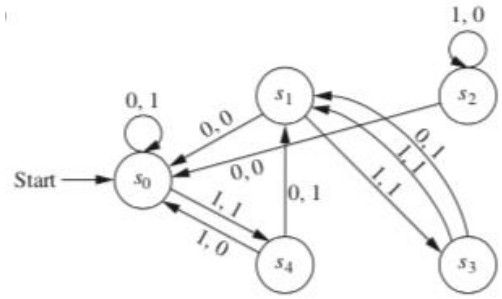
A6.1. a)



A6.1. b)



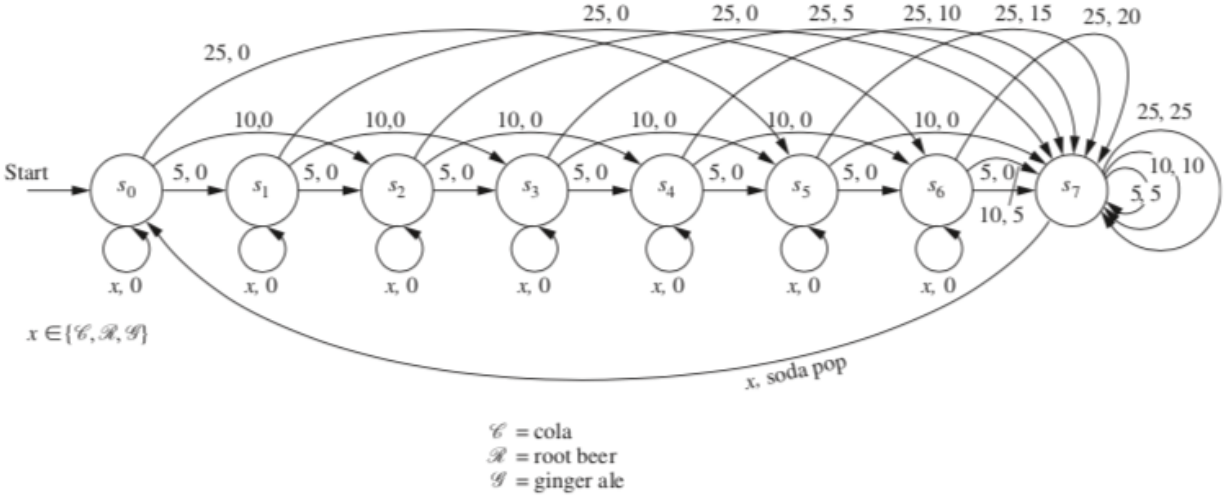
A6.1. c)



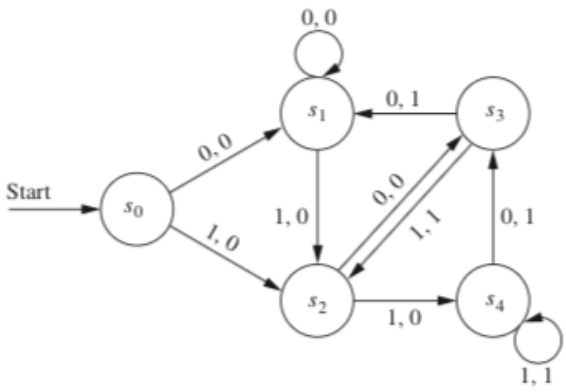
A6.2. a) 01010 **b)** 01000 **c)** 11011

A6.3. a) 1100 **b)** 00110110 **c)** 11111111111

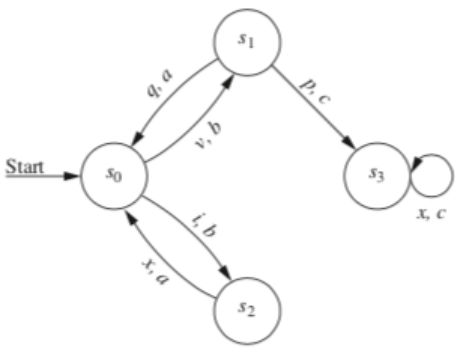
A6.5.



A6.6.

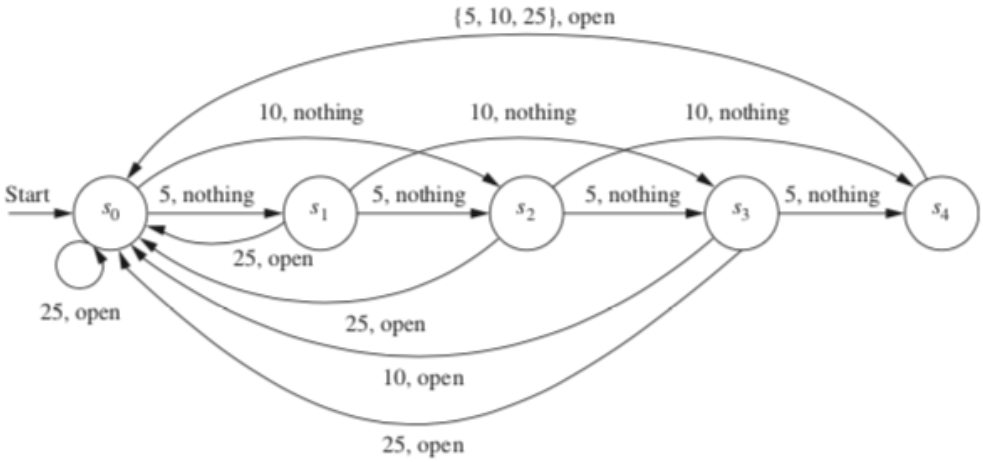


A6.7.



