10

Capabilities, Minimization, and Transformation of Sequential Machines

This chapter extends some of the concepts introduced in Chap. 9 and presents important techniques for the synthesis of sequential machines and for other problems considered in later chapters. The first two sections are concerned with the general finite-state model, its definition, capabilities, and limitations. The last two sections are concerned with the minimization of completely, as well as incompletely, specified machines.

10-1 THE FINITE-STATE MODEL—FURTHER DEFINITIONS

Our attention will be primarily focused on *deterministic machines*, which possess the property that the next state $S(t + 1)$ is determined uniquely by the present state $S(t)$ and the present input $x(t)$. Thus

$$S(t + 1) = \delta\{S(t),x(t)\} \qquad (10\text{-}1)$$

where δ is called the *state transition function*. The value of the output $z(t)$ is, in the most general case, a function of the present state $S(t)$ and

the inputs $x(t)$, i.e.,

$$z(t) = \lambda\{S(t),x(t)\}$$

where λ is called the *output functi*
(10-1) and (10-2) is generally kı
machine, known as the *Moore m*
function of only the present stat
input. In this case

$$z(t) = \lambda\{S(t)\}$$

Thus we arrive at the following for

Definition 10-1 A *synchronous seq*

$$M = (I,O,S,\delta,\lambda)$$

where I, O, and S are finite, nonem
respectively;
$\delta: I \times S \rightarrow S$ is the state tr
λ is the output function suc
$\lambda: I \times S \rightarrow O$ for Mealy ma
$\lambda: S \rightarrow O$ for Moore machin

The cartesian product $I \times S$
ments (I_i,S_j). The state transitio
(I_i,S_j) an element S_k from S, calle
the output function λ associates
from O, while in a Moore machin
states and the outputs.

Input-Output transformations

Consider machine M whose state
four-state machine with one input
which

$$S = \{A,B,C,D\} \qquad I = \{0,1$$

Suppose that the initial state
110; the machine will proceed th
state A, while producing the outp
state A, machine M *transforms* th
larly, for the same initial state, th
into 00010. Since every compute
input-to-output sequences, a finite-

the inputs $x(t)$, i.e.,

$$z(t) = \lambda\{S(t),x(t)\} \tag{10-2}$$

where λ is called the *output function*. A machine possessing properties (10-1) and (10-2) is generally known as a *Mealy machine*. Another machine, known as the *Moore machine*, results when the output is a function of only the present state and is independent of the external input. In this case

$$z(t) = \lambda\{S(t)\} \tag{10-3}$$

Thus we arrive at the following formal definition of a sequential machine.

Definition 10-1 A *synchronous sequential machine* M is a quintuple

$$M = (I,O,S,\delta,\lambda)$$

where I, O, and S are finite, nonempty sets of inputs, outputs, and states, respectively;

$\delta: I \times S \rightarrow S$ is the state transition function;
λ is the output function such that
$\lambda: I \times S \rightarrow O$ for Mealy machines;
$\lambda: S \rightarrow O$ for Moore machines.

The cartesian product $I \times S$ is the set containing all pairs of elements (I_i,S_j). The state transition function δ associates with each pair (I_i,S_j) an element S_k from S, called the *next state*. In a Mealy machine the output function λ associates with each pair (I_i,S_j) an element O_k from O, while in a Moore machine a correspondence exists between the states and the outputs.

Input-Output transformations

Consider machine M whose state diagram is given in Fig. 10-1. It is a four-state machine with one input variable and one output variable for which

$$S = \{A,B,C,D\} \qquad I = \{0,1\} \qquad O = \{0,1\}$$

Suppose that the initial state of M is A and the input sequence is 110; the machine will proceed through states B and C and return to state A, while producing the output sequence 001. Thus, for an initial state A, machine M *transforms* the input sequence 110 into 001. Similarly, for the same initial state, the input sequence 01100 is transformed into 00010. Since every computation involves some transformation of input-to-output sequences, a finite-state machine is capable of performing

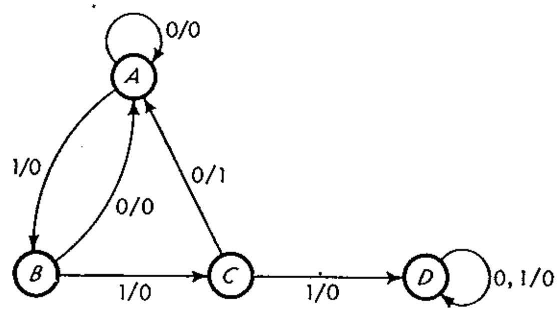

Fig. 10-1 State diagram of machine M.

a variety of computations and of solving a number of problems that can be expressed as transformation of sequences.

An important function of a sequential machine is to determine whether a given input sequence is a member of some prespecified set of sequences. The machine accomplishes this function by accepting those sequences which are members of the set, and rejecting the ones which are not. A machine, when started in its initial state, *accepts* an input sequence by producing an output 1 as it receives the last symbol of that sequence. Thus machine M accepts the sequences 110 and 0110 and rejects the sequence 01100, since its corresponding last output symbol is 0. The sequence detector of Fig. 9-8 can also be described as a machine which accepts those input sequences that are members of the set {all sequences whose last four symbols are 0101}.

The general problem of characterizing the machine's behavior by observing its input-output transformations is quite complex. Clearly, it is impractical to feed a machine with all possible input sequences in order to decide which ones it accepts. The problem increases in complexity if we wish to determine whether two arbitrary machines are related, in the sense that one machine accepts all the sequences accepted by the other. In this chapter we shall present finite experiments to determine the characteristics, capabilities, and limitations of a machine and the relations between machines. These subjects are further developed in Chaps. 13, 14, and 16.

Returning to the state diagram of M, we note that the application of an input 1 to M, when initially in state A, causes a transition to state B. We thus say that B is the 1-successor of A. In general, if an input sequence X takes a machine from state S_i to state S_j, then S_j is said to be the X-*successor* of S_i. For example, state D is the 111-successor of A. If M is known to be initially in either B or C, the 10-successor will be either state A or D. We say that (AD) is the 10-successor of (BC) to mean that A is the 10-successor of B and D of C.

It is evident that no input sequence exists which can take M out of

state D, and thus D is said t
called a *terminal state* if eith
sponding vertex in the state
arcs which emanate from it t
sponding vertex is a *source*; i.e
terminate in it.

A source state is clearl
similarly, no state is accessi
examples of situations which
machine. In other cases cer
from other subsets of states,
terminal state. If for every
exists an input sequence whic
strongly connected. Clearly,
states is not strongly connect

10-2 CAPABILITIES AND LIN

At this point, after having e
synthesis procedures for finit
some basic questions regardin
can a machine do? Are there
transformations that can be p
are imposed on the capabilit
number of its states? Altho
be deferred to Chap. 16, we
solvable by any finite-state
transformations that are real

Let the input to an n-st
of 1's. In response, the mac
state, through a succession
state transitions. Now, if w
it is longer than n, the mach
previously been in. And co
the input remains the same,
repeating fashion. Clearly,
exceed n, and could be smal
output reaches its periodic p
n. The preceding result can
consisting of a string of repea
will become periodic after a

This conclusion leads t
limitations of finite-state m

state D, and thus D is said to be a terminal state. Generally, a state is called a *terminal state* if either of the following is true: (1) The corresponding vertex in the state diagram is a *sink* vertex; i.e., no outgoing arcs which emanate from it terminate in other vertices. (2) The corresponding vertex is a *source*; i.e., no arcs which emanate from other vertices terminate in it.

A source state is clearly not accessible from any other state, and similarly, no state is accessible from a sink state. These are extreme examples of situations which limit the state transitions in a sequential machine. In other cases certain subsets of states may not be reachable from other subsets of states, even if the machine does not contain any terminal state. If for every pair of states S_i, S_j of a machine M there exists an input sequence which takes M from S_i to S_j, then M is said to be *strongly connected*. Clearly, any nontrivial machine which has terminal states is not strongly connected.

10-2 CAPABILITIES AND LIMITATIONS OF FINITE-STATE MACHINES

At this point, after having established several behavioral properties and synthesis procedures for finite-state machines, we turn our attention to some basic questions regarding the capabilities of these machines. What can a machine do? Are there any limitations on the type of input-output transformations that can be performed by a machine? What restrictions are imposed on the capabilities of the machine by the finiteness of the number of its states? Although a precise answer to these questions will be deferred to Chap. 16, we will point out the existence of problems not solvable by any finite-state machine and determine a characteristic of transformations that are realizable by such machines.

Let the input to an n-state machine be an arbitrarily long sequence of 1's. In response, the machine will progress, starting from some initial state, through a succession of states, in accordance with its specified state transitions. Now, if we let the sequence be long enough so that it is longer than n, the machine must eventually arrive at a state it has previously been in. And consequently, from this point on, and because the input remains the same, the machine must continue in a periodically repeating fashion. Clearly, for an n-state machine, the period cannot exceed n, and could be smaller. Moreover, the transient time until the output reaches its periodic pattern cannot exceed the number of states n. The preceding result can easily be generalized to any arbitrary input consisting of a string of repeated symbols. In every such case the output will become periodic after a transient time no longer than n.

