Minimizing the Number of States in Incompletely Specified Sequential Switching Functions*

M. C. PAULL† AND S. H. UNGER†

Summary—**Given a sequential switching function in the form of a flow table in which some of the entries are unspecified, the problem of reducing the number of rows in that flow table is extremely complex, and cannot, in general, be solved by any simple extension of the methods used for completely specified functions. An analysis of the problem is presented, and a partially enumerative solution is evolved. A rough indication of the efficiency of the given procedures may be obtained from the fact that these techniques have been successfully applied to approximately two dozen tables ranging up to about 15 rows. No solution required more than two hours.**

I. Introduction

Terminal characteristics of sequential switching circuits (or machines) are conveniently specified in the form of flow tables[1–4] such as Table I. The rows represent *internal states*, and the columns represent *inputs*. For each *total state* (specified by both the internal state and the input), the table indicates the next internal state, and an output e_k. Thus, for example, if the system starts in state 3 and if the input sequence I_2, I_3, I_3, I_1, I_1, I_2 is applied, then the output sequences e_3, e_2, e_2, e_2, e_1, e_3 will result, and the final internal state will be 5. (This particular table represents a pulsed signal system.)

Previous work in this area has been applicable to machines that are synchronous or asynchronous, with pulsed signals or level type signals. We have found no reason to believe that the results presented here are any less general.

A flow table P will be said to *cover* a flow table Q (written as $P \geq Q$) if, for every state q of Q, there is a corresponding state p of P, such that for every possible sequence of inputs, the resulting output sequence from P when it starts in p will be identical to the sequence obtained from Q when it starts in q.[1–5] Given a fully specified flow table Q, an important problem in sequential circuit synthesis is to find the minimum-row member of the set of tables that cover Q. This minimization problem has been completely solved by Huffman, Moore, and Mealy.[1–3] Another paper on this subject has been written by Aufenkamp and Hohn.[6]

TABLE I

	I_1	I_2	I_3
1	1, e_1	5, e_3	1, e_4
2	3, e_2	4, e_3	2, e_2
3	1, e_1	2, e_3	3, e_1
4	5, e_1	4, e_3	3, e_1
5	2, e_1	5, e_1	2, e_1

The class of flow tables considered may be broadened to include unspecified or "don't care" entries. That is, for certain total states, the next-state entries or the outputs (or both) may be left unspecified. A detailed discussion of how such entries should be interpreted will be presented in Section II. The definition of covering can be generalized to include incompletely specified functions, and the minimization problem can then be reformulated in terms of the broader definition. This will be done in Section III.

Until recently, it was widely believed that the minimization of incompletely specified tables could always be accomplished with modified versions of the above mentioned methods.[7] However, Ginsburg[8–10] has recently discovered that there are examples of such flow tables that cannot be minimized through the use of the classical procedures. He has made a preliminary study of the situation and described some minimization procedures.

In this report, we shall present what we feel is a more thorough analysis of the problem as well as some minimization methods superior to those given by Ginsburg, although still not fully satisfactory in that some enumeration is required.

Formal proofs of theorems will be relegated to an Appendix in order to avoid interrupting the general development. A number of illustrative examples are given in Section V.

II. Incompletely Specified Functions

There are several ways of interpreting unspecified entries in flow tables. According to one interpretation, each blank entry may be filled arbitrarily, but a definite

* Manuscript received by the PGEC, April 14, 1959; revised manuscript received, July 1, 1959.

† Bell Telephone Labs., Inc., Whippany, N. J.

[1] D. A. Huffman, "The synthesis of sequential switching circuits," *J. Franklin Inst.*, vol. 257, pp. 161–190, 275–303; March/April, 1954.

[2] G. H. Mealy, "A method for synthesizing sequential circuits," *Bell Sys. Tech. J.*, vol. 34, pp. 1045–1079; September, 1955.

[3] E. F. Moore, "Gedanken experiments on sequential machines," in "Automata Studies," Princeton University Press, Princeton, N. J., pp. 129–153; 1956.

[4] S. H. Caldwell, "Switching Circuits and Logical Design," John Wiley and Sons, Inc., New York, N. Y.; pp. 453–659; 1958.

[5] This is the same as the equivalence definition used by Huffman, Moore, and Mealy for fully specified flow tables.

[6] D. D. Aufenkamp and F. E. Hohn, "Analysis of sequential machines," IRE Trans. on Electronic Computers, vol. EC-6, pp. 276–285; December, 1957.

