9

Equivalence and Minimization of Sequential Machines

9.1. THE PROBLEM OF RECOGNITION OF EQUIVALENT STATES

We have already said in Section 3.7 that the class of possible input sequences to a machine may be restricted for some reason, and we have seen such a case in Chapter 5. Now consider other constraints that may be imposed on the possible input sequences. They may include the following:

a) Identical symbols shall not appear consecutively.
b) Symbol ρ_j shall not follow symbol ρ_i.
c) An input sequence shall not begin (or, conversely, must begin) with ρ_k.
d) If the sequence contains ρ_s or ρ_t then it cannot contain ρ_q.

In these examples, the infinite set E containing all the possible input symbol sequences of any desired but finite length is split into two subsets: a subset $L \subseteq E$ (which may be finite or infinite), containing all the input sequences allowable in a given s-machine, and a complement of this subset $\bar{L}$, consisting of the set of forbidden input sequences. A special case is that of $L = E$, which means that any input sequence is allowable in the given machine.

Let us note that constraints a - d are in no way related to the state in which the machine happens to be. There may, however, exist other constraints, imposed by the design of the s-machine. For example, its state diagram may show an ith state such as that of Fig. 9.1. Here, the machine cannot accept an input ρ_2, and the constraint on the input sequence is thus imposed by the properties of the machine itself. Such constraints may be imposed on one, several, or even all the states of the machine. The constraints on different states need not

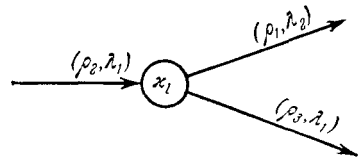


Fig. 9.1.

be identical, and different states may thus forbid different input signals.

If a given input sequence does not violate the constraints imposed by state $\varkappa_i$ and all the states following $\varkappa_i$, it is said to be an *input sequence allowed in state* $\varkappa_i$. The set of all sequences allowed in state $\varkappa_i$ is denoted by $L_{\varkappa_i}$.

Constraints imposed by the properties of the machine are known as *Aufenkamp constraints*. If Aufenkamp constraints are operative, then there is no single, all-encompassing set L such that any member of L would be an allowed input sequence regardless of the initial state of the machine. With Aufenkamp constraints, each of the possible k states of the machine has its own $L_{\varkappa_i}$ which, in general, is different from the other sets $L_{\varkappa_j}$ $(j \neq i)$.

This case differs from that in which the constraints are independent of the states of the s-machine (that is, where there exists a single set L) and where the algorithm for recognizing whether a given sequence belongs to set L can be formulated in terms unrelated to the initial state of the machine (or even completely unrelated to the machine). When the recognition algorithm exists *per se*, that is, may be expressed in terms unrelated to the machine, the corresponding constraints are said to be *constraints per se*, and the set of the possible input sequences is said to be *restricted per se*.

Considering for the time being only sequences restricted *per se*, let us introduce the concept of *equivalence of states* of an s-machine (or a finite automaton). Assume we are given the set L of allowable input sequences, as well as two s-machines S and G (in particular, S may coincide with G). Then state $\varkappa_i$ of S and state $\varkappa_j$ of G are equivalent in terms of L if the two machines (in these two respective states) process the same input sequence from L into identical outputs. If the machines S and G are identical, then this definition merely describes the conditions for equivalence (in terms of L) of the two states of a single s-machine (or automaton). If $L = E$, that is, if the set of allowable input sequences contains all possible sequences, then states $\varkappa_i$ and $\varkappa_j$ are *simply equivalent*.

Our definition of equivalent states underlies the following analytical problem: Given an s-machine and its set of allowable input sequences L, find an algorithm for deciding whether two arbitrarily chosen states of that s-machine are equivalent in terms of L. If we had such an algorithm, we could split the set of all states E into groups of those that are equivalent in terms of L. By a *group of states equivalent in terms of* L we mean a set of states of the s-machine such that: (1) any two states in the group are equivalent in terms of L; and (2) no state from one group is equivalent (in terms

of L) to any other state of any other group. This grouping of states is, as will be seen later, of paramount importance in the minimization of s-machines.

Our generalized analytic problem would be solvable if we had an algorithm for recognizing the equivalence of states, given any s-machine and any set L. But we will show in Section 9.2 that this generalized problem is algorithmically unsolvable, and so we shall be forced to tackle recognition problems one specific case after another, as in Sections 9.3 and 9.4.

9.2. ALGORITHMIC UNSOLVABILITY OF THE GENERALIZED RECOGNITION PROBLEM OF RECOGNITION OF EQUIVALENCE OF STATES

To be useful, the set of allowable input sequences L should be *effectively specified.* In other words, for each specified set L there should exist an algorithm for recognizing whether a given finite sequence of input symbols belongs to L. For example, a finite set L can be effectively specified by simple enumeration of all sequences contained in it. But this cannot be done for an infinite set L, which must be specified in some other way, for instance, by specifying a recognition algorithm. Set L may, for example, be specified verbally by stating that:

1) it contains all sequences longer than three symbols, wherein the fourth symbol is ρ_i; or

2) it contains only those sequences ending in ρ_j which do not comprise ρ_q.

These sets, even though infinite, are fully characterized by their respective verbal descriptions, and thus it is always possible to tell whether they contain any given sequence. The mere fact that such an effective verbal description can be formulated shows that there must exist an algorithm accomplishing the same thing, that is, recognizing whether a given sequence belongs to the given set L. In this sense, the recognition algorithm is the least artificial and the broadest language for effective definition of infinite sets L.

We shall now try to ascertain whether it is possible to determine the equivalence of two states with respect to an arbitrary *effectively specified* set L, that is, a set L defined by a recognition algorithm. To start with, we must formalize the concept of a recognition algorithm. As usual, we turn for help to the theory of algorithms and recursive functions,* which asserts that any set of sequences for

*See Chapter 12, and also Section 8.3.

which one can define "recognition rules" is recursive; conversely, one can define such recognition rules for any recursive set (this assertion is a direct result of Church's theses — see Section 12.11).

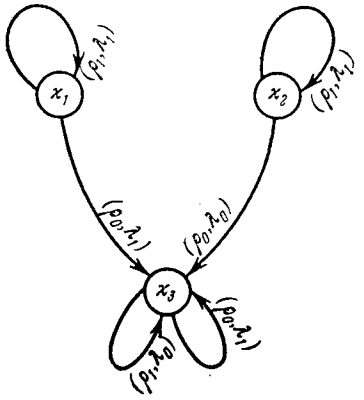


Fig. 9.2.

Let L be an arbitrary recursive set of input sequences, and let x_i and x_j be arbitrary states of s-machines S and G, respectively. Then the following theorem is true:

Theorem. The problem of recognition of equivalence of states x_i and x_j in terms of an arbitrary, effectively defined set L is algorithmically unsolvable.

We shall prove this theorem by demonstrating the algorithmic unsolvability of the narrower problem of recognition of equivalence of states in a special machine, whose allowed set L belongs to a special subclass of recursive sets. If the problem is algorithmically unsolvable in this special case, then it is certainly unsolvable in the general case.

Consider a three-state, s-machine N and its state diagram (Fig. 9.2). In this machine $r = 2$, that is, the input alphabet consists of only two symbols $\{0, 1\}$. The output can be either λ_0 or λ_1.

Now we shall deal with a special class of recursive sets containing sequences of 0 and 1, and defined as follows: Let $\varphi(t)$ be an arbitrary general recursive function, and let the set L_φ contain only the following sequences* of 0 and 1:

$$\mathrm{sg}(\varphi(0));$$
$$\mathrm{sg}(\varphi(0)),\quad \mathrm{sg}(\varphi(1));$$
$$\mathrm{sg}(\varphi(0)),\quad \mathrm{sg}(\varphi(1)),\quad \mathrm{sg}(\varphi(2));$$
$$\mathrm{sg}(\varphi(0)),\quad \mathrm{sg}(\varphi(1)),\quad \mathrm{sg}(\varphi(2)),\quad \mathrm{sg}(\varphi(3));$$
and so on.

Then set L_φ is recursive at any recursive function $\varphi(t)$. Indeed, each $(p+1)$ long sequence of 0 and 1 can be placed into correspondence with a definite value of the integer function

$$\Phi(p) = \sum_{t=0}^{t=p} \mathrm{sg}(\varphi(t))\, 2^t + 2^{p+1}$$

*The notation $\mathrm{sg}(x)$ denotes a function which is equal to 1 for $x \geqslant 1$ and equal to 0 for $x = 0$. The function is undefined for $x < 0$.

defined on set $\Lambda(L_\varphi)$. Function $\Phi(p)$ is an increasing function and, by virtue of the recursivity of $\varphi(t)$, is also recursive.* Consequently, set $\Lambda(L_\varphi)$ is consecutively enumerated as the recursive function $(d)\Phi$ increases, and therefore is a recursive set. Hence the set L_φ must also be recursive.

It is readily seen that the states $\varkappa_i$ and $\varkappa_j$ (where $i, j = 1, 2, 3$; $i \neq j$) of the machine N (Fig. 9.2) are equivalent to each other in terms of L_φ if, and only if, L_φ contains no sequences comprising ρ_0. Thus the problem of recognition of equivalence of states of N is algorithmically solvable only if there exists an algorithm capable of recognizing whether L_φ contains even one sequence comprising ρ_0. But such an algorithm cannot be written unless there is an algorithm for recognizing whether a given arbitrary recursive function $\varphi(t)$ becomes zero at some $t = t_*$. And it has been proved [142] that no such algorithm exists. For that reason, our narrow recognition problem is algorithmically unsolvable, and the generalized problem of recognition equivalence of two states of an arbitrary s-machine with respect to an arbitrary recursive set L is *a fortiori* algorithmically unsolvable. This proves the theorem.**

9.3. RECOGNITION OF THE EQUIVALENCE OF STATES IN THE CASE OF AN UNRESTRICTED SET OF INPUT SEQUENCES

Let no restrictions be imposed on the set of allowable input sequences, that is, let $L = E$. In this case the algorithm merely recognizes the simple equivalence of the s-machine states. For this case we have a straightforward and convenient algorithm, which is due to Aufenkamp and Hohn.***

To start with, let us point out an obvious attribute of equivalence of states: if any two states of an s-machine are equivalent with respect to set L_1, then they will also be equivalent with respect to set L_2, provided $L_2 \subseteq L_1$.**** Conversely, two states can be equivalent with respect to L_1, where $L_1 \supseteq L_2$, only if they are also equivalent with respect to L_2.

---

*See Sections 12.6 and 12.13.

**But in no way implies that the problem is unsolvable in special cases.

***Aufenkamp and Hohn [6] proved only the sufficiency of this algorithm. We shall give a somewhat different proof for its sufficiency, and shall also prove its necessity.

****This is read as L_1 is a subset of L_1.

Now assume that we are given some s-machine, for which we write the interconnection matrix, obtaining, for example, matrix C

$$C = \begin{array}{c} \\ \varkappa_1 \\ \varkappa_2 \\ \varkappa_3 \\ \varkappa_4 \\ \varkappa_5 \\ \varkappa_6 \end{array} \begin{array}{c} \begin{array}{cccccc} \varkappa_1 & \varkappa_2 & \varkappa_3 & \varkappa_4 & \varkappa_5 & \varkappa_6 \end{array} \\ \begin{bmatrix} (\rho_0, \lambda_1) & (\rho_1, \lambda_2) & 0 & 0 & 0 & (\rho_2, \lambda_1) \\ (\rho_2, \lambda_0) & (\rho_1, \lambda_1) & (\rho_0, \lambda_2) & 0 & 0 & 0 \\ 0 & (\rho_0, \lambda_1) & (\rho_2, \lambda_1) & 0 & (\rho_1, \lambda_2) & 0 \\ 0 & (\rho_2, \lambda_1) & 0 & (\rho_0, \lambda_1) & (\rho_1, \lambda_2) & 0 \\ 0 & 0 & (\rho_0, \lambda_2) & (\rho_2, \lambda_0) & (\rho_1, \lambda_1) & 0 \\ 0 & (\rho_1, \lambda_1) & (\rho_0, \lambda_2) & (\rho_2, \lambda_0) & 0 & 0 \end{bmatrix} \end{array}$$

We decompose this matrix into groups of rows containing identical symbol pairs. Thus rows $\varkappa_1$, $\varkappa_4$, and $\varkappa_3$ fall into group 1, and rows $\varkappa_2$, $\varkappa_5$, and $\varkappa_6$ into group 2. We rewrite matrix C so as to be able to reflect this grouping with a minimum amount of effort, and transpose the columns in the same way. We draw a horizontal line between the groups, and obtain matrix

$$C = \begin{array}{c} \\ \varkappa_1 \\ \varkappa_4 \\ \varkappa_3 \\ \varkappa_2 \\ \varkappa_5 \\ \varkappa_6 \end{array} \begin{array}{c} \begin{array}{cccccc} \varkappa_1 & \varkappa_4 & \varkappa_3 & \varkappa_2 & \varkappa_5 & \varkappa_6 \end{array} \\ \begin{bmatrix} (\rho_0, \lambda_1) & 0 & 0 & (\rho_1, \lambda_2) & 0 & (\rho_2, \lambda_1) \\ 0 & (\rho_0, \lambda_1) & 0 & (\rho_2, \lambda_1) & (\rho_1, \lambda_2) & 0 \\ 0 & 0 & (\rho_2, \lambda_1) & (\rho_0, \lambda_1) & (\rho_1, \lambda_2) & 0 \\ \hline (\rho_2, \lambda_0) & 0 & (\rho_0, \lambda_2) & (\rho_1, \lambda_1) & 0 & 0 \\ 0 & (\rho_2, \lambda_0) & (\rho_0, \lambda_2) & 0 & (\rho_1, \lambda_1) & 0 \\ 0 & (\rho_2, \lambda_0) & (\rho_0, \lambda_2) & (\rho_1, \lambda_1) & 0 & 0 \end{bmatrix} \end{array},$$

which differs from the original matrix C in that rows and columns $\varkappa_2$ and $\varkappa_4$ are transposed.