This conclusion leads to many interesting results which exhibit the limitations of finite-state machines. For example, suppose we want to design a machine which receives a long sequence of 1's and is to produce an output 1 when and only when the number of inputs that it has received so far is equal to $k(k+1)/2$, for $k = 1, 2, 3, \ldots$. That is, the desired input-output transformation has the form

$$\begin{array}{rl} \text{Input} = & 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ \cdots \\ \text{Output} = & 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ \cdots \end{array}$$

Clearly, since the output does not become eventually periodic, no finite-state machine can produce such an infinite sequence.

In Sec. 9-1 we designed a serial adder which is capable of adding serially two binary numbers of arbitrary length. As another example, demonstrating the limitations on the capabilities of finite-state machines, we shall show that the serial-multiplication problem is not solvable by a *fixed* finite-state machine; i.e., *no finite-state machine with a fixed number of states can multiply two arbitrarily large binary numbers.*

To prove the foregoing assertion, suppose that there exists an n-state machine capable of serially multiplying any two binary numbers. Let us select as the two numbers to be multiplied $2^p \times 2^p = 2^{2p}$, where $p > n$. The inputs are fed serially into the machine, least significant digits first. 2^p is represented by a 1 followed by p 0's, and 2^{2p} is to be represented by a 1 followed by $2p$ 0's. The inputs are fed into the machine during the first $p + 1$ time units, i.e., between t_1 and t_{p+1}, as shown below. During this period the machine produces 0's. At t_{p+1} the input stops, while the machine must go on producing p additional 0's followed by a 1.

t_{2p+1}	t_{2p}	$\cdots$	t_{p+1}	t_p	$\cdots$	t_2	t_1	= time
			1	0	$\cdots$	0	0	= first number
			1	0	$\cdots$	0	0	= second number
1	0	$\cdots$	0	0	$\cdots$	0	0	= product

At this time period between t_{p+1} and t_{2p} the machine receives no input, but since $p > n$, it must have been during that time twice at one of the states. Following the same line of argument pursued earlier, we are led to the conclusion that its output must be periodic and the period is smaller than p. And therefore the machine will never produce the required 1 output.

Note that, for any two finite numbers, we can find a machine which is capable of multiplying them. However, the preceding result demonstrates that, for every finite-state machine capable of performing serial multiplication, we can find such numbers which it could not multiply. The reason for this limitation stems from the limited "memory" available to the machine. While in performing addition it had only to store information regarding a single-digit carry, in the multiplication problem it must be able to store arbitrarily large partial products.

In a similar manner we can show that no finite-state machine with a fixed number of states can perform, for arbitrarily large size blocks, the computation executed by the Turing machine of Sec. 9-5.

A more general and precise study of the capabilities and limitations of finite-state machines must be deferred to Chap. 16, where they will be defined in terms of regular expressions.

10-3 STATE EQUIVALENCE AND MACHINE MINIMIZATION

In constructing the state diagram (or table) of a finite-state machine, it often happens that the diagram contains redundant states, i.e., states whose functions can be accomplished by other states. The number of memory elements required for a realization of the machine is directly related to the number of states. (Recall that, for an n-state machine, $k = [\log_2 n]$ state variables are needed for an assignment.) Consequently, the minimization of the number of states does in many cases reduce the complexity and cost of the realization. Moreover, the diagnosis of sequential machines, which is studied in Chap. 13, is considerably simpler when the machine does not contain redundant states. It is therefore desirable to develop techniques for transforming a given machine into another machine which has no redundant states, so that both have the same terminal behavior.

k-equivalence

Two states, S_i and S_j, of machine M are *distinguishable* if and only if there exists at least one finite input sequence which, when applied to M, causes different output sequences, depending on whether S_i or S_j is the initial state. The sequence which distinguishes these states is called a *distinguishing sequence of the pair* (S_i,S_j). If there is uncertainty as to whether the state of M is S_i or S_j, the application of the corresponding distinguishing sequence yields an output sequence which is sufficient to determine uniquely the unknown state. If there exists for pair (S_i,S_j) a distinguishing sequence of length k, the states in (S_i,S_j) are said to be *k-distinguishable*.

As an example consider the pair (A,B) of machine M_1, whose state table is shown in Table 10-1. The pair (A,B) is 1-distinguishable, since a 1 input applied to M_1 when initially in A yields an output 1, versus an output 0 when it is initially in B. On the other hand, the pair (A,E) is 3-distinguishable, since there is no input sequence of length 2 which distinguishes A from E. The only sequence of length 3 which is a distinguishing sequence for the pair (A,E) is $X = 111$, and the output sequences corresponding to initial states A and E are 100 and 101, respectively. Note that 1101 is also a sequence which distinguishes A from E, although it is not the shortest such sequence. An all-zero sequence, on the other hand, will produce identical output sequences independently of whether the initial state is A or E.

The concept of k-distinguishability leads directly to the definition of k-equivalence and equivalence. States that are not k-distinguishable are said to be *k-equivalent.* For example, states A and E of M_1 are 2-equivalent. States which are k-equivalent are also r-equivalent, for all $r < k$. States that are k-equivalent for all k are said to be *equivalent.* Thus we arrive at the following definition.

Definition 10-2 States S_i and S_j of machine M are said to be *equivalent* if and only if, *for every possible input sequence,* the same output sequence will be produced regardless of whether S_i or S_j is the initial state.

Thus S_i and S_j are equivalent (indicated by $S_i = S_j$) if there is no input sequence which distinguishes them. It will be subsequently shown (see Theorem 10-2) that states that are k-equivalent for all $k \leq n - 1$ are equivalent. Clearly, if $S_i = S_j$ and $S_j = S_k$, then $S_i = S_k$. It therefore follows (see Sec. 2-2) that state equivalence is an equivalence relation, and in consequence of this characteristic, the set of states of the machine can be partitioned into disjoint subsets, known as the *equivalence classes,* so that two states are in the same equivalence class if and only if they are equivalent, and are in different classes if and only if they are distinguishable. Definition 10-2 can be generalized to the case where S_i is a possible initial state in machine M_1, while S_j is an initial state in machine M_2, where both M_1 and M_2 have the same input alphabet.

The procedure of determining the sets of equivalent states in a machine, i.e., the equivalence classes, ensues from the following property. *If S_i and S_j are equivalent states, their corresponding X-successors, for all X, are also equivalent,* since otherwise it would be trivial to construct a distinguishing sequence for (S_i,S_j) by first applying an input sequence that transfers the machine to the distinguishable successors of S_i and S_j.

The minimization procedure

The object of this section is to describe a procedure for determining the sets of equivalent states of a specified machine M. The result sought is a partition on the states of M such that two states are in the same block if and only if they are equivalent.

The first step is to partition the states of M into subsets such that

Table 10-1 Machine M_1

PS	NS, z $x=0$	$x=1$
A	$E,0$	$D,1$
B	$F,0$	$D,0$
C	$E,0$	$B,1$
D	$F,0$	$B,0$
E	$C,0$	$F,1$
F	$B,0$	$C,0$

$P_0 = (ABCDEF)$
$P_1 = (ACE)(BDF)$
$P_2 = (ACE)(BD)(F)$
$P_3 = (AC)(E)(BD)(F)$
$P_4 = (AC)(E)(BD)(F)$

all states in the same subset are 1-equivalent. This is accomplished by placing states having identical outputs under all possible inputs in the same subset. Clearly, two states which are in different subsets are 1-distinguishable. As an example, consider machine M_1 given in Table 10-1. The first partition P_0 corresponds to 0-distinguishability, and it defines our initial "ignorance" regarding the response of the various states, prior to the application of any input. P_1 is obtained simply by inspecting the table and placing those states having the same outputs, under all inputs, in the same block. Thus A, C, and E are in the same block, since their outputs under 0 and 1 inputs are 0 and 1, respectively. A similar argument places B, D, and F in the other block. Clearly, P_1 establishes the sets of states which are 1-equivalent.

The next step is to obtain the partition P_2 whose blocks consist of the sets of states which are 2-equivalent, that is, equivalent under any input sequence of length 2. This is accomplished by observing that two states are 2-equivalent if and only if they are 1-equivalent and their I_i-successors, for all possible I_i, are also 1-equivalent. Consequently, two states are placed in the same block of P_2 if and only if they are in the same block of P_1, and for each possible I_i their I_i-successors are also contained in a block of P_1. This step is carried out by splitting blocks of P_1 whenever their successors are not contained in a common block of P_1. The 0- and 1-successors of (ACE) are (CE) and (BDF), respectively, and since both are contained in common blocks of P_1, the states in (ACE) are 2-equivalent, and therefore (ACE) constitutes a block in P_2. The 1-successor of (BDF) is (DBC), but since (DB) and (C) are not contained in a single block of P_1, the block (BDF) must be split into (BD) and (F), so that the successors of the blocks in the refined† partition are 1-equivalent. In a similar manner P_3 is obtained by splitting the block (ACE) of P_2 into (AC) and (E), since the 1-successors of A, C, and E are D, B, and F, which are not 2-equivalent.