[7] D. D. Aufenkamp, "Analysis of sequential machines, II," IRE Trans. on Electronic Computers, vol. EC-7, pp. 299–306; December, 1958.

[8] S. Ginsburg, "A Synthesis Technique for Minimal State Sequential Machines," Natl. Cash Register Co.; July, 1958. (Unpublished.)

[9] S. Ginsburg, "On the reduction of superfluous states in a sequential machine," *J. Assoc. Comp. Mach.*, vol. 6, pp. 259–282; April, 1959.

[10] S. Ginsburg, "A technique for the reduction of a given machine to a minimal-state machine," this issue, p. 346.

TABLE II

	I_1	I_2
1	1, -	2, e_1
2	3, e_1	1, e_1
3	2, e_2	1, e_1

(a)

	I_1	I_2
1	1, e_1	2, e_1
2	3, e_1	1, e_1
3	2, e_2	1, e_1

(b)

	I_1	I_2
1	1, e_2	2, e_1
2	3, e_1	1, e_1
3	2, e_2	1, e_1

(c)

	I_1	I_2
A	B, e_1	A, e_1
B	A, e_2	A, e_1

(d)

entry must be made. In other words, a function M, such as the one described by Table II(a), is covered by the functions described by both Table II(b) and Table II(c).

Another approach is to specify that the output will simply be ignored whenever the system enters a total state where the output is unspecified. The distinction between the two cases is that in the latter instance, it is not required that the *same* output always occur for the unspecified condition. That is, a function will be said to cover that of Table II(a) if at all times its behavior corresponds to *either* Table II(b) *or* Table II(c), but not necessarily always to the same one.

The difference between these two interpretations is quite significant when an attempt is made to minimize a table such as Table II(a). Applying classical reduction procedures, we see that neither Table II(b) nor Table II(c) can be reduced. Hence, if the unspecified entry is interpreted according to our first proposal, then Table II(a) cannot be reduced. However, under the second interpretation, Table II(d) covers Table II(a) and thus constitutes a reduced version of it. Row A corresponds to a combination of states 1 and 2 of the original table, and row B corresponds to a combination of states 1 and 3. This example constitutes a case where the classical procedures will not yield the best solution due to the implied interpretation of the unspecified entries.

Actually, the second interpretation that allowed us to reduce Table II(a) is the one that conforms to most situations in which such entries occur (for example, where certain input sequences are disallowed). Such a statement is not susceptible to proof, so it will simply be arbitrarily stated here that we shall use the second meaning. The aptness of this choice must be decided for each situation.

Although the illustrations in the preceding paragraph concerned output entries, the same arguments apply to unspecified next-state entries such as in Table III(a).

Using the idea that the desired machine need not behave as though some constant next-state entry were in place of the dash, Table III(a) can be reduced to Table III(b). Rows 1 and 2 are mapped into A, rows 1 and 3 are mapped into B, and row 4 is mapped into C. If it is required that the dash be replaced by some fixed-row number, then no reduction will be possible.

In the case of unspecified next-state entries, an equivalent way of stating our interpretation is to assume that no further inputs will occur after the machine enters the unspecified state.

A special comment is called for by the work of Aufenkamp.[7] He apparently accepts the same interpretation of incompletely specified functions that we are using, although he does not consider total states in which *only* the outputs are unspecified or in which *only* the next-states are unspecified.[11] It is therefore impossible to apply the procedures he describes to either of our two preceding examples. (Indeed, it does not seem possible even to *state* the second example in terms of Aufenkamp's connection matrices.)

TABLE III

	I_1	I_2
1	–, e_1	1, e_1
2	1, e_1	3, e_1
3	4, e_1	2, e_1
4	4, e_2	2, e_1

(a)

	I_1	I_2
A	A, e_1	B, e_1
B	C, e_1	A, e_1
C	C, e_2	A, e_1

(b)

TABLE IV

	I_1	I_2	I_3
1	–	1, 0	2, 0
2	1, 0	3, 0	2, 0
3	3, 1	2, 0	1, 0

(a)

	I_1	I_2	I_3	
A	A, 0	B, 0	A, 0	(12)
B	B, 1	A, 0	A, 0	(13)

(b)

We therefore present Flow Table IV(a), which is of the type which Aufenkamp claims to be able to reduce to its "simplest compatible form." Using the procedures to be described in the following sections, it is easy to reduce the 3-row Table IV(a) to the 2-row Table IV(b). As we understand Aufenkamp's procedures, they would fail to achieve any reduction, since he seeks only *disjoint* partitions of the states, whereas the given table can be reduced only by using the *overlapping* partition [(12), (13)]. This point will be amplified later.