Note that the states of each group are equivalent in terms of the set L^1 comprising all allowable input sequence of unit length. Indeed, within each group the output is independent of the state of the s-machine. For example, an input ρ_0 will always produce an output λ_1 regardless of whether the machine is in state $\varkappa_1$, $\varkappa_3$, or $\varkappa_4$. Furthermore, only states belonging to the same group can be equivalent at arbitrary allowable input sequences. States belonging to different groups are *a priori* nonequivalent in such cases because they are nonequivalent even in terms of set L^1 containing sequences of only unit length.

Our grouping into submatrices is helpful in clarifying some of the equivalence relations in the matrix, but is not sufficient. It does not guarantee that two states of a group will not become nonequivalent during later operation of the machine. To elucidate all of the possible equivalence relationships, we introduce a further decomposition of our matrix into symmetrical submatrices. Thus if the previously drawn horizontal line separated the kth and the $(k+1)$th

rows, we now draw a vertical line to separate the kth and the $(k+1)$th columns. In our example such a symmetrical decomposition is obtained by drawing a vertical line to separate columns $\varkappa_3$ and $\varkappa_2$.

$$C = \begin{array}{c|ccc|ccc} & \varkappa_1 & \varkappa_4 & \varkappa_3 & \varkappa_2 & \varkappa_5 & \varkappa_6 \\ \varkappa_1 & (\rho_0, \lambda_1) & 0 & 0 & (\rho_1, \lambda_2) & 0 & (\rho_2, \lambda_1) \\ \varkappa_4 & 0 & (\rho_0, \lambda_1) & 0 & (\rho_2, \lambda_1) & (\rho_1, \lambda_2) & 0 \\ \varkappa_3 & 0 & 0 & (\rho_2, \lambda_1) & (\rho_0, \lambda_1) & (\rho_1, \lambda_2) & 0 \\ \hline \varkappa_2 & (\rho_2, \lambda_0) & 0 & (\rho_0, \lambda_2) & (\rho_1, \lambda_1) & 0 & 0 \\ \varkappa_5 & 0 & (\rho_2, \lambda_0) & (\rho_0, \lambda_2) & 0 & (\rho_1, \lambda_1) & 0 \\ \varkappa_6 & 0 & (\rho_2, \lambda_0) & (\rho_0, \lambda_2) & (\rho_1, \lambda_1) & 0 & 0 \end{array}.$$

A submatrix in which any pair of symbols present in any row is also present in all the other rows is called a 1-matrix. In our example, the two submatrices below the horizontal line are 1-matrices, but the other two are not (for example, the top left-hand submatrix has a pair (ρ_2, λ_1) in the third row not present in the other rows). We shall try to decompose them symmetrically into 1-matrices. Such a decomposition is achieved by first drawing the minimum number of horizontal lines sufficient to convert the entire matrix into 1-matrices. But this partitioning will not be symmetrical. For this reason, one also draws vertical lines between the columns corresponding to the rows already separated by the horizontal lines. After this, we check whether all resulting submatrices are 1-matrices. If not, we again draw horizontal lines, and so on, until we obtain a completely symmetrical decomposition consisting only of 1-matrices.

In our example we draw a horizontal between rows $\varkappa_4$ and $\varkappa_3$. All the resulting submatrices are 1-matrices, and the vertical, drawn between columns $\varkappa_4$ and $\varkappa_3$ to achieve symmetry, does not upset this property:

$$C = \begin{array}{c|cc|cc|cc} & \varkappa_1 & \varkappa_4 & \varkappa_3 & \varkappa_2 & \varkappa_5 & \varkappa_6 \\ \varkappa_1 & (\rho_0, \lambda_1) & 0 & 0 & (\rho_1, \lambda_2) & 0 & (\rho_2, \lambda_1) \\ \varkappa_4 & 0 & (\rho_0, \lambda_1) & 0 & (\rho_2, \lambda_1) & (\rho_1, \lambda_2) & 0 \\ \hline \varkappa_3 & 0 & 0 & (\rho_2, \lambda_1) & (\rho_0, \lambda_1) & (\rho_1, \lambda_2) & 0 \\ \hline \varkappa_2 & (\rho_2, \lambda_0) & 0 & (\rho_0, \lambda_2) & (\rho_1, \lambda_1) & 0 & 0 \\ \varkappa_5 & 0 & (\rho_2, \lambda_0) & (\rho_0, \lambda_2) & 0 & (\rho_1, \lambda_1) & 0 \\ \varkappa_6 & 0 & (\rho_2, \lambda_0) & (\rho_0, \lambda_2) & (\rho_1, \lambda_1) & 0 & 0 \end{array}.$$

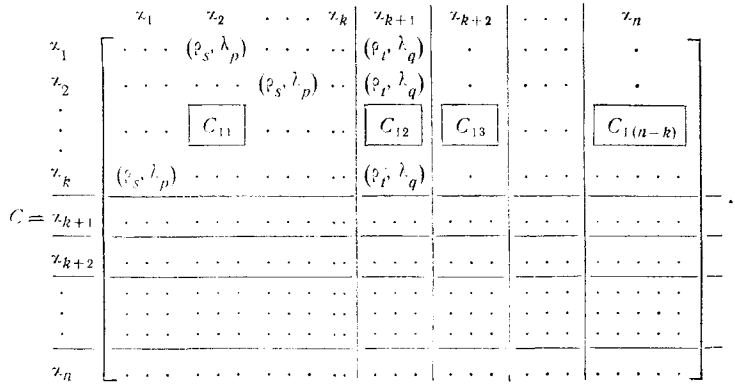
In the general case, this symmetrical decomposition will produce, after a finite number of steps, either of the following two situations:

1. The decomposition is *trivial*, that is, all the resulting submatrices are of the 1×1 order (there are separating lines between all the rows and all the columns).

2. The decomposition is *nontrivial*, that is, we have at least one submatrix of the order $m \times n$, where $\max(m, n) > 1$. In conrast to the trivial case, this partitioning gives *groups* of states. For example, the matrix shown above is split into three groups $\{\varkappa_1, \varkappa_4\}$, $\{\varkappa_3\}$, $\{\varkappa_2, \varkappa_5, \varkappa_6\}$.

The Aufenkamp–Hohn Theorem. The states of an s-machine are equivalent if, and only if, they are members of the same group formed by symmetrical decomposition of the given matrix C.

Proof of sufficiency of the conditions of the theorem. Suppose matrix C can be symmetrically and nontrivially decomposed into 1-matrices, and consider first the simple case when the matrix is



Here, we have only one group containing more than one state. This is the group of k states $\{\varkappa_1, \varkappa_2, \ldots, \varkappa_k\}$, that is, the $k \times k$ 1-matrix C_{11}. Let us write out all the input symbols appearing in C_{11}, and let these symbols be $\rho_{\alpha_1}, \rho_{\alpha_2}, \ldots, \rho_{\alpha_i}$, where $i \leqslant r$. The symbols with subscripts α_{i+1} to α_r do not appear in C_{11}. Now we shall prove that states $\varkappa_1, \varkappa_2, \ldots, \varkappa_k$ are equivalent.

Assume we have an arbitrary input sequence

$$\rho^0 \rho^1 \ldots \rho^j \rho^{j+1} \ldots \tag{9.1}$$

and that $\rho^{j+1} = \rho_{\alpha_{i+m}}$ is the first input symbol of this sequence which does not occur in C_{11}. With this input sequence, the machine output is independent of its initial state (which, by our assumption, must be one of the states of group $\{\varkappa_1, \varkappa_2, \ldots, \varkappa_k\}$). Indeed, until time j the input symbol ρ must belong in C_{11}, and therefore the machine must assume one of the states $\{\varkappa_1, \varkappa_2, \ldots, \varkappa_k\}$. But as long as the machine assumes one of these states, its output will depend only on the input (since C_{11} is a 1-matrix in which all the member pairs are identical.

Therefore, until time j the output sequence does not depend on which of the states $\{\varkappa_1, \varkappa_2, \ldots, \varkappa_k\}$ is the initial state of the machine. Then, at time $j+1$, the input signal becomes ρ^{j+1}. This input signal always shifts the machine into the same state $\varkappa_l$, regardless of which of the states $\{\varkappa_1, \varkappa_2, \ldots, \varkappa_k\}$ the machine happens to be in. This is because $C_{12}, C_{13}, \ldots, C_{1(n-k)}$ are all 1-matrices. Thus, the output is again independent of the previous state of the machine. The subsequent output of the machine is governed by the fact that at time $j+1$ it is in the state $\varkappa_l$ and, accordingly, it ceases to depend on the initial conditions.

Since sequence (9.1) was chosen at random, states $\varkappa_1, \varkappa_2, \ldots, \varkappa_k$ are equivalent.

Now consider the general case. Let C be symmetrically decomposed into 1-matrices

$$C = \begin{array}{c|c|c|c|c|c} & \varkappa_1 \ \varkappa_2 \ldots \varkappa_k & \varkappa_{k+1} \ \varkappa_{k+2} \ldots \varkappa_{k+s} & \ldots & \varkappa_{n-1} & \varkappa_n \\ \hline \varkappa_1, \varkappa_2, \ldots, \varkappa_k & C_{11} & C_{12} & \ldots & \ldots & \ldots \\ \hline \varkappa_{k+1}, \varkappa_{k+2}, \ldots, \varkappa_{k+s} & C_{21} & C_{22} & \ldots & \ldots & \ldots \\ \hline \vdots & \ldots & \ldots & \ldots & \ldots & \ldots \\ \hline \varkappa_{n-1} & \ldots & \ldots & \ldots & \ldots & \ldots \\ \hline \varkappa_n & \ldots & \ldots & \ldots & \ldots & \ldots \end{array}$$

We shall prove that the states occurring in any one group are equivalent. Indeed, any input signal can produce one of the following effects: (1) it can shift the machine from a state belonging to some group into a state belonging to the same group; in this case the output depends only on the input, and not on the state the machine is in; (2) it can shift the machine from a state of the ith group into a state of the jth group $(i \neq j)$. However, it is seen from the structure of matrix C, that in this case this input signal also shifts all the other states of group i into the states of group j. And once the machine is in a state of group j, its output again does not depend on the specific state in which it happens to be in; (3) it can shift the machine from a state of group i into a state $\varkappa_l$ not occurring in any group. If that happens, the input will also shift all the other states of group i into

state $\varkappa_l$, and the subsequent output of the machine will depend only on $\varkappa_l$.

The above reasoning holds for any of the groups of the matrix. Therefore, the output of the s-machine is always independent of the specific initial state of group i in which the machine happens to be. This being the case, groups of equivalent states behave as if each group were a single state. This proves the sufficiency of the conditions of the theorem.

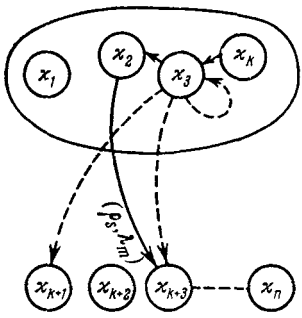


Fig. 9.3.

Proof of necessity of the conditions of the theorem. Consider first a simple case. Let the states $\varkappa_1, \varkappa_2, \ldots, \varkappa_h$ form a group of equivalent states of the s-machine, and let states $\varkappa_{h+1}, \varkappa_{h+2}, \ldots, \varkappa_n$ be nonequivalent to each other and to any state of the above group. Let us draw the state diagram of this machine. We shall now show that if state $\varkappa_i$ of group $\{\varkappa_1, \varkappa_2, \ldots, \varkappa_h\}$ is connected to any state $\varkappa_{h+m}$ outside that group by a path labeled $\rho_s \ldots$, then all the other states of that group must also be connected to the same state $\varkappa_{h+m}$, and their connecting paths must also be labeled $\rho_s \ldots$. That is, the first symbols of the path labels must coincide. For example, if state $\varkappa_2$ of Fig. 9.3 is connected to $\varkappa_{h+3}$ by a path whose label includes ρ_s as the first symbol, than the other states of the group of which $\varkappa_2$ is a member (that is, $\varkappa_1, \varkappa_3, \varkappa_4, \ldots, \varkappa_h$) must also be connected to $\varkappa_{h+3}$, and the connecting paths must also carry ρ_s as the first symbol of their labels. To prove this statement, consider state $\varkappa_2$ of the same group as $\varkappa_3$. On the face of it, the path labeled $\rho_s \ldots$, and originating in $\varkappa_3$ could follow one of the following courses:

1) it could lead from $\varkappa_3$ to one of the states of group $\{\varkappa_1, \varkappa_2, \ldots, \varkappa_h\}$;

2) it could lead from $\varkappa_3$ to one of the states $\varkappa_{h+i}$ (where $i \neq 3$), for example, to state $\varkappa_{h+1}$;

3) it could lead to state $\varkappa_{h+3}$, that is, to the same state as the path labeled $\rho_s \ldots$, originating in $\varkappa_3$.

We shall now show that case (3) is the only one possible. Indeed, assume for a moment that case (1) is possible. That would mean that the same input ρ_s could cause the machine to shift from state $\varkappa_2$ to state $\varkappa_{h+3}$, and from state $\varkappa_3$, which is equivalent to $\varkappa_2$, to some state $\varkappa_j$ of group $\{\varkappa_1, \varkappa, \ldots, \varkappa_h\}$. But state $\varkappa_{h+3}$ is by definition nonequivalent to any of the states of group $\{\varkappa_1, \varkappa_2, \ldots, \varkappa_h\}$; consequently, $\varkappa_{h+3}$ is also nonequivalent to $\varkappa_j$, and thus there would exist a sequence $\rho_{\alpha_1} \rho_{\alpha_2} \rho_{\alpha_3} \ldots$, such that its input to the machine would cause the latter

to generate different output sequences, depending on which of the states—$\varkappa_j$ or $\varkappa_{k+3}$—is the initial state of the machine. However, if this were the case, then the input sequence $\rho_s \rho_{\alpha_1} \rho_{\alpha_2} \rho_{\alpha_3} \ldots$ would also cause the generation of differing output sequences, depending on whether the initial state of the machine is $\varkappa_2$ or $\varkappa_3$. But that would be contrary to the assumed equivalence of states $\varkappa_2$ and $\varkappa_3$. Thus, case (1) is impossible.