† A partition P is said to be a *refinement* of a partition Q if P is smaller than Q.

In general, the P_{k+1} partition is obtained from P_k by placing in the same block of P_{k+1} those states which are in the same block of P_k *and* whose I_i-successors for every possible I_i are also in a common block of P_k. This process places in the same block the states that are $(k+1)$-equivalent, and in different block states that are $(k+1)$-distinguishable. Note that no state can belong to more than one block, since this would make it distinguishable with respect to itself.

If for some k, $P_{k+1} = P_k$, the process terminates and P_k defines the sets of equivalent states of the machine; that is, all states contained in the same block of P_k are equivalent, while states belonging to different blocks are distinguishable. P_k is thus called the *equivalence partition*, and the foregoing procedure is referred to as the *Moore reduction procedure*. For machine M_1, P_3 is the equivalence partition, and therefore states A and C are equivalent, and so are B and D. Before proceeding with the minimization, we shall prove two theorems to establish its validity and determine its length.

Theorem 10-1 *The equivalence partition is unique.*

Proof: Suppose there exist two equivalence partitions, P_a and P_b, and that $P_a \neq P_b$. Then there exist two states, S_i and S_j, which are in the same block of one partition and are not in the same block of the other. Since S_i and S_j are in different blocks of (say) P_b, there exists at least one input sequence which distinguishes S_i from S_j, and therefore they cannot be in the same block of P_a. ∎

Theorem 10-2 *If two states, S_i and S_j, of machine M are distinguishable, then they are distinguishable by a sequence of length $n-1$ or less, where n is the number of states in M.*

Proof: P_1 contains at least two blocks; otherwise M is reducible to a combinational circuit which has only a single state. At each step, the partition P_{k+1} is smaller than or equal to P_k. (Recall that a partition $P_i \leq P_j$ if every block of P_i is contained in a block of P_j, e.g., P_2 of M_1 is smaller than P_1.) If P_{k+1} is smaller than P_k, then it contains at least one more block than P_k. But since the number of blocks is limited to n, at most $n-1$ partitions can be generated in the reduction procedure, and thus, if S_i and S_j are distinguishable, they are distinguishable by a sequence of length $n-1$ or less. ∎

Machine equivalence

Before proceeding with the determination of the minimal machine equivalent to M_1, we shall define precisely what we mean by equivalent and minimal machines.

Definition 10-3 Two machi
only if, for every state in M
in M_2, and vice versa.

The equivalence partit
number of blocks in the equ
minimum number of states t
The machine which contain
is called the *minimal*, or *red*

If we denote the block
β, γ, and δ, corresponding,
obtain machine M_1^* (Table
1-successor of α to be γ, sinc
In this manner M_1^* is spec
response of M_1, and therefo
generated by the equivalen

Ta

Example We shall further i
it to machine M_2 (Tal
blocks of the equivaler
and the reduced mach

Table 10-3 Mach

PS	NS, z
	$x = 0$
A	E,0
B	C,0
C	B,0
D	G,0
E	F,1
F	E,0
G	D,0

Definition 10-3 Two machines, M_1 and M_2, are said to be *equivalent* if and only if, for every state in M_1, there is a corresponding equivalent state in M_2, and vice versa.

The equivalence partition has been shown to be unique. Thus the number of blocks in the equivalence partition of a machine M defines the *minimum* number of states that any machine equivalent to M must have. The machine which contains no equivalent states and is equivalent to M is called the *minimal*, or *reduced*, form of M.

If we denote the blocks of the equivalence partition P_3 of M_1 by α, β, γ, and δ, corresponding, respectively, to (AC), (E), (BD), and (F), we obtain machine M_1^* (Table 10-2). In constructing M_1^* we specify the 1-successor of α to be γ, since the 1-successor of (AC) is (BD), and so on. In this manner M_1^* is specified to duplicate the state transitions and response of M_1, and therefore is equivalent to it. And since it has been generated by the equivalence partition of M_1, it is its minimal form.

Table 10-2 Machine M_1^*

PS	NS, z: $x = 0$	NS, z: $x = 1$
α	β,0	γ,1
β	α,0	δ,1
γ	δ,0	γ,0
δ	γ,0	α,0

Example We shall further illustrate the reduction procedure by applying it to machine M_2 (Table 10-3) and finding its minimal form. The blocks of the equivalence partition P_4 are denoted by $\alpha, \beta, \ldots, \epsilon$, and the reduced machine M_2^* (Table 10-4) results. ■

Table 10-3 Machine M_2

PS	NS, z: $x = 0$	NS, z: $x = 1$
A	E,0	C,0
B	C,0	A,0
C	B,0	G,0
D	G,0	A,0
E	F,1	B,0
F	E,0	D,0
G	D,0	G,0

$P_0 = (ABCDEFG)$
$P_1 = (ABCDFG)(E)$
$P_2 = (AF)(BCDG)(E)$
$P_3 = (AF)(BD)(CG)(E)$
$P_4 = (A)(F)(BD)(CG)(E)$
$P_5 = (A)(F)(BD)(CG)(E)$

Table 10-4 Machine M_2^*

PS	NS, z $x = 0$	$x = 1$
$(A) \to \alpha$	$\epsilon,0$	$\delta,0$
$(F) \to \beta$	$\epsilon,0$	$\gamma,0$
$(BD) \to \gamma$	$\delta,0$	$\alpha,0$
$(CG) \to \delta$	$\gamma,0$	$\delta,0$
$(E) \to \epsilon$	$\beta,1$	$\gamma,0$

The selection of labels α, β, . . . , assigned to the blocks of P_4, is obviously arbitrary. A different assignment of labels would certainly have described a machine with the same behavioral properties. In general, if one machine can be obtained from the other by relabeling its states, they are said to be *isomorphic* to each other. The foregoing results lead to the following basic conclusion:

▶ To every machine M there corresponds a minimal machine M^* which is equivalent to M and is unique up to isomorphism.

The detection of isomorphism is not always easy and is best accomplished by using a canonical representation for the machine. Such a representation is obtained by selecting a state (preferably the starting state if specified) and labeling it A. The next labels are selected in such a way that when successive rows of the table, starting in A and going down through B, C, etc., are read from left to right, the first occurrence of each new label will be in alphabetical order. Whenever a machine is given in this canonical representation, it is said to be in *standard form*. Clearly, when the starting state of a reduced machine is specified, its standard form is unique.

The transformation of a machine into its standard form will be illustrated by means of M_2^*. Denoting α by A implies that its 0-successor ϵ must be denoted B, because it is the first occurrence of a new label. Similarly, its 1-successor δ must be denoted C. Row B (i.e., ϵ) must be relabeled next; its first entry is β, and since it is a new label, it is denoted D. Similarly, γ is denoted E, and the standard form of Table 10-5 results.

The detection of isomorphism when the starting states are not specified is in general not as simple. When the number of states, however, is not too large, isomorphism can be detected by inspecting the state diagrams of the machines. The necessary and sufficient condition

for two machines to be isom
grams be identical, except fo

Tabl

P
$\alpha \to$
$\epsilon \to$
$\delta \to$
$\beta \to$
$\gamma \to$

10-4 SIMPLIFICATION OF IN

In practice, it often occurs
inputs are not possible. For
in state A, will never receive a
transition and its associated
situations the state transition
binations of states and input
thus are left unspecified. S
specified; the determination of
them are the subject of this s

Whenever a state trans
the machine may become un
tion we shall assume that th
when in any of its possible s
next state is encountered, e
input sequence is said to be *ap*
that all outputs encountered
applicable to S_i. The next
possibly for the last symbol
Actually, the specified b
transitions can be described
are completely specified. Th
ing all the dashes in the nex
state T whose outputs are
machine M_3 shown in Table
described by Table 10-7, in w
only the outputs are partiall

for two machines to be isomorphic to each other is that their state diagrams be identical, except for the labeling of their vertices.

Table 10-5 Standard form for M_2^*

PS	NS, z $x = 0$	$x = 1$
$\alpha \to A$	$B,0$	$C,0$
$\epsilon \to B$	$D,1$	$E,0$
$\delta \to C$	$E,0$	$C,0$
$\beta \to D$	$B,0$	$E,0$
$\gamma \to E$	$C,0$	$A,0$

10-4 SIMPLIFICATION OF INCOMPLETELY SPECIFIED MACHINES

In practice, it often occurs that various combinations of states and inputs are not possible. For example, the machine of Table 9-15, when in state A, will never receive a 0 input, and consequently the corresponding transition and its associated output may be left unspecified. In other situations the state transitions are completely defined, but for some combinations of states and inputs the output values may not be critical, and thus are left unspecified. Such machines are said to be *incompletely specified;* the determination of their properties and methods for simplifying them are the subject of this section.