Subsequent sections will concern the methods used in minimizing tables such as those discussed in the preceding examples.

III. General Nature of Minimization Process

An over-all picture of the minimization process will be presented in this section. Some definitions and a theorem are necessary prerequisites.

Definition 1: If a sequence of inputs is applied to flow table P (this is a less cumbersome, if less precise,

[11] *Ibid.;* Ginsburg *does* consider the case where only the next-state is unspecified, but, in effect, he does not distinguish between the case where both entries are blank and the case where only the output is unspecified.

reference to an input to *a machine described by* flow table P) when it is initially in state r, then this sequence will be said to be *applicable* to r if the state of the flow table is specified after each input, except possibly the last. That is, no unspecified next-state entries are encountered when an applicable input sequence is applied, except possibly at the final step.

Definition 2: A state p of a flow table P will be said to *cover* a state q of a flow table Q ($p \geq q$) if, when any input sequence applicable to q is applied to both P and Q when they are initially in states p and q respectively, the outputs obtained from P are the same as the outputs obtained from Q, wherever the latter are specified.

Definition 3: A flow table P *covers* a flow table Q ($P \geq Q$) if, for every state q in Q, there is a state p in P such that $p \geq q$.

Suppose that all the states of flow table Q can be grouped into sets $C_1, C_2, \cdots, C_m$ (C-sets), not necessarily disjoint, such that there is another flow table P with states $p_1, p_2, \cdots, p_m$, where p_1 covers all members of C_1, p_2 covers all members of C_2, and in general p_i covers all members of C_i ($i=1, 2, \cdots, m$). It follows from definition 3 that $P \geq Q$, and if m is less than the number of states of Q, then P can be considered as a *reduced version* of Q. Our goal is to find a *minimum-row version* of the given table: a reduced version with the smallest number of rows.

Definition 4: Let $N(s, I_k)$ be the next-state entry (if specified) for state s and input I_k.

Definition 5: Let $Z(s, I_k)$ be the output (if specified) for state s and input I_k.

Definition 6: A set of states P is *implied* by a set of states R, if, for some input I_k, P is the set of all $N(r, I_k)$'s taken over all pairs (r, I_k), where r belongs to R and $N(r, I_k)$ is specified.

Definition 7: A grouping of all the rows of a flow table into C-sets (not necessarily disjoint) will be said to be *closed* if: 1) every set implied by any C_i is included in one of the C-sets; and 2) $Z(p, I_k) = Z(t, I_k)$ (if both are defined) where p and t belong to C_i, for every I_k and for every i.

Theorem 1: Every minimum-row version of a flow table corresponds to a closed grouping of C-sets of that table, in that each row of the reduced table covers all members of one C-set; and every closed collection of m C-sets corresponds to an m-row flow table that covers the given flow table, provided that every state of the given table is represented in at least one of the C-sets.

The following method always will be shown to yield a minimum-row flow table that covers a given n-row flow table Q.

Process A:

1) Group the states of Q into a number (less than n) of sets C_i which are not necessarily disjoint and whose union is the set of all states of Q.
2) Determine whether this grouping can correspond to a machine P which covers Q. (In other words, test the grouping for closure.)
3) Repeat this process for all possible groupings and select the grouping (or one of the groupings) with the smallest number of sets that satisfies step 2).
4) Form the reduced table (which is a minimum-row table according to Theorem 1), by using a single state in place of each set of states of the grouping found in step 3) as follows:
   a) $Z(p_i, I_k)$ is left unspecified if the outputs for all members of C_i are unspecified in column I_k of table Q. Any specified outputs in I_k for members of C_i will be the same (definition 7) and, in such cases, this is the value to use for $Z(p_i, I_k)$.
   b) $N(p_i, I_k)$ is left unspecified if next-state entries in column I_k are unspecified for all members of C_i. Otherwise (according to definiton 7), there will be at least one C-set such that all specified I_k next-state entries for C_i are included in that set. In this case, choose one such set, say C_j, and let $N(p_i, I_k) = p_j$.