Now assume that case (2) holds. Then an input ρ_s would shift the machine from state $\varkappa_2$ to state $\varkappa_{k+3}$, and from state $\varkappa_3$ to state $\varkappa_{k+1}$. But $\varkappa_{k+1}$ is, by definition, not equivalent to $\varkappa_{k+3}$; consequently, there would exist, just as in case (1), a sequence $\rho_{\beta_1} \rho_{\beta_2} \rho_{\beta_3} \ldots$, such that its input to the machine would cause the latter to generate different outputs, depending on which of the states—$\varkappa_{k+1}$ or $\varkappa_{k+3}$—is the initial one. But then the input of sequence $\rho_s \rho_{\beta_1} \rho_{\beta_2} \rho_{\beta_3} \ldots$ would again prove the nonequivalence of states $\varkappa_2$ and $\varkappa_3$, which would contradict the starting assumptions. Consequently, case (2) cannot hold, and the only possible case is (3), shown in Fig. 9.4. Here all states of group $\{\varkappa_1, \varkappa_2, \ldots, \varkappa_k\}$ are connected to the same state $\varkappa_{k+3}$ outside the group, and all the connecting paths bear a label whose first symbol is ρ_s. The second symbol of the label must also coincide, since otherwise it would be possible to prove by means of an input signal of length 1 that some pair of states from group $\{\varkappa_1, \varkappa_2, \ldots, \varkappa_k\}$ is nonequivalent, which would contradict the conditions of the problem.

It follows that if any state of group $\{\varkappa_1, \varkappa_2, \ldots, \varkappa_k\}$ *is connected with one of the states* $\varkappa_{k+1}, \varkappa_{k+2}, \ldots, \varkappa_n$ *by a path labeled* (ρ_s, λ_m), *then all the states of group* $\{\varkappa_1, \varkappa_2, \ldots, \varkappa_k\}$ *are also connected to that state by paths labeled* (ρ_s, λ_m) [*see Fig. 9.4*].

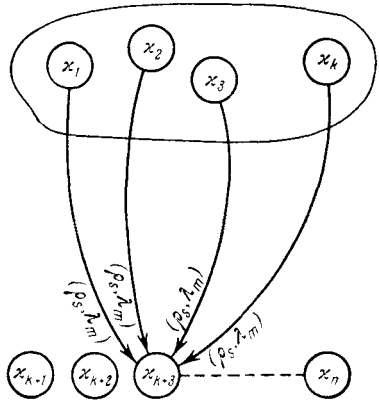


Fig. 9.4.

It also follows from the above that *if a path labeled* (ρ_s, λ_m) *connects a state of group* $\{\varkappa_1, \varkappa_2, \ldots, \varkappa_k\}$ *with another state of that group, then all the other similarly labeled paths from all the other states* $\{\varkappa_1, \varkappa_2, \ldots, \varkappa_k\}$ *must also terminate in states belonging to that group. That is, no path labeled* (ρ_s, λ_m) *leads to a state* $\varkappa_{k+1}, \varkappa_{k+2}, \ldots, \varkappa_n$ *outside the group.*

If this is so, then the interconnection matrix of our machine will be

$$C = \begin{array}{c|c|c|c|c|c} & \varkappa_1\ \varkappa_2 \ldots \varkappa_k & \varkappa_{k+1} & \varkappa_{k+2} & & \varkappa_n \\ \varkappa_1 \\ \varkappa_2 \\ \vdots \\ \varkappa_k & C_{11} & C_{12} & C_{13} & & C_{1\,(n-k+1)} \\ \hline \varkappa_{k+1} & C_{21} & \ldots & \ldots & \ldots & \ldots \\ \hline \varkappa_{k+2} & C_{31} & \ldots & \ldots & \ldots & \ldots \\ \hline \vdots & \ldots & \ldots & \ldots & \ldots & \ldots \\ \hline \varkappa_n & C_{(n-k+1)\,1} & \ldots & \ldots & \ldots & \ldots \end{array}.$$

Here all the submatrices $C_{11}, C_{12}, \ldots, C_{1(n-k+1)}$ are 1-matrices by virtue of the above italicized statements. The other submatrices C_{ij} (where $i = 2, 3, \ldots, n-k+1$ and $j = 1, 2, \ldots, n-k+1$) are also 1-matrices since they all contain only one row (or one member).

Our arguments also hold in the general case where there are several groups of pairwise equivalent states. However, some of the individual states $\varkappa_{k+1}, \varkappa_{k+2}, \ldots, \varkappa_n$ must then be replaced by groups of states; each of these groups behaves in a manner completely analogous to the group $\{\varkappa_1, \varkappa_2, \ldots, \varkappa_k\}$ of our particular case. This concludes the proof of the theorem.

The Aufenkamp and Hohn theorem results in a simple and very convenient algorithm for determining which groups of states of a given s-machine are equivalent. This algorithm merely consists of symmetrical decomposition of the interconnection matrix of the given s-machine.

9.4 RECOGNITION OF EQUIVALENCE OF STATES FOR THE CASE OF INPUT SEQUENCES OF LIMITED LENGTH

We shall now consider the problem of equivalence of states when the set L of allowable input sequences cannot contain sequences

comprising more than q symbols (that is, we analyze the operation of the s-machine during the first q discrete instants after the input). It is required to find an algorithm recognizing those states which are equivalent in terms of L, and to group these states together.

Since the total number of differing input symbols ρ_i is finite, and since no sequence can contain more than q such symbols, the number of different sequences in set L must be finite. This being the case, the required algorithm must exist. To ascertain that any two states $\varkappa_i$ and $\varkappa_j$ of a machine are equivalent in terms of L, it is sufficient to prove that given identical inputs from L, the machine starting from state $\varkappa_i$ will generate the same output as the same machine starting from state $\varkappa_j$, and that this will happen at all possible inputs from L. One can prove this by scanning either the state diagram of the machine, its interconnection matrix, or any other of its representations, or by an experiment on an existing machine. The algorithm for recognizing equivalence thus entails scanning of all the input-output relationships which are possible for a given set of two states. This obviously is a huge task. One way of organizing and, possibly, minimizing this unwieldy scanning procedure is to raise the interconnection matrix of the given s-machine to a power, a procedure described in Section 3.6. Let us now recall the properties of matrix C^q.

1. The element C^q_{ij} of C^q enumerates all those input sequences of length q which shift the machine from state $\varkappa_i$ to state $\varkappa_j$, as well as the corresponding output sequences.

2. Since the state of the machine at $t = p + 1$ is uniquely defined by its state and input at $t = p$, a single row of C^q cannot contain two elements whose terms comprise identical input sequences.

3. Each input sequence of length q must appear in each row of C^q.

Starting from these properties of C^q, one can derive the following method for determining the states equivalent in terms of L. Let us arrange set L in order of increasing sequence length. We now take the shortest sequence of L (if there are several such sequences, all of equal length, we can use any one of these), and find in matrix C^q (where q is the maximum length of a sequence of L) all those input sequences whose initial segments coincide with our shortest sequence. We mark these coinciding segments in some way, for instance, by placing dots over each of their constituent symbols. We repeat this procedure with each successive sequence of L (sequences of equal length can be taken in an arbitrary order). Each symbol of C^q is marked only once; that is, if we find a matching sequence in

C^q, we place dots only over those symbols which are still unmarked. This matching procedure finally gives a matrix which has sequences carrying dots over all their symbols as well as sequences that have only some initial segments marked, or no markings at all.

For example, if $q = 3$; the set L contains the four sequences

$$\rho_1;\ \rho_1\rho_2;\ \rho_2\rho_1;\ \rho_2\rho_1\rho_2,$$

and the s-machine has the state diagram of Fig. 3.11 (for its matrix C^3 see Section 3.6), then the matrix sequences are marked as follows:

$$C^3 = \begin{array}{c|ccc} & \alpha_1 & \alpha_2 & \alpha_3 \\ \alpha_1 & [(\dot{\rho}_1\rho_1\dot{\rho}_2,\ \lambda_3\lambda_1\lambda_3) \vee & [(\dot{\rho}_1\dot{\rho}_2\rho_2,\ \lambda_3\lambda_2\lambda_2) \vee & [(\dot{\rho}_1\dot{\rho}_2\rho_1,\ \lambda_3\lambda_2\lambda_1) \vee \\ & \vee (\dot{\rho}_2\dot{\rho}_1\dot{\rho}_2,\ \lambda_1\lambda_1\lambda_3)] & \vee (\rho_2\dot{\rho}_2\dot{\rho}_2,\ \lambda_1\lambda_2\lambda_2) \vee & \vee (\rho_2\rho_2\rho_1,\ \lambda_1\lambda_2\lambda_1)] \\ & & \vee (\dot{\rho}_1\rho_1\rho_1,\ \lambda_3\lambda_1\lambda_2) \vee & \\ & & \vee (\dot{\rho}_2\dot{\rho}_1\rho_1,\ \lambda_1\lambda_1\lambda_2)] & \\ \alpha_2 & (\rho_2\dot{\rho}_1\dot{\rho}_2,\ \lambda_2\lambda_1\lambda_3) & [(\dot{\rho}_1\dot{\rho}_2\rho_1,\ \lambda_1\lambda_3\lambda_3) \vee & [(\rho_2\rho_2\rho_1,\ \lambda_2\lambda_2\lambda_1) \vee \\ & & \vee (\dot{\rho}_1\dot{\rho}_2\rho_2,\ \lambda_1\lambda_3\lambda_1) \vee & \vee (\dot{\rho}_1\rho_1\rho_1,\ \lambda_1\lambda_2\lambda_1)] \\ & & \vee (\rho_2\rho_2\rho_2,\ \lambda_2\lambda_2\lambda_2) \vee & \\ & & \vee (\dot{\rho}_1\rho_1\rho_2,\ \lambda_1\lambda_2\lambda_2) \vee & \\ & & \vee (\dot{\rho}_2\dot{\rho}_1\rho_1,\ \lambda_2\lambda_1\lambda_2)] & \\ \alpha_3 & (\dot{\rho}_1\rho_1\rho_2,\ \lambda_2\lambda_1\lambda_3) & [(\dot{\rho}_2\dot{\rho}_1\dot{\rho}_2,\ \lambda_3\lambda_3\lambda_2) \vee & [(\dot{\rho}_2\dot{\rho}_1\rho_1,\ \lambda_3\lambda_3\lambda_1) \vee \\ & & \vee (\rho_2\rho_2\rho_2,\ \lambda_3\lambda_1\lambda_3) \vee & \vee (\rho_2\rho_2\rho_1,\ \lambda_3\lambda_1\lambda_1) \vee \\ & & \vee (\dot{\rho}_1\dot{\rho}_2\rho_2,\ \lambda_2\lambda_2\lambda_2) \vee & \vee (\dot{\rho}_1\dot{\rho}_2\rho_1,\ \lambda_2\lambda_2\lambda_1)] \\ & & \vee (\dot{\rho}_1\rho_1\rho_1,\ \lambda_2\lambda_1\lambda_2)] & \end{array}$$

After marking, we delete from C^q all those input sequences (together with the corresponding outputs) which do not carry dots (in our example of C^3, these are the sequences $\rho_2\rho_2\rho_2$ and $\rho_2\rho_2\rho_1$). In input sequences where dots appear only over the initial segments, we delete the unmarked symbols, that is, the tail ends. We also chop off the corresponding tail-end sections of the output sequences, and we obtain a *C^q matrix abridged by L.* For example, our C^3 matrix is abridged by L to give

$$\begin{array}{c|ccc} & \alpha_1 & \alpha_2 & \alpha_3 \\ \alpha_1 & (\rho_1,\ \lambda_3) \vee (\rho_2\rho_1\rho_2,\ \lambda_1\lambda_1\lambda_3) & (\rho_1,\ \lambda_3) \vee (\rho_2\rho_1,\ \lambda_1\lambda_1) \vee & (\rho_1\rho_2,\ \lambda_3\lambda_2) \\ & & \vee (\rho_1\rho_2,\ \lambda_3\lambda_2) & \\ \alpha_2 & (\rho_2\rho_1\rho_2,\ \lambda_2\lambda_1\lambda_3) & (\rho_1\rho_2,\ \lambda_1\lambda_3) \vee (\rho_1,\ \lambda_1) \vee & (\rho_1,\ \lambda_1) \\ & & \vee (\rho_2\rho_1,\ \lambda_2\lambda_1) & \\ \alpha_3 & (\rho_1,\ \lambda_2) & (\rho_2\rho_1\rho_2,\ \lambda_3\lambda_3\lambda_2) \vee & (\rho_2\rho_1,\ \lambda_3\lambda_3) \vee \\ & & \vee (\rho_1\rho_2,\ \lambda_2\lambda_2) \vee (\rho_1,\ \lambda_2) & \vee (\rho_1\rho_2,\ \lambda_2\lambda_2) \end{array}$$

Thus the abridged matrix contains only those input sequences (and the corresponding outputs) which are present in L. Now we can

define a simple scheme for recognition of equivalence of states: *Two states x_i and x_j of an s-machine are equivalent in terms of L if, and only if, the pairs of input and output sequences of row i of its abridged (by L) C^q matrix match exactly those of row j and there are no unmatched pairs in either row.* Thus, in the abridged matrix of our example there are no two rows with exactly the same pairs, and therefore this s-machine has no states equivalent to each other in terms of L.

However, even this algorithm, which is an improvement over the disorganized scanning of all possible input-output relationships, is still unwieldy, especially at large values of q. For this reason, one tries to avoid the necessity of scanning all input sequences from L. Instead, one tries to reduce each problem to those particular cases where such scanning is not needed. Let us consider one such case.

Let set L contain all the sequences of length smaller or equal to q. Set L is a subset of set E containing all input sequences. For that reason, any two states equivalent in terms of E (that is, simply equivalent) are also equivalent in terms of L. Now we have to ask ourselves when do groups of states of a given machine, which are equivalent in terms of E, coincide with the groups equivalent in terms of L; that is, when are the states which are equivalent in terms of L also equivalent in terms of E? If these two decompositions into groups coincide, then we can use the Aufenkamp - Hohn algorithm; however, if the groupings do not coincide, we may have to resort to the scanning procedure described above, or to some new method.