Whenever a state transition is unspecified, the future behavior of the machine may become unpredictable. In order to avoid such a situation we shall assume that the input sequences applied to the machine, when in any of its possible starting states, are such that no unspecified next state is encountered, except possibly at the final step. Such an input sequence is said to be *applicable* to the starting state S_i of M. Note that all outputs encountered need not be specified for a sequence to be applicable to S_i. The next states, however, must be specified, except possibly for the last symbol of the sequence.

Actually, the specified behavior of a machine with partially specified transitions can be described by another machine whose state transitions are completely specified. This transformation is accomplished by replacing all the dashes in the next-state entries by T and adding a terminal state T whose outputs are unspecified. As an illustration, consider machine M_3 shown in Table 10-6. The specified behavior of M_3 can be described by Table 10-7, in which all state transitions are specified, and only the outputs are partially defined.

Table 10-6 Machine M_3 with unspecified transitions

PS	NS, z $x = 0$	NS, z $x = 1$
A	B,1	—
B	—,0	C,0
C	A,1	B,0

Table 10-7 An equivalent description where all transitions are specified

PS	NS, z $x = 0$	NS, z $x = 1$
A	B,1	T,—
B	T,0	C,0
C	A,1	B,0
T	T,—	T,—

Compatible states

In Sec. 10-3 we defined state and machine equivalence. We shall find it useful to generalize these concepts as follows.

Definition 10-4 State S_i of M_1 is said to *cover, or contain*, state S_j of M_2 if and only if every input sequence applicable to S_j is also applicable to S_i, and its application to both M_1 and M_2 when they are initially in S_i and S_j, respectively, results in identical output sequences whenever the outputs of M_2 are specified.

The covering concept can be extended to machines as follows: *Machine M_1 is said to cover machine M_2 if and only if, for every state S_j in M_2, there is a corresponding state S_i in M_1 such that S_i covers S_j.* Clearly, the machine specified by Table 10-6 is covered by that of Table 10-7. If state S_i of machine M covers another state S_j of the same machine, then only S_i must be retained, while S_j may be deleted.

Definition 10-5 Two states, S_i and S_j, of machine M are *compatible* if and only if, for every input sequence applicable to both S_i and S_j, the same output sequence will be produced *whenever both outputs are specified* and regardless of whether S_i or S_j is the initial state.

Hence S_i and S_j are compatible if and only if their outputs are not conflicting (i.e., identical when specified) and their I_i-successors, for every I_i for which both are specified, are either the same or also compatible. In general, three or more states, S_i, S_j, S_k, . . . , are compatible if and only if, for every applicable input sequence, no two conflicting output sequences will be produced, without regard as to which of the above states is the initial state. Thus a set of states $(S_i, S_j, S_k, \ldots)$ is called a *compatible* if all its members are compatible.

A compatible C_i is said
patible C_j if and only if ever
C_i. A compatible is *maximal*
(Note that a single state that
maximal compatible.) Thus
patibles, that in effect is equiv
subset of a compatible is also
Generalizing slightly,
specified machines, the analog
is the compatibility relation.
these two relations will be po

The nonuniqueness of the red

Before developing the simplifi
machines, we shall illustrate s
ing the minimization proced
Table 10-8.

Tabl

PS
A
B
C
D
E

The dashes in row A, col
outputs associated with these
specified according to our co
1's, we find that states A an
and corresponding successors
bine these states by redirectin
to B. The resulting simplif
reduced form, and thus cann
choose to specify the dashes a
and E are equivalent, and in
lent. Thus we may relabel
respectively, and the minimal

A compatible C_i is said to be *larger* than, or to *cover*, another compatible C_j if and only if every state contained in C_j is also contained in C_i. A compatible is *maximal* if it is not covered by any other compatible. (Note that a single state that is not compatible with any other state is a maximal compatible.) Thus, if we find the set of all the maximal compatibles, that in effect is equivalent to finding all compatibles, since every subset of a compatible is also a compatible.

Generalizing slightly, we find that in the case of incompletely specified machines, the analog to the equivalence relation studied earlier is the compatibility relation. The similarities and differences between these two relations will be pointed out subsequently.

The nonuniqueness of the reduced and minimal machines

Before developing the simplification procedure for incompletely specified machines, we shall illustrate some of the difficulties encountered in applying the minimization procedure of Sec. 10-3 to machine M_4 shown in Table 10-8.

Table 10-8 Machine M_4

PS	NS, z $x = 0$	$x = 1$
A	$C,1$	$E,-$
B	$C,-$	$E,1$
C	$B,0$	$A,1$
D	$D,0$	$E,1$
E	$D,1$	$A,0$

The dashes in row A, column 1, and row B, column 0, mean that the outputs associated with these transitions will be ignored, and thus may be specified according to our convenience. If we replace both dashes by 1's, we find that states A and B become equivalent since their outputs and corresponding successors are identical. Consequently, we may combine these states by redirecting to A all the transitions presently leading to B. The resulting simplified machine, shown in Table 10-9, is in reduced form, and thus cannot be further simplified. If, however, we choose to specify the dashes as 0's, then it is easy to verify that states A and E are equivalent, and in addition states B, C, and D become equivalent. Thus we may relabel the blocks (AE) and (BCD) by α and β, respectively, and the minimal machine of Table 10-10 results.

Table 10-9 A simplified reduced machine, M_4^*

PS	NS, z: $x = 0$	NS, z: $x = 1$
A	$C,1$	$E,1$
C	$A,0$	$A,1$
D	$D,0$	$E,1$
E	$D,1$	$A,0$

Table 10-10 A minimal machine, $M_4{}^\dagger$

PS	NS, z: $x = 0$	NS, z: $x = 1$
$(AE) \to \alpha$	$\beta,1$	$\alpha,0$
$(BCD) \to \beta$	$\beta,0$	$\alpha,1$

From the foregoing example the following observations can be made. States A and B of M_4 are compatible, and if C and D are also compatible, so are A and E. But states B and E are 1-distinguishable, and therefore are incompatible. Consequently, since it is not transitive, the compatibility relation is not an equivalence relation. It thus follows that *a set of states is a compatible if and only if every pair of states in that set is compatible*. For example, states B, C, and D of M_4 form the compatible (BCD), since (BC), (BD), and (CD) are compatibles.

Both machines M_4^* and $M_4{}^\dagger$ cover M_4, and their numbers of states are each smaller than the number of states of M_4. Both are in reduced form; i.e., they contain no redundant states. This situation in which two different reduced machines cover a third one is evidently in contrast to Theorem 10-1. This poses serious difficulty in applying the previously derived minimization procedure, since we can no longer be content with finding a reduced machine covering the original one, and our aim must be to find a reduced machine which not only covers the original machine but also has a minimal number of states.

A further and very crucial difference between completely and incompletely specified machines is demonstrated by means of machine M_5 (Table 10-11). Because of the output entries, the only candidates for state equivalence are states A and B or B and C. And because of the next-state entries, A is equivalent to B only if B is equivalent to C. But for A and B to be equivalent, the dash must be replaced by a 0, while for B and C to be equivalent, the dash must be replaced by a 1. Evidently, there is no way of specifying the unspecified entry so as to achieve any state equivalence. However, a hasty conclusion that M_5 is in reduced form would be false, as is shown subsequently.

The augmented machine of Table 10-12 is obtained by a process known as *state splitting*. This process involves the replacement of a state S_i by two or more states, S_i', S_i'', . . . , such that each of the new states covers S_i. To ensure that the augmented machine covers the original one, it is necessary to modify the next-state entries, so that each

Table 10-11 Machine M_5

PS	NS, z: $x = 0$	NS, z: $x = 1$
A	$A,0$	$C,0$
B	$B,0$	$B,-$
C	$B,0$	$A,1$

Table 10-12 Augmented machine

PS	NS, z: $x = 0$	NS, z: $x = 1$
A	$A,0$	$C,0$
B'	$B',0$	$B'',-$
B''	$B^+,0$	$B',-$
C	$B^+,0$	$A,1$

transition to S_i is replaced by a transition to either S_i' or S_i'', etc. In our case, state B has been split into B' and B'', and the next-state entries modified as shown in Table 10-12, where the symbol B^+ means that the transition may be either B' or B''. Clearly, the augmented machine covers M_5 and is reducible to it by letting $B' = B'' = B$.

In general, since B' and B'' both cover B, we may specify the next-state entries B arbitrarily as B' or B''. If, however, we select the specification shown in Table 10-12, a simplification of M_5 becomes possible. States A and B' are compatible if their 1-successors C and B'' are. Similarly, states B'' and C are compatible if their 1-successors B' and A are. Thus, if we designate the compatibles (AB') and $(B''C)$ by α and β, respectively, we obtain the minimal machines of Table 10-13. The result is Table 10-13a or 10-13b, depending on whether B^+ is specified as B' or B''.