It is important to realize that a state of Q may appear in more than one of these sets. This is the key point that was missed in applying the classical reduction process to incompletely specified machines. For example, the states of the machine described by Table III(a) are grouped as (12), (13), (4) to obtain Table III(b). The process of forming the C_i sets is not unique since, in general, many machines can be found to cover a given machine.

Process A is complete, and will always yield the desired result. However, except in the most trivial cases, an enormous amount of enumeration would be required in testing all groupings that satisfy step 1). Most of this enumeration can be eliminated, and the basis for a simplified approach will be discussed in the next section.

IV. Compatibles—Components of Groupings

Each grouping of states referred to in the previous section consists of a collection of sets of states. In this section, we shall describe procedures for determining which sets of states should be considered in such groupings.

Definition 8: A *compatible* is a set of states of a flow table that constitutes one member of some closed grouping. Every state of a flow table is, by itself, a compatible.

Definition 9: Two states are *incompatible* if they do not constitute a compatible.

One reduction procedure for flow table M may now be written as follows.

Process B:

1) If there is a compatible consisting of all states in M, then obviously it is a closed collection and thus corresponds to a reduced flow table.
2) If no one compatible corresponds to a reduced table, then test for closure (definition 7) each collection of compatibles whose union includes all

states of M, starting with 2-member groupings, then testing the 3-member groupings, etc. If no collection of $n-1$ or fewer compatibles is satisfactory, then M cannot be reduced.

3) If a satisfactory set of compatibles is found, then construct the reduced machine using step 4 of process A.

The next problem is to find the compatibles. One method is based on Theorem 2.

Theorem 2: A set of states is a compatible if, and only if, every pair of states in that set is a compatible.

Thus, by considering only pairs of states, all compatibles can be found. For example, if a, b, c and d are states, then abc is a compatible if ab, ac and bc are compatibles. If they are, then to see whether d can be added to this compatible it is necessary to determine whether ad, bd, and cd are also compatibles.

The problem now is to determine which pairs of rows of a given flow table are compatibles.

Definition 10: The *chain* generated by a set of states B is the collection of *all* sets produced by the following recursive process:

1) B is a member of the chain.
2) If X is a member of the chain, then all sets implied (definition 6) by X are added to the chain.

As an example of how a chain is formed, consider Table V. Fig. 1 illustrates the formation of the chain

TABLE V

	I_1	I_2	I_3
1	1, 0	3, 1	4, 1
2	–, 0	5, 1	6, 1
3	–	1, 0	5, 1
4	2, 1	3, 0	4, 0
5	6, 1	–	4, 0
6	3, 1	6, –	1, 0

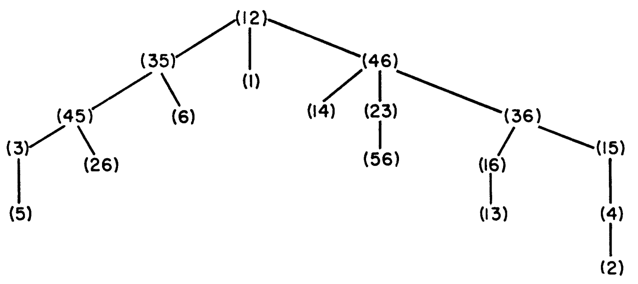

Fig. 1—A chain.

generated by (12). For example, in column I_2, the next-state entries for rows 1 and 2 are 3 and 5, respectively. Hence, the line from (12) to (35) in the figure. Since, in column I_3, the next-state entries for rows 3 and 5 are 5 and 4, respectively, (45) is added to the chain. The process is continued until no new members can be found. The compatibles can now be found with the aid of Theorem 3.

Theorem 3: A set B is a compatible if, and only if, there is no set X in the chain generated by B, such that, for some k, $Z(x_1, I_k)$ and $Z(x_2, I_k)$ are defined and unequal, where x_1 and x_2 belong to X.

When using this theorem to test a given set for compatibility, it is advisable to check the outputs of each set for agreement as the chain is being constructed, since a contradiction early in the process may make it unnecessary to complete the chain. For instance, in the example of Fig. 1, it would have been possible to terminate the test after (35) was shown to belong to the chain because the outputs of rows 3 and 5 are contradictory for input I_3.

Since tests for pairwise compatibility are basic to all of our procedures, it will be useful to have a systematic form for carrying out such tests and recording the results. The *implication table* illustrated in Table VI(b), leads to a particularly convenient and orderly procedure for deriving compatibles. We shall demonstrate the method by means of an example based on the flow Table VI(a).