The answer to this question is associated with the relationship between the number of states of the machine k, and the maximum length of an allowable input sequence q. We shall show that if q is sufficiently large then it may be possible to recognize all the *non-equivalent* states. Then each pair of states nonequivalent in terms of E will also be nonequivalent in terms of L.

Assume that we are given a sequential machine S with k states, and that we symmetrically decompose its interconnection matrix by means of the Aufenkamp - Hohn method. Now we have k^* groups of equivalent states (obviously, $k^* \leqslant k$).

We shall try to prove that if $q \geqslant k^* - 1$, then the grouping of equivalent states, obtained by the Aufenkamp - Hohn procedure, produces groups which coincide with those equivalent in terms of L; if that is true, then at $q \geqslant k^* - 1$ we can solve the equivalence problem by means of the Aufenkamp - Hohn algorithm, and the number of resulting groups will indeed be equal to k^*.

Let us devise a machine S^* having k^* states and the following characteristics: (a) for each state of machine S there is an equivalent

state of machine S^* and, conversely, for each state of machine S^* there is an equivalent state of machine S; (b) no two states of S^* are equivalent. It will be shown in Section 9.7 that such a machine can always be devised.

We shall now apply Moore's theorem (Section 11.2) which states that if a machine N has k states and all the states are nonequivalent to each other, then for each pair of states $\varkappa_i$ and $\varkappa_j$ there always exists an input sequence not longer than $k-1$ that allows us to differentiate between these two states. Since all the states of S^* are pairwise nonequivalent [see characteristic (b) above], sequences not longer than k^*-1 will differentiate between all the nonequivalent states of this machine. Therefore, if $q \geqslant k^*-1$, all these "differentiating" sequences are contained in L, all states nonequivalent in terms of L can be distinguished, and the Aufenkamp - Hohn algorithm can be used.

However, if $q < k^*-1$, then the grouping in terms of L may not coincide with the grouping with respect to E. In this case one may be forced to resort to the scanning procedure in order to obtain a grouping in terms of L (one way of accomplishing such scanning is the above method of raising matrix C to the power of q).

Sometimes one can avoid the scanning in such cases by estimating the lower bound of the number of states equivalent in terms of L. Thus let us partition matrix C into 1-matrices using only horizontal lines. Then the states of the machine are divided into m groups. These will be groups of states equivalent with respect to set L^1 of all the input sequences of length 1 (set L^1 coincides with the alphabet $\{\rho_1, \rho_2, \ldots, \rho_r\}$). Obviously, the number of groups of states equivalent with respect to L cannot be less than m, since $q \geqslant 1$ and $L^1 \subseteq L$ and, consequently, any two states equivalent with respect to L are also equivalent with respect to L^1. Thus, m is the desired lower bound.

For the same reason, k^* is the upper bound of the number of groups of states which are equivalent in terms of L, since $L \subseteq E$, so that any two simply equivalent states are also equivalent in terms of L.

Thus, if m turns out to be equal to k^* then, despite the fact that $q < k^*-1$, one can use the Aufenkamp - Hohn algorithm.

In the practical application of the Aufenkamp - Hohn algorithm, m and k^* are obtained at different stages of the computation. Thus m is obtained in the first stage, when horizontal lines are drawn to partition matrix C into groups. If, however, the vertical lines drawn subsequently to achieve symmetry "spoil" this grouping, then other horizontal lines must be drawn, and so on, so that ultimately one

one obtains $k^* > m$. Thus one knows immediately whether the Aufenkamp - Hohn algorithm is applicable.

Restricting our discussion of the equivalence problem to the cases described in this and the previous sections, we shall make two brief observations regarding other definitions of the allowable input sequences L.

1. One important case (particularly in the theory of relay-contact circuits) is that where L contains all sequences in which no two identical symbols are repeated consecutively. It can be shown that for this case there exists an algorithm for recognizing equivalent states. However, the present authors know of no algorithm which would be suitable for practical use.

2. If Aufenkamp constraints are operative, then the very statement of the problem must be changed: in this case it makes no sense to talk of two equivalent states $\varkappa_i$ and $\varkappa_j$ since states $\varkappa_i$ and $\varkappa_j$ may allow different sets of input sequences. Here $L_{\varkappa_i}$ may be forbidden in $\varkappa_j$, and vice versa. However, in this case one may sometimes use a concept which is akin to that of equivalence. This is the concept of compatibility of states, which is defined as follows:

Two states—state $\varkappa_i$ of machine S and state ζ_j of machine G—are said to be compatible if, and only if both machines—machine S in initial state $\varkappa_i$ and machine G in initial state ζ_j—having acquired any input sequence from the intersection of set $L_{\varkappa_i}$ with set L_{ζ_j}, will generate identical output sequences (in particular, S and G may be the same machine).* In accordance with this definition, states $\varkappa_i$ and ζ_j must be compatible if that intersection is an empty set, that is, if states $\varkappa_i$ and ζ_j have no allowable input sequences in common. If $L_{\varkappa_i}$ and L_{ζ_j} coincide then, of course, compatibility reduces to equivalence in terms of the common set.

Now, the group of states $\{\varkappa_1, \varkappa_2, \ldots, \varkappa_k\}$ is said to be a *group of pseudoequivalent states* if, and only if, any two states $\varkappa_i$ and $\varkappa_j$ of that group are compatible. This concept is frequently very useful; in particular, it can be applied for minimization of an s-machine which is subject to Aufenkamp constraints (see Section 9.8).

9.5. EQUIVALENCE, MAPPING AND MINIMIZATION OF SEQUENTIAL MACHINES

So far, we discussed the equivalence of individual states; now we shall turn to the equivalence of entire s-machines.

---

*An *intersection* of two sets contains all points belonging to both sets.

Two s-machines, S and G, are said to be equivalent in terms of L if, and only if, for each state $\varkappa_i$ of S there exists at least one state ζ_j of G equivalent to it in terms of L and, conversely, if for each state ζ_s of G there exists at least one state $\varkappa_t$ of S equivalent to it with respect to L.

This definition says any input sequence from L must be allowed both in S and in G. If the set of all sequences allowed in S is L_S, and the analogous set for G is L_G, then L must satisfy the condition

$$L \subseteq L_S \cap L_G,$$

where $L_S \cap L_G$ denotes the intersection of sets L_S and L_G. When $L = E$ (that is, L contains all the possible sequences), we shall say that the machines S and G are ***simply equivalent.*** In this case $L_S = L_G = L = E$.

Machine S maps onto machine G in terms of set L (or G maps S in terms of L) if, and only if, for each state $\varkappa_i$ of S there exists at least one state ζ_j of G equivalent to it in terms of L. If $L = E$, then S ***simply maps*** onto G.

From our definitions of mapping and equivalence we can deduce the following: if machine S maps onto machine G in terms of L, and G maps onto S in terms of the same L, then S and G are machines which are equivalent in terms of L.

The equivalence relationship between S and G is denoted by $S \sim G$, while the mapping of S onto G is written as $S \subset G$.

Equivalent machines are identical as far as processing of input sequences into output sequences is concerned. If machine S maps (or maps in terms of L) onto machine G, then this means that G substitutes for S (however, the converse is not true).

Consider two equivalent s-machines S_1 and S_2, and let their states be partitioned into groups of equivalent states. Now we take some such group s_1^α of S_1 and select any state $\varkappa_i$ from this group. Then S_2 will have a state $\varkappa_j$ equivalent to $\varkappa_i$. Let $\varkappa_j$ belong to the group (of equivalent states) s_2^β of S_2. If that is so, then any state belonging to s_1^α of S_1 is equivalent to any state belonging to s_2^β of S_2. However, none of the states of s_1^α of S_1 is equivalent to any of the states of s_2^γ of S_2, if $\gamma \neq \beta$. Therefore, each group of equivalent states of S_1 corresponds to one and only one group of equivalent states of S_2. The symmetry of the equivalence relationship (it follows from $S_1 \sim S_2$ that $S_2 \sim S_1$), implies that the converse statement is also true, that is, each group of equivalent states of S_2 corresponds to one, and only one group of equivalent states of S_1. Accordingly, the two equivalent machines S_1 and S_2 contain the same number

of groups of equivalent states, and machines S_1 and S_2 differ only in the number of states in each of the corresponding equivalent groups.

If, however, we are given two machines S and G such that G maps S ($S \subset G$), there is no one-to-one correspondence between their groups of equivalent states: all we can say is that to each group of equivalent states of S there corresponds one and only one group of equivalent states of G. However, the converse is not true. Accordingly, G may have more groups of equivalent states than S; thus, machines S and G may differ not only in the number of states in each (equivalent) group, but also in the number of (equivalent) groups.

All of the above also holds if we consider equivalence and mapping in relation to a set L restricted *per se*.

Now let us discuss the minimization of an s-machine S. Minimization of an s-machine S with respect to a set L (of allowable sequences) shall mean finding another s-machine G satisfying these two conditions:

1) G maps S ($G \supset S$) with respect to L.

2) There is no other s-machine mapping S in terms of L and containing fewer states than G.

An s-machine G satisfying these conditions is said to be *minimal for S in terms of L*.

Let us point out that if there exists an algorithm for recognizing states equivalent in terms of L then, in principle, there also exists a trivial minimization algorithm in terms of L. Indeed, if machine S has k states, then the number of internal states in G (which is minimal for S) cannot exceed k. In principle, therefore, we could scan all the machines whose number of states does not exceed k (there is a finite number of such machines). And since there must exist an algorithm for recognizing states equivalent with respect to L, we can check whether each of these machines maps S. Obviously, such a trivial algorithm has no practical value, and we would like to find practical algorithms. So far, such an algorithm exists only for the case where all input sequences are allowed. We shall describe it in the next section.

9.6. MINIMIZATION OF A SEQUENTIAL MACHINE WITH AN UNRESTRICTED SET OF ALLOWABLE INPUT SEQUENCES

Let S be a sequential machine with k internal states decomposed into groups of equivalent states as in Section 9.3. (Figure 9.5 shows a section of the state diagram of this machine.) Consider the first of these groups. Its states are the termini of paths from other states.

In turn, as we have shown in Section 9.3, any state of Group 1 may also be the origin of either of the following paths:

a) a path leading to another state of Group 1. If the first symbol in the label of this path is ρ_p, then all similarly labeled paths, originating in any state of Group 1, must also terminate at a state of this group. The second symbols in the labels of all these paths are identical.

b) a path leading to a state of another group, for example, state $\varkappa_j$ of Group M. If the first symbol in the label of this path is ρ_s, then all similarly labeled paths, originating in any state of Group 1, must also terminate in $\varkappa_j$. The second symbols in the labels of all these paths are identical.

Because they exhibit these characteristics, we can replace all the states of Group 1 by a single state. All the paths to the individual states now terminate in the circle replacing that group. The paths originating in the states of this group will, in case (a), be replaced by a loop labeled (ρ_p, λ_q), and, in the case (b) by a path labeled (ρ_s, λ_t) originating in the new circle and leading to a circle replacing the states of Group M. Figure 9.6 shows such a replacement for the partial state diagram of Fig. 9.5.

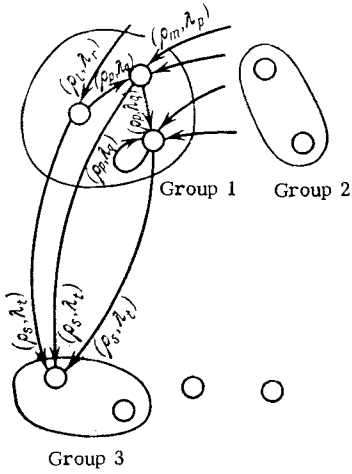


Fig. 9.5.

In the same way, we replace all the other groups of states. As a result, the machine S is transformed into machine G. It is evident that G is equivalent to S.

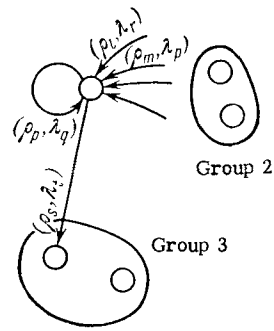


Fig. 9.6.

Indeed, by virtue of characteristics (a) and (b), the output of S at any input sequence and in any initial state $\varkappa_i$ is identical to that which would be generated by G at the same input sequence, provided G was in the initial state $\varkappa_j$ which replaces the group of which $\varkappa_i$ was a member. At the same time, the number of states of machine G is

equal to the number of groups of equivalent states of S, and one cannot further reduce this number by combining these states into groups.

It has been shown in Section 9.5 that all equivalent machines have the same number of equivalent groups, and that the number of such groups in machines mapping such equivalent machines cannot be lower. Thus a minimal machine cannot have fewer states than there are groups of equivalent states in the machine being minimized. *For this reason, machine G is, indeed, minimal for S.* It follows from this that, *in the absence of bounds on the set of input sequences*, (a) the minimal *machine belongs to the class of equivalent machines*, and (b) *the minimization problem is merely one of finding groups of equivalent states, that is, it can be solved by means of the Aufenkamp - Hohn algorithm* (see Section 9.3).