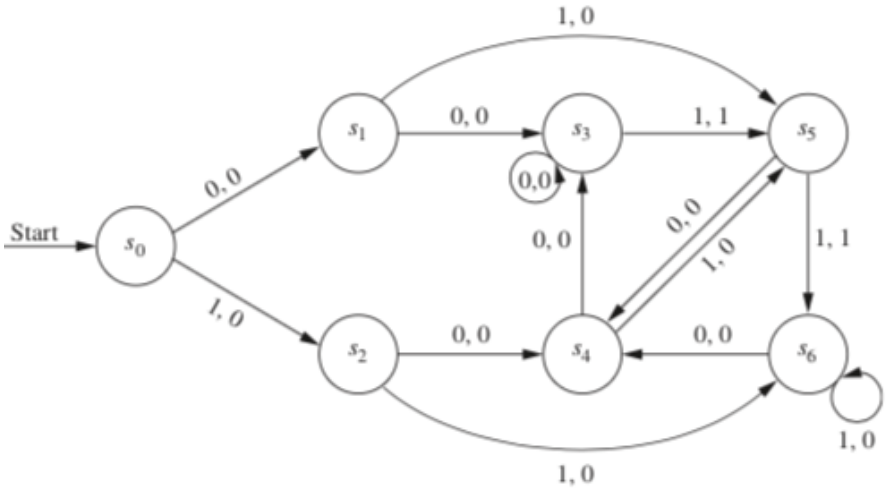
v = Valid ID
i = Invalid ID
p = Valid password
q = Invalid password
a = "Enter user ID"
b = "Enter password"
c = Prompt
x = Any input

A6.8.

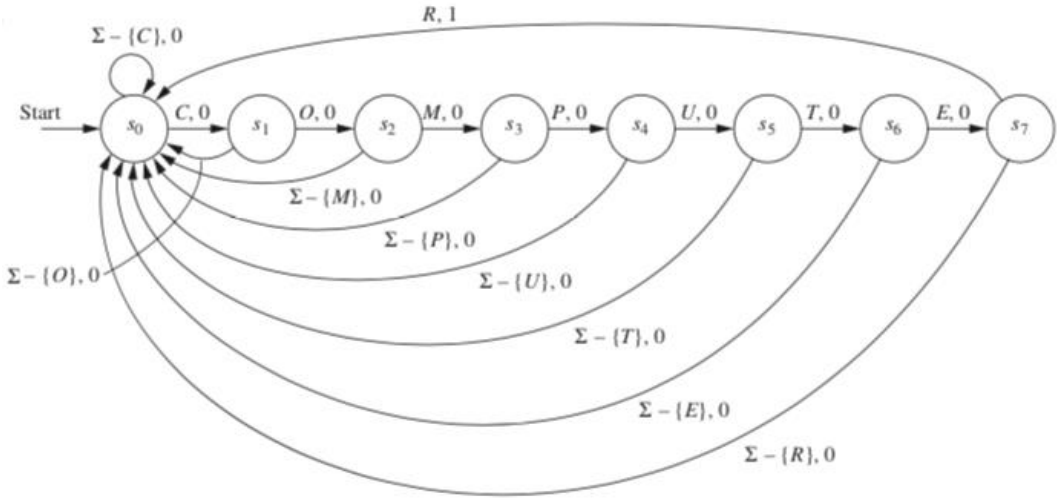


A6.9. Let s_0 be the start state and let s_1 be the state representing a successful call. From s_0, inputs of 2, 3, 4, 5, 6, 7, or 8 send the machine back to s_0 with output of an error message for the user. From s_0 an input of 0 sends the machine to state s_1, with the output being that the 0 is sent to the network. From s_0 an input of 9 sends the machine to state s_2 with no output; from there an input of 1 sends the machine to state s_3 with no output; from there an input of 1 sends the machine to state s_1 with the output being that the 911 is sent to the network. All other inputs while in states s_2 or s_3 send the machine back to s_0 with output of an error message for the user. From s_0 an input of 1 sends the machine to state s_4 with no output; from s_4 an input of 2 sends the machine to state s_5 with no output; and this path continues in a similar manner to the 911 path, looking next for 1, then 2, then any seven digits, at which point the machine goes to state s_1 with the output being that the ten-digit input is sent to the network. Any "incorrect" input while in states s_5 or s_6 (that is, anything except a 1 while in s_5 or a 2 while in s_6) sends the machine back to s_0 with output of an error message for the user. Similarly, from s_4 an input of 8 followed by appropriate successors drives us eventually to s_1, but inappropriate outputs drive us back to s_0 with an error message. Also, inputs while in state s_4 other than 2 or 8 send the machine back to state s_0 with output of an error message for the user.

A6.10.



A6.11.



A6.13.

State	f		g
	Input		
	0	1	
s_0	s_1	s_2	1
s_1	s_1	s_0	1
s_2	s_1	s_2	0

A6.15 **a)** 11111 **b)** 1000000 **c)** 100011001100

A6.16