TABLE VI

	I_1	I_2	I_3	I_4
1	2, –	3, 0	–	4, –
2	3, 0	5, 0	–	–
3	4, –	6, –	3, –	–
4	5, 1	3, 0	–	1, –
5	–	6, 0	–	–
6	–	–, 1	4, –	2, –

(a)

2	23 35				
3	24 36	34 56			
4	25	X	45 36		
5	36	56	$\surd$	36	
6	X	X	34	X	X
	1	2	3	4	5

(b)

2	X				
3	X	X			
4	X	X	45 36		
5	36	X	$\surd$	36	
6	X	X	34	X	X
	1	2	3	4	5

(c)

There is one cell in Table VI(b) for each pair of rows of the given flow table. The first step in our process is to fill these cells column by column, starting at the top of each column in the following manner.

Process C:

1) If there is a contradictory pair of outputs for a given row-pair in any input column, then an X is placed in the corresponding cell of the implication table and no further attention need be given to that row-pair. This condition occurs for (16), (26), (46), (56), and (24) in our example.
2) If, for row-pair ab, the outputs are not contradictory, then enter all row-pairs implied by ab in the ab cell. (It is never necessary to enter ab in the ab cell.) Thus, in cell (12) we enter (23) and (35), in cell (13) we enter (24) and (36), etc. In cell (35) there are no pairs to be entered, so a check mark is inserted. Note that in cell (14) we enter only (25), omitting (14). At this point, the table is in the form of Table VI(b).

After the first "pass" through the table has been made, a second inspection is made of all cells corresponding to row-pairs that are entries in the table. In our example we inspect, in sequence, the cells corresponding to (23), (35), (24), (36), (25), (36), etc. If the entry in position ab is an X, then an X is inserted in all cells that contain ab, and any other entries in such cells may be ignored in all further steps. Thus, in this step we insert an X in position (13) since this cell contains a (24) entry. X's are also inserted in the cells corresponding to (23) and (25) because these cells contain (56) as an entry, and cell (56) contains an X. This process is then repeated. The next time, X's are inserted in position (12), which contains a (23) entry, and (14), which contains a (25) entry. On the next inspection, no new X's are added, and the process is thus terminated. For the sake of clarity, the final implication table is shown as a separate table, namely Table VI(c). (In practice, there is no need to redraw the original table.)

All row-pairs whose entries are not X's are compatible, and those with X's are incompatible. The process described here is fully equivalent to the method of Theorem 3 (applied to 2-member sets) and is sufficiently well defined to be programmed on a digital computer.

Definition 11: A *maximal compatible* (MC) is a set of rows which form a compatible and which is not included in any larger compatible.

If we can find all of the MC's, then we have in effect found all compatibles, since an obvious corollary of Theorem 2 is that every subset of a compatible is also a compatible.

We shall use Table VII to illustrate the method. (The non-X cells in this implication table have not been filled in since their contents are irrelevant for the purpose of finding the MC's.)

TABLE VII

2	X				
3					
4	X	X	X		
5		X		X	
6					X
	1	2	3	4	5

Process D: Start from the rightmost column of the implication table and work leftward constructing a list of compatibles as follows.

1) Write down the pairwise compatibles in the first column that has any. In our example, this is (46).
2) Examine the next column to the left (say column k). If k is pairwise compatible with both members of any previously listed 2-member compatibles, then add k to these compatibles, replacing the pairs with 3-member compatibles. Add to the list any other pairwise compatibles in the k column. In our example, for $k=3$ we add (35) and (36) to the list. For $k=2$ we replace (36) by (236).
3) Proceed to the next column on the left which has one or more non-X entries (say f). If f is compatible with all members of some previously listed compatibles, then add f to those compatibles. If f is compatible with all the members of any subsets of previously listed compatibles, then add to the list compatibles consisting of f and these subsets. Also add any other pairwise compatibles in the f column. At each step, eliminate compatibles included in other members of the list. Repeat this process until all columns have been examined. The final list (augmented by any single states not in any pairwise compatibles) will contain all of the MC's. In our example, for $f=1$, we replace (35) with (135) and add (136) to the list. The resulting MC's are therefore (46), (236), (135) and (136).

This is a straightforward process which is not too laborious, and which can be programmed.

There is a dual procedure for finding the MC's which is based on the use of incompatible pairs to break down large sets of rows into smaller and smaller sets until only maximum compatibles remain. The basic operation performed in this process is as follows.