Since this algorithm is used, it is convenient to work with matrix C rather than the state diagram, and replace groups of states by a single one directly in the matrix. For example, consider the matrix C of Section 9.3

$$C = \begin{array}{c|cccccc} & \varkappa_1 & \varkappa_2 & \varkappa_3 & \varkappa_4 & \varkappa_5 & \varkappa_6 \\ \varkappa_1 & (\rho_0, \lambda_1) & (\rho_1, \lambda_2) & 0 & 0 & 0 & (\rho_2, \lambda_1) \\ \varkappa_2 & (\rho_2, \lambda_0) & (\rho_1, \lambda_1) & (\rho_0, \lambda_2) & 0 & 0 & 0 \\ \varkappa_3 & 0 & (\rho_0, \lambda_1) & (\rho_2, \lambda_1) & 0 & (\rho_1, \lambda_2) & 0 \\ \varkappa_4 & 0 & (\rho_2, \lambda_1) & 0 & (\rho_0, \lambda_1) & (\rho_1, \lambda_2) & 0 \\ \varkappa_5 & 0 & 0 & (\rho_0, \lambda_2) & (\rho_2, \lambda_0) & (\rho_1, \lambda_1) & 0 \\ \varkappa_6 & 0 & (\rho_1, \lambda_1) & (\rho_0, \lambda_2) & (\rho_2, \lambda_0) & 0 & 0 \end{array}.$$

By symmetrical decomposition into 1-matrices we obtain

$$C = \begin{array}{c|cc|c|ccc} & \varkappa_1 & \varkappa_4 & \varkappa_3 & \varkappa_2 & \varkappa_5 & \varkappa_6 \\ \varkappa_1 & (\rho_0, \lambda_1) & 0 & 0 & (\rho_1, \lambda_2) & 0 & (\rho_2, \lambda_1) \\ \varkappa_4 & 0 & (\rho_0, \lambda_1) & 0 & (\rho_2, \lambda_1) & (\rho_1, \lambda_2) & 0 \\ \hline \varkappa_3 & 0 & 0 & (\rho_2, \lambda_1) & (\rho_0, \lambda_1) & (\rho_1, \lambda_2) & 0 \\ \hline \varkappa_2 & (\rho_2, \lambda_0) & 0 & (\rho_0, \lambda_2) & (\rho_1, \lambda_1) & 0 & 0 \\ \varkappa_5 & 0 & (\rho_2, \lambda_0) & (\rho_0, \lambda_2) & 0 & (\rho_1, \lambda_1) & 0 \\ \varkappa_6 & 0 & (\rho_2, \lambda_0) & (\rho_0, \lambda_2) & (\rho_1, \lambda_1) & 0 & 0 \end{array}.$$

Thus we have three groups of equivalent states, $\{\varkappa_1, \varkappa_4\}$, $\{\varkappa_3\}$, and $\{\varkappa_2, \varkappa_5, \varkappa_6\}$.

The replacement of groups of states of a state diagram by a single state is equivalent to the replacement of each 1-matrix of the symmetrical decomposition by a single element, which is a disjunction (union) of all the elements of the 1-matrix being replaced.

In our example, this replacement will give the following interconnection matrix for the minimal machine G:

$$C = \begin{array}{c} \\ \varkappa_1' \\ \varkappa_2' \\ \varkappa_3' \end{array} \begin{array}{c} \begin{array}{ccc} \varkappa_1' & \varkappa_2' & \varkappa_3' \end{array} \\ \begin{bmatrix} (\rho_0, \lambda_1) & 0 & (\rho_1, \lambda_2) \vee (\rho_2, \lambda_1) \\ 0 & (\rho_2, \lambda_1) & (\rho_0, \lambda_1) \vee (\rho_1, \lambda_2) \\ (\rho_2, \lambda_0) & (\rho_0, \lambda_2) & (\rho_1, \lambda_1) \end{bmatrix} \end{array}$$

Its state diagram is shown in Fig. 9.7.

So far we have dealt with the minimization of an s-machine whose set of input sequences is infinite. The problem of minimization of an s-machine in which *per se* restrictions are operative is tied to the still unsatisfactorily solved problem of finding groups of states equivalent in terms of L for the same case (see Section 1 and Section 9.4).

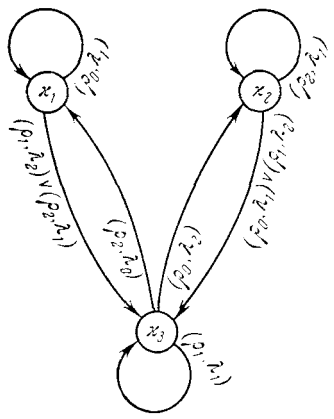


Fig. 9.7.

In addition, minimization with respect to $L \neq E$ is associated with the following additional difficulty, which would exist even if we had an algorithm for finding groups of states equivalent in terms of L. Thus, earlier in this section we were able to replace a group of states by a single state by using properties of the paths in the state diagram (see p. 238). However, if $L \neq E$, then, generally speaking, the paths do not possess the properties specified in (a) and (b), p. 238. Thus two paths, the labels of which contain identical first symbols and originating in states which are equivalent in terms of L may terminate in states which are nonequivalent in terms of L. Consider, for example, the section of the state diagram (Fig. 9.8) for the case where L does not contain any sequence with two consecutive identical symbols. Let states κ_1 and κ_2 be equivalent in terms of L, that is, belong to one group. Further, let states κ_3 and κ_4 be equivalent in terms of L' which contains all the sequences of L except those beginning with the symbol ρ_s, and let κ_3 and κ_4 be nonequivalent in terms of L since they generate different output symbols at those sequences from L which begin with ρ_s. Then the paths of Fig. 9.8 do not contradict the equivalence of κ_1 and κ_2 in terms of L (for L does not contain any sequences beginning with two consecutive symbols ρ_s), but they do contradict condition (b) of p. 238.

It follows from the foregoing that at $L \neq E$ the states belonging to one group of equivalent states cannot, generally speaking, be replaced by a single state. If this were done, two paths labeled with an identical first symbol ρ_s would originate in the same new state and lead to two different states—a condition which contradicts the very definition of an s-machine.

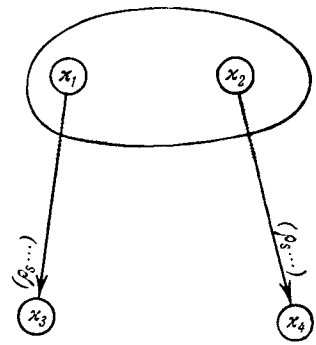


Fig. 9.8.

So far we have not imposed any restrictions on the processing of sequences by the s-machine which is being minimized. However, in the next section we shall consider a special case where the s-machine operates as a finite automaton.

9.7. MINIMIZATION OF A SEQUENTIAL MACHINE WHEN IT OPERATES AS A FINITE AUTOMATON

Assume that we are given the basic table of a finite automaton whose states are coded in symbols from the alphabet $\{\varkappa_1, \varkappa_2, \ldots, \varkappa_h\}$. Let us recode the symbols, replacing all $\varkappa_i$ by λ_i; we shall assume that $l = k$, where l is the number of symbols λ. Obviously, the basic table of the finite automaton now contains λ's with the subscripts of the $\varkappa$'s they replace.

It is required to devise a minimal sequential machine which would realize this automaton, that is, would process inputs into outputs in the same way as the automaton. The set of allowable input sequences may be restricted or unrestricted.

We shall consider two cases: a case where there are no restrictions on the input sequences, and a case where the input sequences may not contain two consecutive identical symbols.

Case 1. Set L Contains All Possible Sequences (that is, $L = E$)

We shall analyze this case on an automaton A given in the form of Table 9.1.

We have shown in Section 9.6 that, in the absence of restrictions on the input sequences, the minimal s-machine for a given machine N belongs to the class of machines equivalent to N, and that none of

its states has other equivalents. Consequently, our required minimal s-machine must also be equivalent to automaton A.

Table 9.1

λ \ ρ	ρ_1	ρ_2	ρ_3
λ_1	λ_4	λ_3	λ_2
λ_2	λ_1	λ_2	λ_4
λ_3	λ_2	λ_1	λ_1
λ_4	λ_4	λ_3	λ_2

Let us first construct an s-machine which is not minimal, but which is suitable for further minimization. It will have as many states as there are rows in the automaton of Table 9.1, and its state diagram (Fig. 9.9) has as many circles, numbered consecutively 1 - 4. We draw paths between these circles as per the Table 9.1. Each path has a label whose first symbol is the subscript of the corresponding ρ from the table, while the second symbol is the number of the circle in which the path terminates. Since this is a diagram of some s-machine, we replace the numbers in each circle by symbols $\varkappa_i$ and the numbers (m, n) in the paths

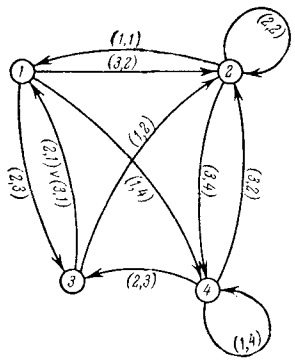


Fig. 9.9.

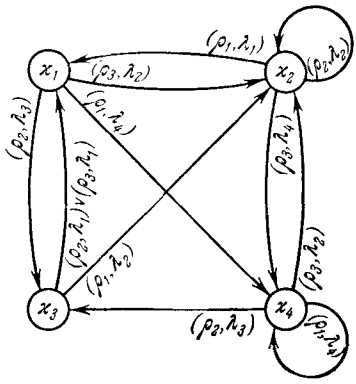


Fig. 9.10.

labeled by symbols (ρ_m, λ_n); this gives Fig. 9.10. Now we derive the interconnection matrix of this machine:

$$C = \begin{array}{c} \\ \varkappa_1 \\ \varkappa_2 \\ \varkappa_3 \\ \varkappa_4 \end{array} \begin{array}{c} \begin{array}{cccc} \varkappa_1 & \varkappa_2 & \varkappa_3 & \varkappa_4 \end{array} \\ \begin{bmatrix} 0 & (\rho_3, \lambda_2) & (\rho_2, \lambda_3) & (\rho_1, \lambda_4) \\ (\rho_1, \lambda_1) & (\rho_2, \lambda_2) & 0 & (\rho_3, \lambda_4) \\ (\rho_2, \lambda_1) \vee (\rho_3, \lambda_1) & (\rho_1, \lambda_2) & 0 & 0 \\ 0 & (\rho_3, \lambda_2) & (\rho_2, \lambda_3) & (\rho_1, \lambda_4) \end{bmatrix} \end{array}.$$

By analogy with the above state diagram, all the nonzero elements of the interconnection matrix belonging to the same column have identical subscripts on the λ symbol. These subscripts coincide

with the number of the column. This matrix C can be transformed into the interconnection matrix C' of an equivalent minimal s-machine, which therefore is a minimal s-machine operating in the same way as automaton A.

First we decompose C into 1-matrices by means of horizontals only. We get a 1-matrix from rows 1 and 4. We transpose these rows and get

$$C = \begin{array}{c|cccc} & \varkappa_1 & \varkappa_4 & \varkappa_2 & \varkappa_3 \\ \varkappa_1 & 0 & (\rho_1, \lambda_4) & (\rho_3, \lambda_2) & (\rho_2, \lambda_3) \\ \varkappa_4 & 0 & (\rho_1, \lambda_4) & (\rho_3, \lambda_2) & (\rho_2, \lambda_3) \\ \hline \varkappa_2 & (\rho_1, \lambda_1) & (\rho_3, \lambda_4) & (\rho_2, \lambda_2) & 0 \\ \hline \varkappa_3 & (\rho_2, \lambda_1) \vee (\rho_3, \lambda_1) & 0 & (\rho_1, \lambda_2) & 0 \end{array}.$$

Now we draw horizontals between rows κ_4 and κ_2, and between rows κ_2 and κ_3, and obtain three 1-matrices whose columns contain either zeros or identical pairs [for example, the two-row matrix on top has only zeros in column 1, only pairs (ρ_1, λ_4) in column 2, and so on]. This is the result of the previously mentioned property of C: the second digits are the same in each column of C. But in 1-matrices, where the columns contain only identical pairs, the first digits of each column will also coincide. If this is so, then all we need to do in order to form groups of equivalent states to partition C into 1-matrices by horizontals only: since the elements in the columns of each 1-matrix coincide, vertical lines cannot "spoil" this symmetrical grouping.

This property, in turn, means the following: the groups of states of an s-machine (with matrix C) which are simply equivalent, and those which are equivalent in terms of set L_1 comprising all input sequences of length 1, coincide. Therefore, to find all the groups of equivalent states, it is sufficient to partition C into groups of states equivalent in terms of L_1, a partition achieved simply by decomposing C into 1-matrices by means of horizontal lines.

In our example, state $\varkappa_1$ of matrix C is equivalent to state $\varkappa_4$. To construct a minimal s-machine, we replace these two states with a single state $\varkappa_1$. Then we draw verticals between columns 2 and 3 and 3 and 4 of C. In this symmetrical decomposition we replace each newly generated 1-matrix by the union

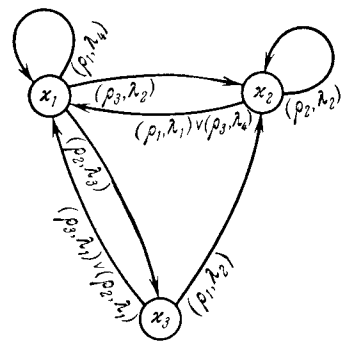


Fig. 9.11.

of all its elements, and obtain the interconnection matrix C' of the minimal s-machine:

$$C' = \begin{array}{c} \\ \varkappa_1 \\ \varkappa_2 \\ \varkappa_3 \end{array} \begin{array}{c} \begin{array}{ccc} \varkappa_1 & \varkappa_2 & \varkappa_3 \end{array} \\ \begin{bmatrix} (\rho_1, \lambda_4) & (\rho_3, \lambda_2) & (\rho_2, \lambda_3) \\ (\rho_1, \lambda_1) \vee (\rho_3, \lambda_4) & (\rho_2, \lambda_2) & 0 \\ (\rho_2, \lambda_1) \vee (\rho_3, \lambda_1) & (\rho_1, \lambda_2) & 0 \end{bmatrix} \end{array}$$

The corresponding state diagram is shown in Fig. 9.11.

Note also that for each set of identical rows of matrix C (in our case, rows 1 and 4) there always exists a set of identical rows in the automaton table (here, rows 1 and 4 of Table 9.1), and vice versa. Consequently, inspection of the automaton table immediately shows the number of states of a minimal s-machine realizing this automaton (one needs only to count the number of differing rows in the table of the automaton).

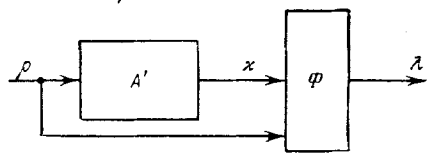


Fig. 9.12.