Table 10-13 Two minimal machines corresponding to M_5

PS	NS, z: $x = 0$	NS, z: $x = 1$
$(AB') \to \alpha$	$\alpha,0$	$\beta,0$
$(B''C) \to \beta$	$\alpha,0$	$\alpha,1$

(a) Setting $B^+ = B'$.

PS	NS, z: $x = 0$	NS, z: $x = 1$
$(AB') \to \alpha$	$\alpha,0$	$\beta,0$
$(B''C) \to \beta$	$\beta,0$	$\alpha,1$

(b) Setting $B^+ = B''$.

The foregoing example demonstrates the nonuniqueness of the minimal machine in the case of incompletely specified machines. The minimal machines of Table 10-13 were obtained by allowing state B to be split so that it can be made equivalent to both A and C (by specifying the unspecified output differently). This points out the main difference between completely and incompletely specified machines. *While the equivalence partition consists of disjoint blocks, the subsets of compatibles may be overlapping.*

The merger graph

In reducing machine M_4 we actually specified the don't-care entries, and thus transformed the incompletely specified machine into a completely specified one. Such a specification may not be the optimal one, and thus will drastically reduce our freedom in simplifying the machine. It is therefore desirable first to generate the entire set of compatibles, and then to select an appropriate subset, which will form the basis for a state reduction leading to a minimal machine.

Since a set of states is compatible if and only if every pair of states in that set is compatible, it is sufficient to consider only pairs of states and to use them to generate the entire set. We shall refer to a compatible pair of states as a *compatible pair*. Let the I_k-successors of S_i and S_j be S_p and S_q, respectively; then (S_pS_q) is said to be implied by (S_iS_j). For example, the compatible (CF) of machine M_6 (Table 10-14) is implied by

Table 10-14 Machine M_6

PS	I_1	I_2	I_3	I_4
	NS, z			
A	—	*C*,1	*E*,1	*B*,1
B	*E*,0	—	—	—
C	*F*,0	*F*,1	—	—
D	—	—	*B*,1	—
E	—	*F*,0	*A*,0	*D*,1
F	*C*,0	—	*B*,0	*C*,1

(AC), and so on. Thus, if (S_iS_j) is a compatible pair, then (S_pS_q) is referred to as its *implied pair*. In general, a set of states P is *implied* by a set of states Q if, for some input I_k, P is the set of all I_k-successors of the states in Q. The merger graph presented subsequently serves as the major tool in the determination of the set of all compatibles.

The *merger graph* of an n-state machine M is an undirected graph defined as follows:

1. It consists of n vertices, each of which corresponds to a state of M.
2. For each pair of states (S_iS_j) in M whose next-state and output entries are not conflicting, an undirected arc is drawn between vertices S_i and S_j.
3. If for a pair of states (S_iS_j) the corresponding outputs under all inputs are not conflicting, but the successors are not the same, an interrupted arc is drawn between S_i and S_j, and the implied pairs are entered in the space.

Consider machine M_6 (Table 10-14) and its merger graph, shown in Fig. 10-2. Since the next-state and output entries of states A and B are not conflicting, an arc is drawn between vertices A and B. States A and C, on the other hand, have nonconflicting outputs, but the successors under input I_2 are C and F. Therefore (AC) is a compatible only if (CF) is, and consequently an interrupted arc is drawn between vertices A and C, and (CF) is entered in the space. Similarly, (AD) is a compatible if and only if (BE) is, and thus (BE) is entered in the space of the interrupted arc drawn between A and D. On the other hand, no arc is drawn between A and E, since these states are incompatible, their outputs under I_2 and I_3 being conflicting. In a similar manner, every possible pair of states is checked, and the entire merger graph completed.

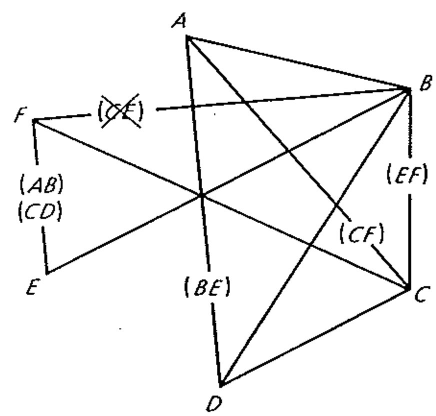

Fig. 10-2 Merger graph for machine M_6.

The merger graph displays all possible pairs of states and their implied pairs, and since a pair of states is compatible only if its implied pair is, it is now necessary to check and determine whether the implied pairs are indeed compatibles. A pair (S_pS_q) is incompatible if no arc is drawn between vertices S_p and S_q. In such a case, if (S_pS_q) is written in the space of an interrupted arc, the entry (S_pS_q) is crossed off, and the corresponding arc is ignored. For example, the condition for (BF) to be compatible is that (CE) be compatible, but since there is no arc drawn between C and E, (CE) is incompatible and the arc between B and F is ignored. Thus states B and F are incompatible. Next, it is necessary to check whether the incompatibility of (BF) does not invalidate any other implied pair, that is, if (BF) is not written in the space of another interrupted arc, and so on. The interrupted arcs which remain in the graph, after all the implied pairs have been verified to be compatible, are regarded as solid ones.

For machine M_6 the merger graph reveals the existence of nine compatible pairs:

$$(AB),(AC),(AD),(BC),(BD),(BE),(CD),(CF),(EF)$$

Moreover, since (AB), (AC), and (BC) are compatibles, then (ABC) is also a compatible, and so on. In this manner the entire set of compatibles of M_6 can be generated from its compatible pairs.

In order to find a minimal set of compatibles, which covers the original machine and can be used as a basis for the construction of a minimal machine, it is often useful to find the set of maximal compatibles. Recall that a compatible is maximal if it is not contained in any other compatible. In terms of the merger graph, we are looking for complete polygons which are not contained within any higher-order complete polygons. [A *complete polygon* is one in which all possible $(n - 3)n/2$ diagonals exist, where n is the number of sides in the polygon.] Since the states covered by a complete polygon are all pairwise compatible, they constitute a compatible; and if the polygon is not contained in any higher-order complete polygon, they constitute a maximal compatible.

In Fig. 10-2 the set of highest-order polygons are the tetragon $(ABCD)$ and the arcs (CF), (BE), and (EF). Generally, after a complete polygon of order n has been found, all polygons of order $n - 1$ contained in it can be ignored. Consequently, the triangles (ABC), (ACD), etc., are not considered. Thus the following set of maximal compatibles for machine M_6 results:

$$\{(ABCD),(BE),(CF),(EF)\}$$

The closed sets of compatibles

Consider the set of compatibles $\{(ABCD),(EF)\}$ of machine M_6. Since this is the minimal number of compatibles covering all the states of M_6, it defines a *lower bound* on the number of states in the minimal machine which covers M_6. But if we select the maximal compatible $(ABCD)$ to be a state in the reduced machine, its I_2- and I_3-successors, (CF) and (BE), respectively, must also be selected. However, since none of these compatible pairs is contained in the above set, the lower bound cannot be achieved, and the set of maximal compatibles $\{(ABCD),(EF)\}$ cannot be used to define the states of a minimal machine that covers M_6.

Definition 10-6 A set of compatibles (for machine M) is said to be *closed* if, for every compatible contained in the set, all its implied compatibles are also contained in the set. A closed set of compatibles which contains all the states of M is called a *closed covering.*

Example For M_6 the set $\{(AD),(BE),(CD)\}$ is closed. The set $\{(AB),$ $(CD),(EF)\}$ is a closed covering. ■

The closed covering serves, for incompletely specified machines, the same function that the equivalence partition serves for completely specified machines. It specifies the states which are compatible and which may be covered by a single state of a reduced machine. However, as demonstrated by the preceding examples, the closed covering is not unique, and our task is to select the one which has the minimum number of compatibles, and thus defines a minimal-state machine which covers the original one.

The set containing all the maximal compatibles is, clearly, a closed covering, since it covers all the states of the machine, and every implied compatible is contained in the set. Consequently, the set of maximal compatibles places an *upper bound* on the number of states in the machine which covers the original one. For machine M_6, this bound is four. It must be noted at this point that the upper bound is meaningless when the number of maximal compatibles is larger than the number of states in the original machine.

In the preceding discussion we showed that the bounds on the number of states in the minimal machine can be derived from the set of all the maximal compatibles. For machine M_6 these bounds were found to be two and four, but since the lower bound cannot be achieved, it becomes necessary to determine whether a closed covering containing three compatibles can be found. These compatibles need not necessarily be maximal; in fact, the maximal compatible $(ABCD)$ cannot be included in that set, since it implies the entire set of maximal compatibles.