If $abcdef$ is being considered as a possible MC, and if b and e are incompatible, then the candidate set is split into two parts, one with b omitted, and the other with e omitted, namely $acdef$ and $abcdf$. The latter two sets are now candidates, and the next incompatible pair is applied to each in a similar manner. At each stage, candidate sets are omitted if they are contained in other candidate sets.

An efficient way of carrying out these steps is described below, again using Table VII as an example.

Process E: Start with column 1 and work to the right.

1) Write down the set of all states except for 1. Then write down a set of states consisting of 1 followed by all states corresponding to non-X rows in column 1. In our example, we would have at this point (23456) and (1356).
2) Move to the next column containing X's (say column j). Examine those sets on the list that contain j. Replace each such set which also includes at least one state corresponding to rows that are X's in column j by two sets—the original set without j, and the original set without any of the states that are incompatible with j, as indicated by the X's in the j column. Repeat this process for each column, and at each stage eliminate sets included in other sets on the list.

In our example, the list of sets progresses as below after the indicated column is inspected.

1) (23456) (1356)
2) (3456) (236) (1356)
3) (456) (236) (1356)
4) (46) (236) (1356)
5) (46) (236) (135) (136).

The final result, as shown in step 5), is the same as that obtained with Process D. A method for finding the MC's without using 2-member compatibles is given in the Appendix.

It is possible to devise variations combining elements of both of the methods described above. For example, in Table VI(c) we might have observed at once that 1 is compatible only with 5 and that 2 is not compatible with any other row. Then (15) and (2) could have been listed at once as MC's, and the incompatibles could have been used to reduce (3456) to (345) and (36).

V. Using the Compatibles

The minimization process can now be restated:

Process F:

1) Find the maximal compatibles, using the procedures of Section IV.
2) Find the smallest closed collection of compatibles.
3) Merge each compatible of the chosen collection into a single row of a new reduced table as described in step 4) of Process A.

It would be most satisfying to be able to continue the development by presenting a systematic procedure for selecting a minimal closed set of compatibles. Unfortunately, we have been unable to find any such general method (other than enumeration). The remainder of the paper will be concerned with various isolated tools, miscellaneous results (mainly of a negative nature), and a series of nontrivial examples to illustrate some of the ideas and to show that the situation is really not as bad as the preceding portion of this paragraph would seem to indicate. We have applied the principles evolved here to about two dozen flow tables containing as many as fifteen rows, and have always succeeded in finding a minimum solution within about two hours. The use of a digital computer would no doubt extend the size of the tables that could be reduced.

An upper bound on the number of states in any minimized table is the number of MC's, since the set of all MC's is always closed.

A lower bound can be found in terms of *incompatibles*, which are sets of states that are pairwise incompatible. *Maximal incompatibles* are those incompatibles not included in any other incompatibles those which can be found by interchanging X's and non-X's in the final implication table and using either of the procedures described for finding MC's. The number of elements in the largest maximal incompatible is a lower bound on the number of rows in the reduced table. This was pointed out by Ginsburg.[9]

The following result, which is almost self-evident, is sometimes useful and leads to a simple solution in the case of completely specified flow tables.

Theorem 4: If each MC of Q contains a state not contained in any other MC, then the minimum-state flow table P such that $P \geq Q$ can be synthesized from the closed set of all MC's. The number of states of P is equal to the number of MC's.

Since for a completely specified flow table, the premise of the above result is met "supremely," in that the MC's for a completely specified machine are disjoint, one needs only to find the MC's to find a minimum-state covering flow table for this case.

The following is a variation of reduction Process F:

Process G:

1) Find the MC's (and thereby all of the compatibles).
2) Find the chain generated by each compatible (definition 10). Some compatibles generate identical chains, so that the number of chains will often be less than the number of compatibles.
3) Try all combinations of chains. Within each combination, eliminate compatibles that are included in other compatibles. (If a compatible p appears in a minimum closed collection, then q cannot be a member of that collection if q is a subset of p.) The minimum closed collection of compatibles will consist of one of these reduced collections.
4) Merge each of the selected compatibles into one row of a reduced table.

(Note that in the following examples some of the flow tables are not strongly connected.[3] This fact is of no significance, since each of these tables can be converted to strongly connected tables by adding supplementary input columns. This can always be done without causing any other changes in the problems.)