Having the state diagram, we can compile the table of the automaton A' and converter Φ which comprise the minimal s-machine operating as automaton A in accordance with Fig. 9.12. Our state diagram (Fig. 9.10) thus yields Tables 9.2 and 9.3.

Table 9.2

$\varkappa$ \ ρ	ρ_1	ρ_2	ρ_3
$\varkappa_1$	$\varkappa_1$	$\varkappa_3$	$\varkappa_2$
$\varkappa_2$	$\varkappa_1$	$\varkappa_2$	$\varkappa_1$
$\varkappa_3$	$\varkappa_2$	$\varkappa_1$	$\varkappa_1$

Table 9.3

$\varkappa$ \ ρ	ρ_1	ρ_2	ρ_3
$\varkappa_1$	λ_4	λ_3	λ_2
$\varkappa_2$	λ_1	λ_2	λ_4
$\varkappa_3$	λ_2	λ_1	λ_1

Again, the table of automaton A' (Table 9.2) could have been obtained directly from the table of automaton A (Table 9.1) merely by deleting one of the identical rows (the fourth; if there are several such rows,

all but one are deleted), and then replacing throughout the remainder of the table those symbols which are the same as the heading(s) of the deleted row(s) [λ_4 in our example] by the heading of the retained row (in our case, we replace λ_4 by λ_1).

The converter table can also be obtained directly from the automaton table. Again we delete superfluous identical rows of Table 9.1 (row 4), and in the remaining table substitute $\varkappa_i$'s for λ_i's in all row headings.

Thus we have a simple, straightforward algorithm for direct derivation of the tables of automaton A' and converter Φ which, in accordance with the scheme of Fig. 9.12, constitute the minimal s-machine realizing automaton A. The state diagram and the interconnection matrix were only necessary for proving the validity of this algorithm.

Case 2. Set L Has No Sequences Comprising Two Consecutive Identical Symbols

If L_1 is the set of input sequences of length 1, and E is the set containing all possible input sequences, then obviously we shall have the following relationship:

$$L_1 \subset L \subset E. \tag{9.2}$$

If the number of groups of equivalent states is m^*, the numbers of groups of states equivalent in terms of L and L_1 are, respectively, m and m^{**}, then by virtue of (9.2)

$$m^{**} \leqslant m \leqslant m^*.$$

In Case 1 we have shown that the groupings of equivalent states and of states equivalent in terms of L_1 coincide. Consequently, $m^{**} = m^*$, and from (9.2) we get

$$m^{**} = m = m^*.$$

For this reason states equivalent in terms of L_1 will also be equivalent in terms of E in this case. Therefore one can minimize the numbers of states by replacing each group by a single state, using the above method where it was assumed that $L = E$. Thus minimal s-machines for sets E and L coincide in Case 2, and the minimization proceeds as if there were no restrictions on the input sequences.

9.8. MINIMIZATION OF MACHINES IN THE CASE OF AUFENKAMP-TYPE CONSTRAINTS

The obvious approach to the minimization problem in this case is as follows.

Let N be an s-machine with k states, subject to arbitrary Aufenkamp-type constraints. We shall say that to minimize N means devising a new machine P with a minimal number of states such that for each state $\varkappa_i$ of N there is at least one state χ_j of P. States $\varkappa_j$ must satisfy the following conditions:

a) Any input sequence allowed in $\varkappa_i$ of N is allowed in χ_j of P.

b) If N is in state $\varkappa_i$ and P is in state χ_j, and if some arbitrary sequence from the set of input sequences allowed in N when in the state $\varkappa_i$ is fed to both machines, then both will convert it into identical output sequences.

We shall say that a *machine* P (which need not necessarily be minimal) *satisfying conditions (a) and (b) realizes a pseudomapping of machine* N. Thus P "can do" whatever N can. That is, it can take any input sequence allowed in N and process it into the same output sequence.

We shall now describe an Aufenkamp algorithm resulting in a machine P which is a pseudomapping of machine N and has fewer states than N, but is not necessarily minimal.

On a state diagram, the presence of Aufenkamp-type constraints manifest itself in that the number of paths originating at some circles is smaller than that of various inputs $\rho_1, \rho_2, \ldots, \rho_r$. This means that the machine cannot respond to some inputs when it is in certain states.

The state diagram, in turn, is the starting point for the construction of the interconnection matrix. Again, the effect of the constraints on that matrix is that the latter may contain rows with fewer symbol pairs than there are inputs $\rho_1, \rho_2, \ldots, \rho_r$. For example, consider the matrix

$$C = \begin{array}{c} \\ \varkappa_1 \\ \varkappa_2 \\ \varkappa_3 \\ \varkappa_4 \\ \varkappa_5 \end{array} \begin{array}{c} \begin{array}{ccccc} \varkappa_1 & \varkappa_2 & \varkappa_3 & \varkappa_4 & \varkappa_5 \end{array} \\ \begin{bmatrix} 0 & (\rho_1, \lambda_0) & (\rho_3, \lambda_2) & 0 & 0 \\ (\rho_2, \lambda_1) & 0 & 0 & (\rho_3, \lambda_2) & 0 \\ 0 & (\rho_3, \lambda_2) & 0 & 0 & (\rho_1, \lambda_0) \\ 0 & 0 & (\rho_2, \lambda_2) & 0 & 0 \\ (\rho_2, \lambda_2) & 0 & (\rho_1, \lambda_3) & 0 & 0 \end{bmatrix} \end{array}.$$

The first row lacks a symbol pair incorporating the input ρ_2, the second lacks the pair associated with ρ_1, and the fourth contains only a single pair (that associated with ρ_2).

We shall say that a submatrix of an interconnection matrix C is a generalized 1-matrix if it has the following property: *if any row of the generalized 1-matrix contains a pair* (ρ_m, λ_n), *then none of the remaining rows of that matrix will contain pairs in which this input symbol* ρ_m *is associated with a different symbol* λ.

We shall cite here, without proof, the following theorem of Aufenkamp [5]: *Assume the interconnection matrix C is decomposed by horizontal lines into groups of rows constituting generalized 1-matrices, and is then further partitioned by vertical lines to achieve a symmetrical decomposition into generalized 1-matrices. Provided no two generalized 1-matrices of a given group contain the same input symbol* ρ_m, *the states of this group are pseudoequivalent.* Thus, machine N can be minimized by replacing each group of pseudoequivalent states by a single state. This is done by replacing each generalized 1-matrix of a symmetrical decomposition by one term which represents a union (disjunction) of all the elements of the 1-matrix being replaced. This gives a matrix C' of machine P which realizes a pseudomapping of N and has fewer states provided, of course, that the symmetrical decomposition of C is nontrivial.

To illustrate, let N have the state diagram of Fig. 9.13, with matrix C shown above (p. 246). First, we draw horizontals to decompose C into generalized 1-matrices. In contrast to the case where there were no restrictions and there was only one way of partitioning C into 1-matrices, now we have several possibilities. For example, the rows of C may be divided into three groups, the first comprising the rows 1 and 4, the second — rows 2 and 3, and the third—row 5. However, we can also divide C into two groups, the first comprising 1, 2 and 3, and the second—rows 4 and 5. It is important to realize that the mode of partitioning will definitely affect the possibility of minimizing N. For example, let us partition C in the most economic way, that is, into the two groups discussed above. We thus draw a horizontal between rows 3 and 4 and obtain two generalized 1-matrices. Then, for symmetry, we draw a vertical between columns 3 and 4. This "spoils" our decomposition because the top group of rows now contains two generalized 1-matrices, each containing ρ_1, and ρ_3 in violation of the above-cited theorem of Aufenkamp. The states of the group are thus not pseudoequivalent. To remedy this situation, we draw horizontals between rows 1 and 2, and 2 and 3

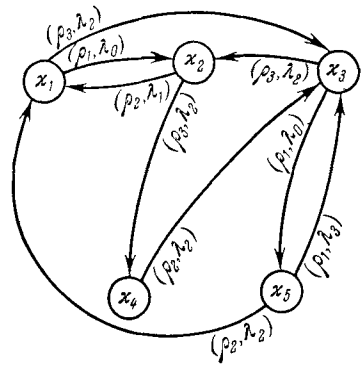


Fig. 9.13.

(in the general case, there are several possibilities for achieving such adjustments), so that we now have four groups, none of which contains two generalized 1-matrices with the same symbol ρ. But, for symmetry, we must also draw verticals between columns 1 and 2, and 2 and 3. This again spoils the decomposition because the group comprising rows 4 and 5 now has two generalized 1-matrices, each containing a pair with ρ_2. We are therefore forced to draw a horizontal between rows 4 and 5, and a corresponding vertical between the columns. Obviously this decomposition is trivial, and thus the machine cannot be minimized in this way.

Assume, however, that we start with a seemingly less economical partition of C, that is, the one comprising three groups discussed above (group 1 comprises rows 1 and 4, group 2—rows 2 and 3, and group 3—row 5). We thus rewrite C as follows:

$$C = \begin{array}{c|cc|cc|c} & \kappa_1 & \kappa_4 & \kappa_2 & \kappa_3 & \kappa_5 \\ \kappa_1 & 0 & 0 & (\rho_1, \lambda_0) & (\rho_3, \lambda_2) & 0 \\ \kappa_4 & 0 & 0 & 0 & (\rho_2, \lambda_2) & 0 \\ \hline \kappa_2 & (\rho_2, \lambda_1) & (\rho_3, \lambda_2) & 0 & 0 & 0 \\ \kappa_3 & 0 & 0 & (\rho_3, \lambda_2) & 0 & (\rho_1, \lambda_0) \\ \hline \kappa_5 & (\rho_2, \lambda_2) & 0 & 0 & (\rho_1, \lambda_3) & 0 \end{array}.$$

Here we have only one conflict—two matrices of group 2 contain ρ_3. This is easily fixed by drawing a horizontal between rows κ_2 and κ_3, and a corresponding vertical between the columns. We now have one group of pseudoequivalent states comprising more than one row, that is, group 1. This group can be replaced by a single state, giving the matrix of a somewhat minimized machine P' (it has four states vs the five of N) which realizes the pseudomapping of N.

Finally, we could partition C into the following three groups: group 1 comprising rows 1 and 2, group 2—rows 3 and 4, and group 3—row 5. This immediately yields a symmetrical decomposition which needs no "fixing":

$$C = \begin{array}{c|cc|cc|c} & \kappa_1 & \kappa_2 & \kappa_3 & \kappa_4 & \kappa_5 \\ \kappa_1 & 0 & (\rho_1, \lambda_0) & (\rho_3, \lambda_2) & 0 & 0 \\ \kappa_2 & (\rho_2, \lambda_1) & 0 & 0 & (\rho_3, \lambda_2) & 0 \\ \hline \kappa_3 & 0 & (\rho_3, \lambda_2) & 0 & 0 & (\rho_1, \lambda_0) \\ \kappa_4 & 0 & 0 & (\rho_2, \lambda_2) & 0 & 0 \\ \hline \kappa_5 & (\rho_2, \lambda_2) & 0 & (\rho_1, \lambda_3) & 0 & 0 \end{array}.$$

As a result, the pseudomapping of N is realized by a machine P'' which has only three states and a matrix C'':

$$C'' = \begin{array}{c} \\ \kappa_1 \\ \kappa_2 \\ \kappa_3 \end{array} \begin{array}{c} \begin{array}{cccc} \kappa_1 & & \kappa_2 & \kappa_3 \end{array} \\ \begin{bmatrix} (\rho_1, \lambda_0)\ (\rho_2, \lambda_1) & (\rho_3, \lambda_2) & 0 \\ (\rho_3, \lambda_2) & (\rho_2, \lambda_2) & (\rho_1, \lambda_0) \\ (\rho_2, \lambda_2) & (\rho_1, \lambda_3) & 0 \end{bmatrix} \end{array}.$$

Its state diagram is shown in Fig. 9.14.

Thus to achieve optimum results in applying this algorithm one must try out all the possibilities for symmetrical decomposition of C into generalized 1-matrices.* Furthermore, this algorithm does not necessarily yield a minimal machine P: the matrix C of machine N may be decomposed into groups of pseudoequivalent states replaceable by a single state thereby minimizing it without fulfilling the conditions of the second Aufenkamp theorem. For example, consider the machine of Fig. 9.15. Here, state $\varkappa_1$ does not admit an input ρ_1. The corresponding interconnection matrix C is

$$C = \begin{array}{c} \\ \varkappa_1 \\ \varkappa_2 \\ \varkappa_3 \end{array} \begin{array}{c} \begin{array}{ccc} \varkappa_1 & \varkappa_2 & \varkappa_3 \end{array} \\ \begin{bmatrix} (\rho_2, \lambda_0) & (\rho_3, \lambda_0) & 0 \\ (\rho_1, \lambda_0) & (\rho_3, \lambda_0) & (\rho_2, \lambda_0) \\ (\rho_3, \lambda_0) & (\rho_2, \lambda_0) & (\rho_1, \lambda_1) \end{bmatrix} \end{array}.$$

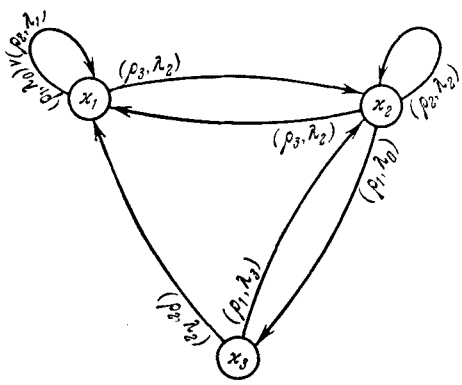


Fig. 9.14.

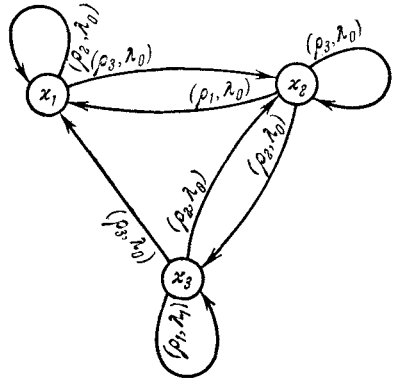


Fig. 9.15.