An inspection of the merger graph of Fig. 10-2 reveals that states A and B can be covered by the compatible pair (AB), and similarly, states C and D can be covered by (CD); no pairs are implied by these compatibles, which thus form a closed set. In order to obtain the desired covering, all we need is a single compatible which covers state E and F. Fortunately, the pair (EF) is compatible, and it implies the pairs (AB) and (CD) which are contained in the above set. Consequently, the set $\{(AB),(CD),(EF)\}$ is a closed covering containing three compatibles, and thus yields a minimal three-state machine which covers M_6. This machine is shown in Table 10-15. In a similar manner we can show that

Table 10-15 A minimal machine covering M_6

PS	NS, z			
	I_1	I_2	I_3	I_4
$(AB) \to \alpha$	$\gamma,0$	$\beta,1$	$\gamma,1$	$\alpha,1$
$(CD) \to \beta$	$\gamma,0$	$\gamma,1$	$\alpha,1$	—
$(EF) \to \gamma$	$\beta,0$	$\gamma,0$	$\alpha,0$	$\beta,1$

the set $\{(AD),(BE),(CF)\}$ is also a closed covering which corresponds to another minimal machine containing M_6.

The preceding closed coverings have been obtained by inspecting the merger graph and employing a "trial-and-error" procedure. In the following section we shall discuss in detail a more systematic procedure for determining the minimal closed coverings; but it should be pointed out from the outset that no straightforward procedure is known as yet, and a certain amount of search is unavoidable.

The compatibility graph

Consider machine M_7 and its merger graph, shown in Table 10-16 and Fig. 10-3, respectively. The merger graph is constructed in the usual manner; since states A and B are incompatible, the arc between C and E is crossed off, and as a result (AE) and (BD) are also found to be incompatible. The set of maximal compatibles derived from the merger graph contains four members and is given by

$$\{(ACD),(BC),(BE),(DE)\}$$

Table 10-16 Machine M_7

PS	NS, z			
	I_1	I_2	I_3	I_4
A	—	—	$E,1$	—
B	$C,0$	$A,1$	$B,0$	—
C	$C,0$	$D,1$	—	$A,0$
D	—	$E,1$	$B,-$	—
E	$B,0$	—	$C,-$	$B,0$

The *compatibility graph* is a directed graph whose vertices correspond to all compatible pairs, and an arc leads from vertex (S_iS_j) to vertex (S_pS_q) if and only if (S_iS_j) implies (S_pS_q). It is a tool which aids in the search for a minimal closed covering.

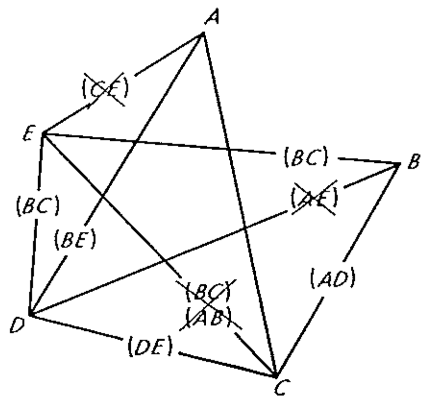

Fig. 10-3 Merger graph for machine M_7.

The compatible pairs and their implied pairs are usually obtained from the merger graph, and since a set of states is a compatible if and only if every pair of states in that set is compatible, then for a given machine, the set of compatible pairs defines uniquely the entire set of compatibles.† In the compatibility graph of machine M_7 (Fig. 10-4) an arc leads from vertex (AD) to vertex (BE) because the compatibility of (BE) is implied by that of (AD). No arcs emanate from (AC) since no other compatible is implied by it.

A subgraph of a compatibility graph is said to be closed if, for every vertex in the subgraph, all outgoing arcs and their terminating vertices also belong to the subgraph. If, in addition, every state of the machine is covered by at least one vertex of the subgraph, then the subgraph forms a closed covering for that machine.

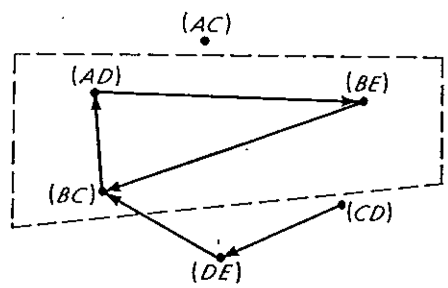

Fig. 10-4 Compatibility graph for machine M_7.

† In order to take into account those states which are incompatible with all other states, the definition of the set of compatible pairs must be generalized to include the pairs corresponding to self-compatibility, i.e., (AA), (BB), etc.

Example The compatibility graph of Fig. 10-4 contains seven closed subgraphs [including (AC) alone and the graph itself], six of which form closed coverings for M_7; among them we find the subgraphs corresponding to the following coverings:

$$\{(BC),(AD),(BE)\} \qquad \{(AC),(BC),(AD),(BE)\}$$
$$\{(DE),(BC),(AD),(BE)\} \quad \blacksquare$$

The compatibility graph itself forms a closed covering. However, it is often desirable to look for a closed subgraph which yields a simpler machine. If a closed subgraph containing the compatible pairs (S_iS_j), (S_jS_k), and (S_iS_k) has been found, the compatible $(S_iS_jS_k)$ can be formed, and so on. Although the number of states in the minimal machine is not necessarily proportional to the number of vertices in the closed graph, the inclusion of many redundant vertices in it does tend to increase the size of the machine. Unfortunately, there is no simple, precise procedure leading to the selection of the minimal closed covering, and trial-and-error technique cannot be avoided. The compatibility graph thus serves to display the various possible reduced machines which correspond to the different closed coverings.

In the compatibility graph of machine M_7, state B is covered by vertices (BE) and (BC); and since at least one of them must be included in any closed covering, the entire triangle $\{(BC),(AD),(BE)\}$ must also be included. This triangle, being a closed graph which covers every state of M_7, implies that the corresponding set of compatibles yields the desired minimal machine. Its state table is shown in Table 10-17, where the entry β/γ means that the next state may be either β or γ.

Table 10-17 A minimal machine which covers M_7

PS	NS, z: I_1	I_2	I_3	I_4
$(AD) \to \alpha$	—	γ,1	γ,1	—
$(BC) \to \beta$	β,0	α,1	β/γ,0	α,0
$(BE) \to \gamma$	β,0	α,1	β,0	β/γ,0

The merger table

When dealing with machines having a large number of states, it may be more convenient to record the compatible pairs and their implications in a merger table of the form illustrated in Fig. 10-5, instead of using the

	A	B	C	D	E
B	EF				
C	BC	AC, EF			
D	×	×	EF		
E	×	×	✓	CD, CF	
F	DE	~~AB, DF~~	BC, DE	~~BD~~	BC, CD

Fig. 10-5 Merger table for machine M_8.

merger graph. Each cell of the table corresponds to the compatible pair defined by the intersection of the row and column headings. The incompatibility of two states is recorded by placing an $\times$ in the corresponding cell, while their compatibility is recorded by a check mark ($\checkmark$). The entries in cell S_i,S_j are the pairs implied by (S_iS_j).

As an example, let us construct the merger table for machine M_8. The table is shown in Fig. 10-5. An $\times$ is inserted in cell (AD), since states A and D have conflicting outputs; a check mark is inserted in cell (CE), because state E contains state C. In a similar way the entire table is completed, and the implied compatibles are entered in the appropriate cells. Now it becomes necessary to check whether these entries indeed correspond to compatible pairs. Starting from the rightmost cell, we find no contradiction until we arrive at the entry (BD) in cell (DF). Since there is an $\times$ in cell (BD), the pair (DF) is incompatible and is therefore "crossed off." As a consequence of the incompatibility of (DF), the pair (BF) is also incompatible, and the corresponding cell is crossed off.

Table 10-18 Machine M_8

PS	NS, z: I_1	NS, z: I_2
A	E,0	B,0
B	F,0	A,0
C	E,–	C,0
D	F,1	D,0
E	C,1	C,0
F	D,–	B,0

Once the merger table has been completed, we continue to construct the corresponding compatibility graph and to find a closed subgraph, in order to obtain the smallest closed set of compatibles. Before continuing in the above-outlined direction, we shall pause and describe a procedure for finding the set of all maximal compatibles. This procedure is the tabular counterpart to that of finding complete polygons in the merger graph. It is executed in the following manner:

1. Start in the rightmost column of the merger table and proceed left until a column containing a compatible pair is encountered. List all the compatible pairs in that column. In our example this step yields the pair (EF).
2. Proceed left to the next column containing at least one compatible pair. If the state to which this column corresponds is compatible with all members of some previously determined compatible, add this state to that compatible to form a larger compatible. If the state is not compatible with all members of a previously determined compatible, but is compatible with some members of such a compatible, form a new compatible which includes those members and the state in question. Next list all compatible pairs which are not included in any previously derived compatible.
3. Repeat step 2 until all columns have been considered. The final set of compatibles constitutes the set of maximal compatibles.