Example 1: Looking at Table VIII(a), it seems obvious that row 2 can be eliminated immediately by merging it with row 1. The resulting table (with the merged pair labelled 2 is shown as Table VIII(b). Table IX(a) is the implication table for Table VIII(b) and the MC's are simply (2), (4) and (35). All are required to form the necessary closed set, and so the reduced table having rows A, B and C corresponding to (2), (4) and (35), respectively, is shown as Table IX(b). This table meets both the upper and lower bounds for Table VIII(b).

TABLE VIII

	I_1	I_2	I_3
1	3, 0	5, 1	–
2	3, 0	5, –	–
3	2, –	3, 0	1, –
4	2, 0	3, –	5, –
5	–	5, 0	1, –

(a)

	I_1	I_2	I_3
2	3, 0	5, 1	–
3	2, –	3, 0	2, –
4	2, 0	3, –	5, –
5	–	5, 0	2, –

(b)

TABLE IX

3	X		
4	23X	25X	
5	X	√	25X 35
	2	3	4

(a)

	I_1	I_2	I_3
A	C, 0	C, 1	–
B	A, 0	C, –	C, –
C	A, –	C, 0	A, –

(b)

Now let us attempt to reduce the original Table [VIII(a)] directly. The implication table is Table X(a). Using Process E for finding the MC's, the following steps result.

1) (2345) (124)
2) no change
3) (245) (235) (124)
4) (24) (25) (235) (124) so the MC's are (124) and (235).

Thus, using both MC's, we obtain the *two*-row Table X(b), where A and B correspond to (124) and (235), respectively. This table meets both the lower and upper bounds for Table VIII(a).

TABLE X

2	√			
3	X	35		
4	23 35	23 35	15X	
5	X	√	√	15X 35
	1	2	3	4

(a)

	I_1	I_2	I_3
A	B, 0	B, 1	B, –
B	B, 0	B, 0	A, –

(b)

By making what appeared to be an obvious merger in the original table, we lost an opportunity to reduce the table to two rows instead of three. The reason, in this case, is that Table VIII(b) is more completely specified than is Table VIII(a). Since any change which *relaxes* the specifications is clearly not allowed, we see that the moral of this example is that it is unwise to make preliminary mergers in a given flow table. Such changes are always safe only in the trivial cases where two rows are identical, or where there is a row with *all* outputs and next-states left unspecified.

Example 2:

TABLE XI

	I_1	I_2	I_3	I_4	I_5	I_6	I_7
1	6, 0	1, –	4, –	3, –	–	–	–
2	–, 1	–	–	–	3, –	4, –	5, –
3	3, –	5, –	–	–	6, 0	2, –	–
4	–	–	6, –	5, –	–, 1	–	1, –
5	1, –	–	1, 1	–	2, –	–	3, –
6	–	4, –	–, 0	2, –	–	5, –	–

(a)

2	X				
3	15 36	24 36			
4	35 46	15	X		
5	16 14	23 35	13 26	13 16	
6	14 23	45	25 45	25	X
	1	2	3	4	5

(b)

Table XI(b) is the implication table for Table XI(a). Using Process D for finding the MC's, the steps are:

1) (45) (46)
2) (45) (46) (35) (36)
3) (245) (246) (235) (236)
4) (245) (246) (235) (236) (145) (146) (135) (136).

Step 4) gives the MC's. The upper bound on the number of necessary rows is thus useless since it exceeds 6. The lower bound is 2.

Our first thought now might be to choose (135) and (246) as the numbers of the closed set, since they include all six states. However, a check of the flow table shows that (135) implies (136), which is not included in either (135) or (246). Thus (135) and (246) do not constitute a closed set. Further efforts at choosing a closed set from among the MC's are balked by similar difficulties as a result of the wealth of implications associated with each compatible.

If we construct a chain of implications starting with any of the pairwise compatibles, we find that *all* the pairwise compatibles will be included. In other words, no collection of compatibles is closed unless every pairwise compatible is included in at least one member of the collection.

Furthermore, if we examine the original flow table, we find that (145) implies (146), which implies (235), and so forth, so that all of the MC's except (246) form a circular chain of implications. Thus, if any of the MC's other than (246) appear in a collection, then that collection will not be closed unless it contains the seven MC's in the chain. But this collection would have more members than the number of rows in Table XI(a) and hence would not correspond to a reduction.

If (246) appeared, then the pairs (25), (23), (14), (15), (16), (45), (13), (35), and (36) would also have to appear. Again, this would not constitute a reduction. Finally, if only pairs appeared, all twelve would be necessary.