*Note that this algorithm is tantamount to the scanning of all the possible additional definitions of this s-machine, that is, to the scanning of all the possibilities for drawing missing paths on the state diagram, with subsequent minimization of machines so obtained without any restrictions on the input sequence.

There are only two ways in which C could be symmetrically partitioned into generalized 1-matrices:

$$C = \begin{array}{c} \\ \varkappa_1 \\ \varkappa_2 \\ \varkappa_3 \end{array} \begin{array}{c} \begin{array}{cc} \varkappa_1 & \varkappa_2 \end{array} \\ \left[\begin{array}{cc|c} (\rho_2, \lambda_0) & (\rho_3, \lambda_0) & 0 \\ (\rho_1, \lambda_0) & (\rho_3, \lambda_0) & (\rho_2, \lambda_0) \\ \hline (\rho_3, \lambda_0) & (\rho_2, \lambda_0) & (\rho_1, \lambda_1) \end{array}\right] \end{array},$$

$$C = \begin{array}{c} \\ \varkappa_1 \\ \varkappa_3 \\ \varkappa_2 \end{array} \begin{array}{c} \begin{array}{cc} \varkappa_1 & \varkappa_3 \end{array} \\ \left[\begin{array}{cc|c} (\rho_2, \lambda_0) & 0 & (\rho_3, \lambda_0) \\ (\rho_3, \lambda_0) & (\rho_1, \lambda_1) & (\rho_2, \lambda_0) \\ \hline (\rho_1, \lambda_0) & (\rho_2, \lambda_0) & (\rho_3, \lambda_0) \end{array}\right] \end{array}.$$

Obviously, neither decomposition satisfies the second Aufenkamp theorem: in the first case, the two top generalized 1-matrices contain the same pair (ρ_2, λ_0), while in the second case the common pairs are (ρ_2, λ_0) and (ρ_3, λ_0). Nevertheless this machine can be minimized, the corresponding minimal s-machine (two states) being that of Fig. 9.16. Here state $\varkappa_A$ of the pseudomapping corresponds to states $\varkappa_1$ and $\varkappa_2$ of the original machine, while state $\varkappa_B$ is equivalent to state $\varkappa_3$.*

Now we shall describe a method devised by Gill [149], which yields a minimal machine for any given s-machine subject to Aufenkamp-type constraints. This method requires, as a first step, that all pairs of compatible states of the given machine be determined. There are methods for determining the compatibility of states, but we shall describe only one.

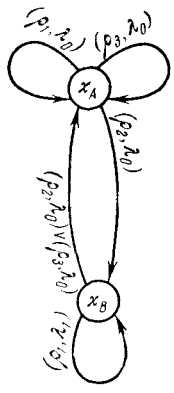


Fig. 9.16.

Suppose, for example, that we want to find out whether the ith and the jth states of a given machine are compatible. To achieve this, we construct a "tree" from the common starting point (i, j). Its branches correspond to the inputs which are common to states $\varkappa_i$ and $\varkappa_j$. If this procedure yields different outputs even for a single input, then we immediately know that states $\varkappa_i$ and $\varkappa_j$ are not compatible. If, however, the states prove compatible, then we write over the branches the corresponding input-output pairs, and at their ends the pairs of states into which $\varkappa_i$ and $\varkappa_j$ are shifted by these inputs. Each pair of such states then serves as the starting node for another branch of the tree. During

*This example also shows that no additional definition of the machine of Fig. 9.15 yields a minimal machine with two states: for no path containing ρ_1 in the label and originating at $\varkappa_1$ will generate equivalent states, regardless of what λ is in the label.

this construction we keep on crossing out nodes if:

1. The same node (that is, a node with a label consisting of the same symbols) has already been encountered anywhere else on the tree.

2. The node label consists of two identical digits, that is, if an input shifts both states $\varkappa_i$ and $\varkappa_j$ into the same state $\varkappa_m$.

3. No new branches can be drawn from this node, that is, the states corresponding to that node have no common inputs.

We stop either if states $\varkappa_i$ and $\varkappa_j$ are incompatible or if all the paths of the tree lead to crossed out nodes. In the latter case, we may conclude that the pairs of states corresponding to all the nodes (crossed out or otherwise) of the tree are also compatible.

For example, Fig. 9.17 illustrates the tree for states $\varkappa_1$ and $\varkappa_2$ of the machine of Fig. 9.15, while Fig. 9.19 shows the tree for states $\varkappa_2$ and $\varkappa_3$ of the machine of Fig. 9.18. In either case, the tested states prove to be compatible; in the first case, we also have another pair of compatible states ($\varkappa_1$ and $\varkappa_3$), in addition to $\varkappa_1$ and $\varkappa_2$, whereas in the second case, $\varkappa_4$ and $\varkappa_6$, $\varkappa_3$ and $\varkappa_5$, and $\varkappa_1$ and $\varkappa_2$ turn out to be compatible, in addition to $\varkappa_2$ and $\varkappa_3$.

This procedure also yields an estimate of the maximum number of steps (the worst case) needed to determine the compatibility of two states of a machine with n states. This estimate is obtained from*

$$l = \frac{n^2 - n}{2}.$$

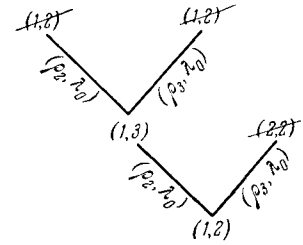


Fig. 9.17.

Now we can determine all the pairs of compatible states for any machine. The machine of Fig. 9.15 has two such pairs [$\{\varkappa_1, \varkappa_2\}$ and $\{\varkappa_1, \varkappa_3\}$], while that of Fig. 9.18 has nine $\{\varkappa_1, \varkappa_2\}$, $\{\varkappa_1, \varkappa_3\}$, $\{\varkappa_1, \varkappa_5\}$, $\{\varkappa_2, \varkappa_3\}$, $\{\varkappa_2, \varkappa_4\}$, $\{\varkappa_2, \varkappa_5\}$, $\{\varkappa_3, \varkappa_5\}$, $\{\varkappa_3, \varkappa_6\}$, $\{\varkappa_4, \varkappa_6\}$. These pairs can now be divided into groups of pseudoequivalent states. For example, the preceding list contains pairs $\{\varkappa_1, \varkappa_2\}$, $\{\varkappa_1, \varkappa_3\}$ and $\{\varkappa_2, \varkappa_3\}$, so that states $\varkappa_1$, $\varkappa_2$ and $\varkappa_3$ form a group of pseudoequivalent states $\{\varkappa_1, \varkappa_2, \varkappa_3\}$.**

Following this line of reasoning, we can divide the states of the machine into a minimal number of groups of pseudoequivalent states.

---

*This estimate is accurate, since there are cases where l is attained. It is interesting to compare this estimate with that for the number of steps necessary to recognize the nonequivalence of states in a machine without restrictions (see Section 11.2); in this case, $l = (n - 1)$, that is, the n^2 terms do not appear.

**Remember that a group of pseudoequivalent states is one in which all the states are pairwise compatible.

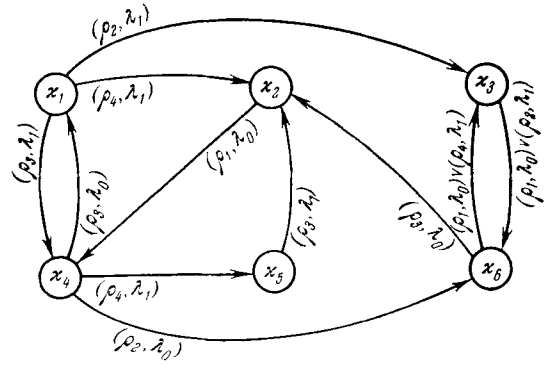


Fig. 9.18.

In our example, there will be four such groups:

$$\{x_1, x_2, x_3, x_5\}, \{x_2, x_4\}, \{x_3, x_6\}, \{x_4, x_6\}.$$

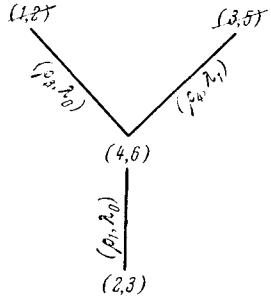


Fig. 9.19.

In the general case (just as in our example), these groups intersect.

Let us now return to the minimization of an s-machine subject to Aufenkamp-type constraints. Assume an arbitrary s-machine with n states, and assume that at least one minimal machine $S_{\min}$ with k states can be constructed for it. If $k < n$, then at least one state of $S_{\min}$ must pseudomap two or more states of S. Assume that Σ_i is the set of all states of S which correspond to state x_i of $S_{\min}$, and assume that such sets of states $\Sigma_1, \Sigma_2, \ldots, \Sigma_k$ of S can be assembled for all the states $x_1, x_2, \ldots, x_k$ of $S_{\min}$. Such grouping of states of S has the following properties:

1. The grouping $\Sigma_1, \Sigma_2, \ldots, \Sigma_k$ embraces all the states of S, that is, each state of S belongs to at least one set Σ.

2. States belonging to any one set Σ_i are pseudoequivalent.

3. All states of a given group Σ_i which allow a given input ρ_s, are shifted by it into states of the same new group Σ_j (in particular, i can be equal to j).

While the first two of these properties are obvious, the third requires an explanation. For example, let state x_i of $S_{\min}$ belong to Σ_i, and let state x_j, into which x_i is shifted by input ρ_s, belong to group Σ_j. Now assume that there exists a state of Σ_i which is shifted by ρ_s into a state x_l not belonging to Σ_j. Then observation of the

behavior of S and $S_{\min}$ at all input sequences allowed for states of Σ_i leads to the following conclusions: (a) any input allowed in state $\varkappa_l$ of S is also allowed in state $\varkappa_j$ of $S_{\min}$; and (b) if the input is a sequence allowed in state $\varkappa_l$, then both machines (that is, S starting from $\varkappa_l$, and $S_{\min}$ starting from $\varkappa_j$) will generate identical output sequences. But this simply means that state $\varkappa_l$ corresponds to state $\varkappa_j$ under pseudomapping and consequently it must, contrary to our initial assumption, belong to group Σ_j, so that the third property must hold.

The grouping $\Sigma_1, \Sigma_2, \ldots, \Sigma_k$ is known as the *specific grouping of states of machine S.*

It follows from the foregoing that the minimization algorithm involves finding a minimal specific grouping of states of S (of which there may be one or more), and the subsequent replacement of each group Σ_i by a single state. Since all states of each group are pseudoequivalent, any group Σ_i must belong to (or coincide with) some group of the minimal decomposition of the states of S into groups of pseudoequivalent states. Thus, this minimization algorithm consists of scanning of all various possible specific groupings of S in the search for the minimal one—a very cumbersome procedure. However, there are algorithms for organizing this scanning to reduce waste motions (see, for example, [149]). There are also "intermediate" algorithms which, while reducing the amount of scanning required, give better results than Aufenkamp's algorithm, even though they do not assume minimality.

Let us now return to our two examples (Figs. 9.15 and 9.18).

For the machine of Fig. 9.15, the grouping into pseudoequivalent states $\{\varkappa_1, \varkappa_2\}$ and $\{\varkappa_1, \varkappa_3\}$ is also the minimal specific grouping. We shall prove this.

Let us code group $\{\varkappa_1, \varkappa_2\}$ by A, and group $\{\varkappa_1, \varkappa_3\}$ by B. Now let us trace the possible results of various inputs:

A	ρ_1	$\varkappa_1$ does not allow ρ_1, $\varkappa_2$ shifts to $\varkappa_1$,	A or B
	ρ_2	$\varkappa_1$ shifts to $\varkappa_1$, $\varkappa_2$ shifts to $\varkappa_3$,	B
	ρ_3	$\varkappa_1$ shifts to $\varkappa_2$, $\varkappa_2$ shifts to $\varkappa_2$,	A
B	ρ_1	$\varkappa_1$ does not allow ρ_1, $\varkappa_3$ shifts to $\varkappa_3$,	B
	ρ_2	$\varkappa_1$ shifts to $\varkappa_1$, $\varkappa_3$ shifts to $\varkappa_2$,	A
	ρ_3	$\varkappa_1$ shifts to $\varkappa_2$, $\varkappa_3$ shifts to $\varkappa_1$,	A

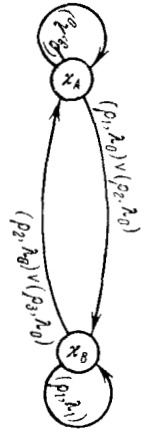


Fig. 9.20.

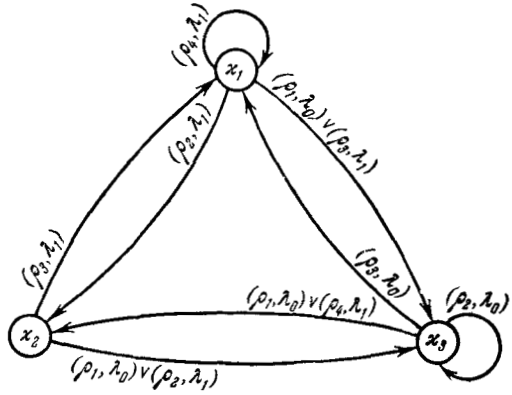


Fig. 9.21.

Now we readily construct two minimal machines for the machine of Fig. 9.15, replacing states of A by a state $\varkappa_A$ and those of B by $\varkappa_B$ (Figs. 9.16 and 9.20). For the machine of Fig. 9.18 there also are

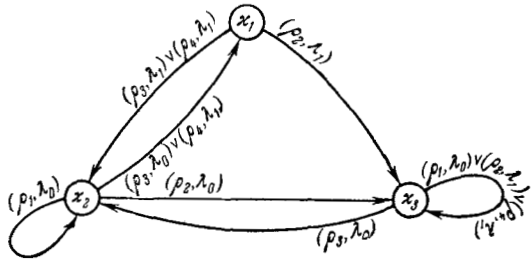


Fig. 9.22.

two possible minimal specific groupings: $\{\varkappa_1, \varkappa_2\}$, $\{\varkappa_3, \varkappa_5\}$, and $\{\varkappa_4, \varkappa_6\}$, or $\{\varkappa_1, \varkappa_5\}$, $\{\varkappa_2, \varkappa_4\}$, and $\{\varkappa_3, \varkappa_6\}$. The state diagram of the minimal machine corresponding to the first of these is shown in Fig. 9.21, while that of the second one is represented in Fig. 9.22.