Applying this procedure to the merger table of machine M_8 yields the following sequence of compatibility classes:

Column E:	(EF)
Column D:	(EF), (DE)
Column C:	(CEF), (CDE)
Column B:	(CEF), (CDE), (BC)
Column A:	(CEF), (CDE), (ABC), (ACF)

From column C it is evident that state C is compatible with states D, E, and F, and consequently the compatibles generated previously are enlarged to include state C. Column B, on the other hand, consists of a single compatible pair, which is added to the previously generated list. From column A, rows B and C, we obtain the compatible (ABC), while rows C and F, together with previously available compatibility relations, yield the compatible (ACF). The final list is the set of maximal compatibles of machine M_8.

The set of maximal compatibles clearly indicates that machine M_8 can be covered by a four-state machine, and cannot be covered by any two-state machine. To determine whether a three-state machine which

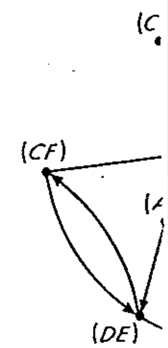

Fig. 10-6 Co
M_8.

covers M_8 exists, we construc
10-6. It must be emphasize
short cut can be taken, and t
directly from the state ta
merger graph or table.

An initial inspection of t
subgraph which covers every
tices. In fact, any such grap
are (AC), (BC), (EF), and (C
seem that there exists no thr
ever, it has been pointed out
larger closed subgraph if the
patible pairs to yield larger c
add vertex (AB) to the preced
of five compatible pairs {(A
to the following closed coveri

$\{(ABC),(CD),(EF)\}$

Thus the minimum-state ma
states, and is given in Table 1

Table 10

PS
(ABC)
(CD)
(EF)

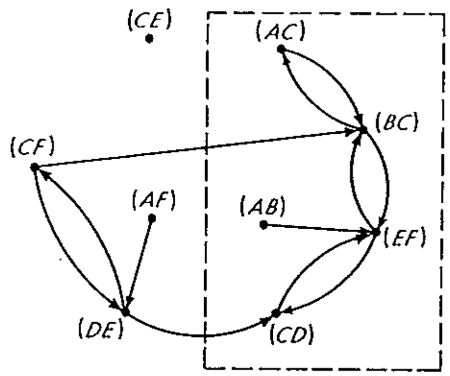

Fig. 10-6 Compatibility graph for machine M_8.

covers M_8 exists, we construct the compatibility graph, as shown in Fig. 10-6. It must be emphasized at this point that in many simple cases a short cut can be taken, and the compatibility graph can be constructed directly from the state table, without the need to find first the merger graph or table.

An initial inspection of the compatibility graph does not reveal any subgraph which covers every state of M_8 and consists of just three vertices. In fact, any such graph must contain the subgraph whose vertices are (AC), (BC), (EF), and (CD). And since this graph is closed, it may seem that there exists no three-state machine which covers M_8. However, it has been pointed out earlier that it may be desirable to find a larger closed subgraph if the added vertices can be used to merge compatible pairs to yield larger compatibles. In the above example, if we add vertex (AB) to the preceding subgraph, we obtain a set which consists of five compatible pairs $\{(AB),(AC),(BC),(EF),(CD)\}$ and is reducible to the following closed covering:

$$\{(ABC),(CD),(EF)\}$$

Thus the minimum-state machine which covers M_8 consists of three states, and is given in Table 10-19.

Table 10-19 A minimal machine which covers M_8

PS	NS, z	
	I_1	I_2
$(ABC) \to \alpha$	γ,0	α,0
$(CD) \to \beta$	γ,1	β,0
$(EF) \to \gamma$	β,1	α,0

machines was first studied by
ıded to synchronous machines
re for incompletely specified
ınd Unger [8], and Kohavi [5].
chines are available in Grasselli

inimal State Sequential Machines,
-8, no. 1, pp. 13–24, March, 1959.
ıs States in a Sequential Machine,
259–282, April, 1959.
mbined Row-Column Reduction of
nth *Symposium Switching and Auto-*
witching Circuits, *J. Franklin Inst.*,
). 275–303, 1954.
ified Sequential Switching Circuits,
tute of Brooklyn, PIBMRI, May,
quential Circuits, *Bell System Tech.*
quential Machines," pp. 129–153,
ess, Princeton, N.J., 1956.
Number of States in Incompletely
IRE Trans. Electron. Computers,

er bound on the length of the
a strongly connected n-state
once, regardless of the initial

ne for which the length of such
(A machine for which the
er of trials.)

ith a periodic input sequence

ust eventually become periodic,

M_1^* (Table 10-2) to the input
, find the period of the output

sequence and the amount of time required for the periodic behavior to start.

10-3. Prove that there exists no finite-state machine that accepts precisely *all* those sequences that read the same forward as backward, i.e., sequences that are their own reverses. (Such sequences are called *palindromes.*)

Hint: Suppose that there exists an n-state machine that accepts all palindromes; then it accepts the sequence $\underbrace{00 \cdots 0}_{n+1}01\underbrace{00 \cdots 00}_{n+1}$.
But this implies that it also accepts a sequence that is not a palindrome.

✓ **10-4.** Determine which of the machines with the following specifications is realizable with a finite number of states. If any machine is not realizable, explain why.

(*a*) A machine is to produce an output of 1 whenever the number of 1's in the input sequence, starting at $t = 1$, exceeds the number of 0's. For example, if the input is 01100111, the required output is 00100011.

(*b*) A machine with a single input line and 10 output lines numbered 0 through 9 is to be designed so that, following the nth input pulse, only one output pulse will be produced in the line whose corresponding number is equal to the nth digit of π (i.e., $3.14 \cdots$).

✓ **10-5.** (*a*) Find the equivalence partition for the machine shown in Table P10-5.

(*b*) Show a *standard form* of the corresponding reduced machine.

(*c*) Find a minimum-length sequence that distinguishes state A from state B.

Table P10-5

PS	NS, z $x = 0$	 $x = 1$
A	B,1	H,1
B	F,1	D,1
C	D,0	E,1
D	C,0	F,1
E	D,1	C,1
F	C,1	C,1
G	C,1	D,1
H	C,0	A,1

10-6. For each of the machines in Table P10-6, find the equivalence partition and a corresponding reduced machine in standard form.

Table P10-6

PS	NS, z: x = 0	x = 1
A	B,0	E,0
B	E,0	D,0
C	D,1	A,0
D	C,1	E,0
E	B,0	D,0

(a)

PS	NS, z: x = 0	x = 1
A	F,0	B,1
B	G,0	A,1
C	B,0	C,1
D	C,0	B,1
E	D,0	A,1
F	E,1	F,1
G	E,1	G,1

(b)

PS	NS, z: x = 0	x = 1
A	D,0	H,1
B	F,1	C,1
C	D,0	F,1
D	C,0	E,1
E	C,1	D,1
F	D,1	D,1
G	D,1	C,1
H	B,1	A,1

(c)

10-7. Two columns of the state table of an eight-state, p-inputs finite-state machine are shown in Table P10-7. Prove that this machine has either no equivalent states or else no distinguishable states.

Table P10-7

PS	NS, z: I_i	I_j
A	A,1	H,0
B	C,1	A,0
C	D,1	B,0
D	E,1	C,0
E	F,1	D,0
F	G,1	E,0
G	H,1	F,0
H	B,1	G,0

10-8. A *transfer sequence* $T(S_i,S_j)$ is defined as the shortest input sequence that takes a machine from state S_i to state S_j.

(a) Find a general procedure to determine the transfer sequence for a given machine and two specified states.

(b) Find a transfer sequence $T(A,G)$ for the machine shown in Table P10-8.

Hint: It is helpful to first determine which states can be reached from S_i by sequences of length 1, then by sequences of length 2, and so on.

10-9. (a) Develop a proced
that distinguishes a state S
(b) Apply your proce
that distinguishes state A
Hint: Start from the
in separate blocks.

10-10. The *direct sum* M_1 +
by combining the tables of
P10-10, so that each state
symbol.
(a) Use the direct su
M_1 is equivalent to state H
(b) Prove that machi
(c) Under what sta
equivalent?
Hint: Find the equiv

PS	NS, z: x = 0	x = 1
A	B,0	C,1
B	D,1	C,0
C	A,1	C,0
D	B,1	C,0

M_1

Table P10-8

PS	NS, z $x = 0$	$x = 1$
A	$A,0$	$B,0$
B	$C,0$	$D,1$
C	$E,0$	$D,0$
D	$F,0$	$E,1$
E	$G,0$	$A,0$
F	$G,0$	$B,1$
G	$C,0$	$F,0$

10-9. (*a*) Develop a procedure to determine the shortest input sequence that distinguishes a state S_i from another state, S_j, of a given machine.

(*b*) Apply your procedure to determine the shortest input sequence that distinguishes state A from state G in the machine of Table P10-8.

Hint: Start from the first partition, P_k, in which S_i and S_j appear in separate blocks.

10-10. The *direct sum* $M_1 + M_2$ of two machines, M_1 and M_2, is obtained by combining the tables of the individual machines, as shown in Table P10-10, so that each state of the direct sum is denoted by a distinct symbol.