9.9. ANOTHER DEFINITION OF EQUIVALENCE OF SEQUENTIAL MACHINES

Sometimes one encounters in the literature a definition of equivalence of sequential machines which differs from that of Section 9.4.

Outwardly, that definition appears similar to ours, but in reality there is a vast difference between them.

That other definition may be formulated as follows: two s-machines S and G are equivalent if at any (identical) input to both machines, there is at least one state ξ_j of G for each state $\varkappa_i$ of S, and at least one state $\varkappa_i$ of S for each state ξ_j of G such that S and G, starting from $\varkappa_i$ and ξ_j, respectively, will generate identical outputs.

In this case, the equivalence between states depends, in general, on the input sequence. At some inputs some states of S may correspond to some states of G, but at other inputs the same states of S may correspond to different states of G (and conversely). The only requirement is that there be a unique relationship between states at any one input.

From the practical point of view, the disadvantage of this definition is that in order to find an initial state equivalent to a given one, one must know beforehand the corresponding input sequence. However, in most problems of practical importance the input sequence is not known in advance.

The definition of Section 9.4 imposes more stringent requirements: the equivalence between the states of S and G should not depend on any one input, but must hold for all allowable inputs. Thus if state $\varkappa_i$ of machine S corresponds to state ξ_j of an equivalent machine G, then S and G, starting from states $\varkappa_i$ and ξ_j, respectively, must generate identical outputs at all identical inputs (provided, of course, the inputs are allowed).*

Since the above definition of equivalence is less stringent than that of Section 9.4, it should yield minimal equivalent s-machines with fewer states than those possible in terms of the definition of Section 9.4. Let us illustrate this on an example.

Table 9.4

λ \ p	p_1	p_2
λ_1	λ_3	λ_3
λ_2	λ_2	λ_3
λ_3	λ_2	λ_1

Example. We are given an automaton A and a set L of allowed input sequences; the latter consists of all sequences which do not have two consecutive identical symbols. It is required to construct a minimal s-machine mapping (in terms of L) the automaton A specified by the basic Table 9.4.** The state diagram of A is shown in Fig. 9.23. In accordance with Section 9.7, it follows from Table 9.4, that when

*In some papers equivalence in the sense of Section 9.4 is referred to as *strong* equivalence, while that defined above is called *weak*.

**In Table 9.4, $\varkappa_i$ is already replaced by λ_i (see Section 9.7).

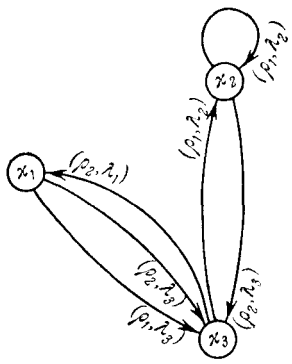


Fig. 9.23.

the definition of Section 9.4 is used, the minimal s-machine operating in the same way as A will have three states (since no two rows of Table 9.4 coincide).

Let us now use the other definition of equivalence and modify the state diagram of Fig. 9.23, replacing it with the diagram of an equivalent (in the sense of Section 9.4) machine, which is convenient for further minimization. The modification procedure is as follows: we replace each circle $\varkappa_i$ of Fig. 9.23 by r circles denoted by $\varkappa_i'$, $\varkappa_i''$, $\varkappa_i'''$, ... From each of the r new circles we draw the same paths, with the same labels, as those which originated in circle $\varkappa_i$. However, the paths terminating in circle $\varkappa_i$ will now be redirected to the new circles in the following manner: circle $\varkappa_i'$ will be the terminal of only those paths (previously leading to the circle $\varkappa_i'$) whose label contains ρ_1 as the first symbol; similarly circle $\varkappa_i''$ will be the terminal of paths whose label has ρ_2, and so on. We thus obtain the diagram of Fig. 9.24. Note that the loop at circle $\varkappa_2$ of Fig. 9.23, labeled (ρ_1, λ_2), is considered as both originating and terminating in that circle; in Fig. 9.24 it is replaced by paths originating in circles $\varkappa_2'$ and $\varkappa_2''$ and terminating only in circle $\varkappa_2''$. Circle $\varkappa_2'$ is seen to be associated with a loop path. Let us also mention that because only paths labeled λ_i lead to circle $\varkappa_i$ of Fig. 9.23 (this is because the s-machine operates as an automaton), the diagram of Fig. 9.24 has the corresponding property: a circle of group $\varkappa_i'$, $\varkappa_i''$, $\varkappa_i'''$, etc., can only be the terminal of paths labeled λ_i. In Fig. 9.24, these groups of circles are encircled with dotted lines.

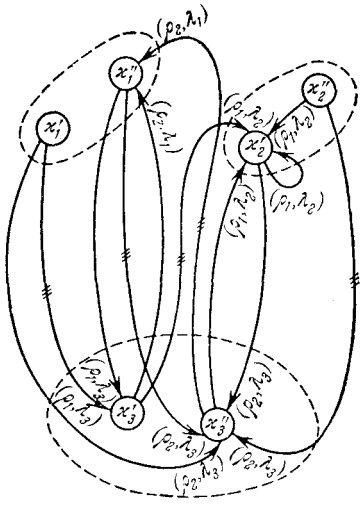


Fig. 9.24.

The modified state diagram of Fig. 9.24 is that of a machine N' which is equivalent (in the sense of Section 9.4) to A (in terms of the set E of all possible input sequences). To check whether the diagrams of Figs. 9.23 and 9.24 pertain to

equivalent machines, it is sufficient to prove that those states in each group of Fig. 9.24 which are encircled by a dotted line, are equivalent (that is, form a group of equivalent states). We replace each such group by single state and return to Fig. 9.23.

We now have machine N' which is equivalent (in the sense of Section 9.4) to A but has many more states (rn states). However, N' allows us an easy transition from *per se* constraints on the input sequences to Aufenkamp-type constraints.

Our inputs to N' shall be exclusively from L (since $L \subset E$), N' is also equivalent to A, in the sense of Section 9.4, in terms of L. Now consider some state of N', for instance, $\varkappa_1''$. We can reach $\varkappa_1''$ only via path (ρ_2, λ_1), that is, upon an input ρ_2; we can leave $\varkappa_1''$ only via path (ρ_1, λ_3), since two inputs ρ_2 cannot succeed each other (condition of problem, see p. 255). It seems, therefore, that we shall never be able to use path (ρ_2, λ_3), leaving $\varkappa_1''$, with the exception of the case in which the machine starts to work in state $\varkappa_1''$: in this case, an input ρ_2 shifts it to $\varkappa_3''$. But we can now use our new definition of equivalence to avoid this complication, for we can now substitute $\varkappa_2'$ [from which there is a path (ρ_2, λ_3) to $\varkappa_3''$] for the initial state $\varkappa_1''$, which solves our problem. Therefore, we can delete path (ρ_2, λ_3) from $\varkappa_1''$ to $\varkappa_3''$.

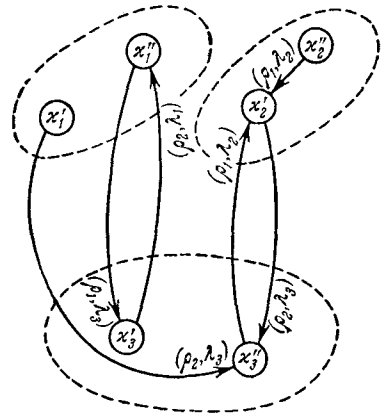


Fig. 9.25.

Following this line of reasoning, we can delete from the state diagram all the paths marked (≡). The general "algorithm for deleting path" is as follows: the path labeled ρ_s originating at $\varkappa_i^s$ (where s is the number of primes) should be deleted. The diagram of Fig. 9.24, minus the deleted paths, is shown in Fig. 9.25, and represents machine N''. That machine operates exactly as N' (and also as A), provided all the inputs belong to set L. But something has happened in this transformation, because now N'' is equivalent, in terms of L, to N' (and consequently, also to A) in a new sense: correspondence between states of N'' and N' now depends on the input sequence. Also, the diagram of N'' shows that a given circle (state) is no longer the origin of all paths that started in circles of machine A; that is, we have transformed the machine from one subject to constraints *per se* to one with Aufenkamp-type constraints.

We shall now minimize N'' by means of the algorithm of Section 9.8.

The interconnection matrix C'' of N'' is

$$C''=\begin{array}{c} \\ x_1' \\ x_1'' \\ x_2' \\ x_2'' \\ x_3' \\ x_3'' \end{array}\begin{array}{c} \begin{array}{cccccc} x_1' & x_1'' & x_2' & x_2'' & x_3' & x_3'' \end{array} \\ \left[\begin{array}{cccccc} 0 & 0 & 0 & 0 & 0 & (\rho_2, \lambda_3) \\ 0 & 0 & 0 & 0 & (\rho_1, \lambda_3) & 0 \\ 0 & 0 & 0 & 0 & 0 & (\rho_2, \lambda_3) \\ 0 & 0 & (\rho_1, \lambda_2) & 0 & 0 & 0 \\ 0 & (\rho_2, \lambda_1) & 0 & 0 & 0 & 0 \\ 0 & 0 & (\rho_1, \lambda_2) & 0 & 0 & 0 \end{array}\right] \end{array}.$$

In general, C'' has a property which follows from the above-mentioned properties of the state diagram: the column with the heading x_i^s (where s is the number of primes) can contain only pairs (ρ_s, λ_i).

Our algorithm minimizes the number of states of the machine by symmetrical decomposition of the starting matrix into generalized 1-matrices. In the case of C'', we need only to draw horizontals to obtain generalized 1-matrices: verticals cannot "spoil" the grouping. These horizontals may be drawn in many ways. One way is to draw them between rows 2 and 3, and 4 and 5. This partitions the states of N'' into groups $\{x_1', x_1''\}$, $\{x_2', x_2''\}$ and $\{x_3', x_3''\}$. If we then replace each group with a single state, we get again the starting automaton A of Fig. 9.23, which had three states. However, a better grouping is obtained by drawing a horizontal between rows 3 and 4 of C''. This divides all the states into groups $\{x_1', x_1'', x_2'\}$ and $\{x_2'', x_3', x_3''\}$. (There is no way of obtaining fewer than two groups because rows 1 and 4 can never be part of one generalized 1-matrix.) Now, we draw a symmetrical vertical line between columns 3 and 4, combine the elements of each generalized 1-matrix, and get the interconnection matrix C''' of a minimal s-machine N'''

$$C'''=\begin{array}{c} \\ x_1 \\ x_2 \end{array}\begin{array}{c} \begin{array}{cc} x_1 & x_2 \end{array} \\ \left[\begin{array}{cc} 0 & (\rho_1, \lambda_3)\vee(\rho_2, \lambda_3) \\ (\rho_1, \lambda_2)\vee(\rho_2, \lambda_1) & 0 \end{array}\right] \end{array},$$

whose state diagram is shown in Fig. 9.26. Machine N''' is a pseudo-mapping of N''. But here the sets of inputs allowable in all the states of N'' coincide with each other and with L. Therefore, N'' also maps N'' in terms of set L.

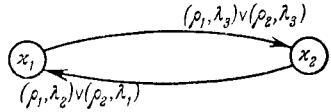


Fig. 9.26.

Note that C'' contains two pairs of identical rows: 1 and 3, and 4 and 6. Row 1 corresponds to state x_1' of Fig. 9.25, and the row 4 to state x_2''. Figure 9.25 shows that these states can only act as initial ones, since there are no paths to them. Therefore, we can further simplify this state diagram by removing these states: thus,

$\varkappa_2'$ can act for $\varkappa_1'$ as an initial state [an identical path (ρ_2, λ_3) leads from $\varkappa_2'$ to $\varkappa_3''$]. Similarly, $\varkappa_2''$ can be replaced by $\varkappa_3''$.

Making similar preliminary simplifications wherever possible, we shall arrive at a diagram with fewer states, that is, at an interconnection matrix of a lower order. In our example, we could have started the algorithmic minimization with a matrix simpler than C''—one with no rows or columns composed exclusively of zeros:

$$C_*'' = \begin{array}{c} \\ \varkappa_1'' \\ \varkappa_2' \\ \varkappa_3' \\ \varkappa_3'' \end{array} \begin{array}{c} \begin{array}{cccc} \varkappa_1'' & \varkappa_2' & \varkappa_3' & \varkappa_3'' \end{array} \\ \begin{bmatrix} 0 & 0 & (\rho_1, \lambda_3) & 0 \\ 0 & 0 & 0 & (\rho_2, \lambda_3) \\ (\rho_2, \lambda_1) & 0 & 0 & 0 \\ 0 & (\rho_1, \lambda_2) & 0 & 0 \end{bmatrix} \end{array}.$$

Then, symmetric decomposition of C_*'' gives the same result as that obtained with C'' as the starting matrix.

Compare now machine N''' (Fig. 9.26) with the initial automaton A (Fig. 9.23): we see that indeed it is the input sequence which governs the equivalence of states of the two machines. Thus, assume the machine of Fig. 9.23 starts up in state $\varkappa_2$, and the input is $\rho_1\rho_2\rho_1\rho_2\rho_1\rho_2\ldots$. Then, to obtain the same output with the machine of Fig. 9.26, the latter must be started from state $\varkappa_2$; if, however, the machine of Fig. 9.23 starts from this state $\varkappa_2$ with an input $\rho_2\rho_1\rho_2\rho_1\rho_2\rho_1\ldots$, then, to obtain the same output from the machine of Fig. 9.26, the latter must be started from state $\varkappa_1$.