(*a*) Use the direct sum to determine whether state A of machine M_1 is equivalent to state H of machine M_2.

(*b*) Prove that machine M_1 is contained in machine M_2.

(*c*) Under what starting conditions are machines M_1 and M_2 equivalent?

Hint: Find the equivalence partition of the direct sum.

Table P10-10

PS	NS, z $x = 0$	$x = 1$
A	$B,0$	$C,1$
B	$D,1$	$C,0$
C	$A,1$	$C,0$
D	$B,1$	$C,0$

M_1

PS	NS, z $x = 0$	$x = 1$
E	$H,1$	$E,0$
F	$F,1$	$E,0$
G	$E,0$	$G,1$
H	$F,0$	$E,1$

M_2

PS	NS, z $x = 0$	$x = 1$
A	$B,0$	$C,1$
B	$D,1$	$C,0$
C	$A,1$	$C,0$
D	$B,1$	$C,0$
E	$H,1$	$E,0$
F	$F,1$	$E,0$
G	$E,0$	$G,1$
H	$F,0$	$E,1$

$M_1 + M_2$

10-11. (*a*) Let M_1 and M_2 be strongly connected and completely specified machines, and suppose that a state S_i of M_1 is equivalent to a state S_j of M_2. Prove that M_1 is equivalent to M_2.

(*b*) Let M_1 be a strongly connected machine, and let M_2 be completely specified. Prove that if S_i of M_1 is equivalent to S_j of M_2, then M_1 is covered by M_2.

10-12. Determine the conditions under which two equivalent machines are isomorphic.

10-13. An unknown two-input, three-state machine produces the output sequence Z in response to the input sequence X:

$$\begin{array}{lcccccccccccc} X: & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ Z: & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}$$

Assuming that A is the initial state, determine the reduced standard form description of the machine.

10-14. In this problem we shall establish a procedure for transforming a Mealy machine into a corresponding Moore machine, so that both accept exactly the same sets of sequences. To obtain the Moore machine, it is first necessary to split every state of the Mealy machine if different output values are associated with the transitions into that state. For example, state B of Table P10-14*a* can be reached from either state A or state C. But since different outputs are associated with these transitions, state B must be replaced by two equivalent states, B_0 with an output 0 and B_1 with an output 1, as shown in Table P10-14*b*. Every transition to B with a 0 output is directed to B_0, and every transition to B with a 1 output to B_1. Applying the same procedure to state D yields the state table of Table P10-14*b*, which can be transformed to the Moore machine of Table P10-14*c*.

We now observe that the Moore machine of Table P10-14*c* accepts those sequences accepted by the Mealy machine of Table P10-14*a*, but in addition it produces an output 1 when started in state A, without having been presented with any input sequence. Thus this Moore machine in fact accepts a zero-length sequence, called the *null sequence*. To prevent this situation, we add a new starting state A', whose state transitions are identical with those of A but whose output is 0, as shown in Table P10-14*d*.

(*a*) Prove that, to every q-output, n-state Mealy machine, there corresponds a q-output Moore machine which accepts exactly the same sequences and has no more than $qn + 1$ states.

(*b*) If the definitio
so that acceptance of th
for transforming a Moor
that both accept the san

(*c*) Prove that if
completely specified, th
strongly connected and

PS	NS, $x = 0$
A	$C,0$
B	$A,1$
C	$B,1$
D	$D,1$

(*a*)

PS	NS $x = 0$	$x =$
A	C	B
B_0	A	D
B_1	A	D
C	B_1	A
D_0	D_1	C
D_1	D_1	C

(*c*)

10-15. Give a procedure t
pletely specified machin
contains M_2 or vice vers

10-16. (*a*) Find all the s
in Table P10-16.

(*b*) Find two mi
machine, and prove that

cted and completely specified
I_1 is equivalent to a state S_j

machine, and let M_2 be com-
equivalent to S_j of M_2, then

ch two equivalent machines

machine produces the output
ce X:

0 1 0
0 0 1

ne the reduced standard form

procedure for transforming a
machine, so that both accept
tain the Moore machine, it is
ly machine if different output
to that state. For example,
om either state A or state C.
with these transitions, state
s, B_0 with an output 0 and B_1
-14*b*. Every transition to B
ransition to B with a 1 output
te D yields the state table of
the Moore machine of Table

ine of Table P10-14*c* accepts
chine of Table P10-14*a*, but
started in state A, without
equence. Thus this Moore
nce, called the *null sequence*.
tarting state A', whose state
whose output is 0, as shown

-state Mealy machine, there
ich accepts exactly the same
tes.

(*b*) If the definition of acceptance by a Moore machine is modified so that acceptance of the null sequence is disregarded, show a procedure for transforming a Moore machine to a corresponding Mealy machine so that both accept the same sequences.

(*c*) Prove that if the Mealy machine is strongly connected and completely specified, the corresponding Moore machine will also be strongly connected and completely specified.

Table P10-14

PS	NS, z: $x = 0$	NS, z: $x = 1$
A	C,0	B,0
B	A,1	D,0
C	B,1	A,1
D	D,1	C,0

(*a*)

PS	NS, z: $x = 0$	NS, z: $x = 1$
A	C,0	B_0,0
B_0	A,1	D_0,0
B_1	A,1	D_0,0
C	B_1,1	A,1
D_0	D_1,1	C,0
D_1	D_1,1	C,0

(*b*)

PS	NS: $x = 0$	NS: $x = 1$	z
A	C	B_0	1
B_0	A	D_0	0
B_1	A	D_0	1
C	B_1	A	0
D_0	D_1	C	0
D_1	D_1	C	1

(*c*)

PS	NS: $x = 0$	NS: $x = 1$	z
A'	C	B_0	0
A	C	B_0	1
B_0	A	D_0	0
B_1	A	D_0	1
C	B_1	A	0
D_0	D_1	C	0
D_1	D_1	C	1

(*d*)

10-15. Give a procedure that can be used to determine whether two incompletely specified machines, M_1 and M_2, are related, so that either M_1 contains M_2 or vice versa.

10-16. (*a*) Find all the state containments present in the machine shown in Table P10-16.

(*b*) Find two minimum-state machines that contain the given machine, and prove that these machines are indeed minimal.

Table P10-16

PS	NS, z $x = 0$	$x = 1$
A	B,0	C,1
B	D,0	C,1
C	A,0	E,0
D	—	F,1
E	G,1	F,0
F	B,0	—
G	D,0	E,0

10-17. For each of the incompletely specified machines shown in Table P10-17, find a minimum-state reduced machine containing the original one.

Table P10-17

PS	NS, z I_1	I_2	I_3
A	C,0	E,1	—
B	C,0	E,–	—
C	B,–	C,0	A,–
D	B,0	C,–	E,–
E	—	E,0	A,–

(a)

PS	NS, z I_1	I_2
A	—	F,0
B	B,0	C,0
C	E,0	A,1
D	B,0	D,0
E	F,1	D,0
F	A,0	—

(b) ✓

✓ **10-18.** Prove that the machine shown in Table P10-18 is minimal.

Table P10-18

PS	NS, z I_1	I_2	I_3	I_4	I_5	I_6	I_7
A	F,0	A,–	D,–	C,–	—	—	—
B	–,1	—	—	—	C,–	D,–	E,–
C	C,–	E,–	—	—	F,0	B,–	—
D	—	—	F,–	E,–	–,1	—	A,–
E	A,–	—	A,1	—	B,–	—	C,–
F	—	D,–	–,0	B,–	—	E,–	—

✓ **10-19.** Find the reduced
Design the circuit using a

PS	(
A	A
B	-
C	A
D	A
E	-
F	-
G	A

10-20. Design a serial to
circuit has a single input
four output lines, z_1, z_2, z
in BCD code. The inpu
first. The outputs are s
input. For example, if
3 code), the required out

10-19. Find the reduced state table for the machine of Table P10-19. Design the circuit using a single *SR* flip-flop.

Table P10-19

PS	*NS*, z_1z_2			
	00	01	11	10
A	*A*,00	*E*,01	—	*A*,01
B	—	*C*,10	*B*,00	*D*,11
C	*A*,00	*C*,10	—	—
D	*A*,00	—	—	*D*,11
E	—	*E*,01	*F*,00	—
F	—	*G*,10	*F*,00	*G*,11
G	*A*,00	—	—	*G*,11

10-20. Design a serial to parallel, Excess-3 to BCD code converter. The circuit has a single input line, receiving messages in Excess-3 code, and four output lines, z_1, z_2, z_4, z_8, which are to reproduce the input messages in BCD code. The inputs arrive serially, with the least significant digit first. The outputs are specified only at the occurrence of every fourth input. For example, if the input sequence is 1001 (which is 6 in Excess-3 code), the required output is $z_1 = 0$, $z_2 = 1$, $z_4 = 1$, $z_8 = 0$.