Hence we conclude that this flow table cannot be reduced, even though the MC's have three members each. Neither bound is met.

Example 3:

TABLE XII

	I_1	I_2
1	–, 0	–
2	–, 0	1, –
3	1, –	–
4	–	–, 0
5	6, 1	4, –
6	–	5, –

(a)

2	√				
3	√	√			
4	√	√	√		
5	X	X	16	√	
6	√	15X	√	√	45
	1	2	3	4	5

(b)

The maximal compatibles are found using Process E as follows:

1) (12346) (23456)
2) (1346) (1234) (3456)

The upper bound is 3 and the lower bound is 2. Note that only (1234) contains a 2 and that this is compatible implies no others. Hence there must be a minimum solution containing (1234). Compatibles with 5 and 6 are now required. If we try (56), then it will be necessary to include another compatible with (45), since (56) implies (45). Suppose we use (456) instead of (56). Then the compatibles (1234) (456) are closed and therefore constitute a solution. Note that the 4 in (1234) serves no purpose and so (123) (456) is also a solution. Let A and B replace (123) and (456), respectively. Then a reduced version of Table XII(a) is Table XIII, which meets the lower bound.

TABLE XIII

	I_1	I_2
A	A, 0	A, –
B	B, 1	B, 0

If we had used (1234), then the unspecified output in the I_2 column of Table XIII would have been fixed at a zero. Thus, when there is an opportunity to replace a compatible by a subset of itself it is desirable to do so, since the reduced table will, in general, then have more unspecified entries.

Example 4:

TABLE XIV

	I_1	I_2	I_3	I_4
1	2, 0	1, 0	3, 0	–
2	2, 0	1, 0	7, 0	–
3	5, 0	1, 0	3, 0	–
4	5, 0	4, 0	3, 0	8, 0
5	5, 0	6, 0	–	–
6	2, 0	6, 0	7, 0	9, 1
7	–	4, 0	7, 0	–
8	5, 0	1, 0	7, 0	8, 0
9	5, 0	1, 0	7, 0	9, 1

(a)

2	37							
3	25	25 37						
4	25	25 14 37	14					
5	25 16	16	16	46 X				
6	37	16	25 16 37	X	25			
7	14 37	14	14	37	46 X	46 X		
8	25 37	25	37	14 37	16	X	14	
9	25 37	25	37	X	16	25 16	14	X
	1	2	3	4	5	6	7	8

(b)

Using Process D:

1) (78) (79)
2) (78) (79) (69)
3) (78) (79) (569) (58)
4) (478) (79) (569) (58)
5) (123478) (12379) (123569) (12358) are the MC's.

The upper bound is 4 and the lower bound is 2.

The MC's (123478) and (123569) include all states. All pairwise compatibles involving 1, 2 or 3 are included in these two compatibles and, since all entries in the implication table contain 1, 2 or 3, (123478) and (123569) are closed with respect to all implied two-number compatibles. No compatibles with more than two members are implied by either (123478) or or (123569), as may be ascertained from a check of the I_2 column of the flow table (this is the only column with more than two rows appearing as next-state entries). Hence, a solution corresponds to (123478) and (123569) as shown in Table

XV, where A replaces the former and B the latter. The lower bound is met. Further analysis would show that (13478) and (12569) also constitute a closed collection, but the resulting flow table is the same. It is *not* possible to obtain *any* reduction of Table XIV(a) without partitioning the rows into overlapping sets. Hence this example serves to illustrate, in a rather spectacular manner, the weakness of the classical merging procedures as applied to incompletely specified flow tables.

TABLE XV

	I_1	I_2	I_3	I_4
A	B, 0	A, 0	A, 0	A, 0
B	B, 0	B, 0	A, 0	B, 1

Example 5:

TABLE XVI

	1	2	3	4
1	–	3, 1	5, 1	2, 1
2	5, 0	–	–	–
3	6, 0	6, 1	–	–
4	–	–	2, 1	–
5	–	6, 0	1, 0	4, 1
6	3, 0	–	2, 0	3, 1

(a)

2	$\surd$				
3	36	56			
4	25	$\surd$	$\surd$		
5	X	$\surd$	X	X	
6	X	35X	$\surd$	X	12 34
	1	2	3	4	5

(b)

Process D yields the MC's (1234), (56) (36) (25) via the following steps.

1) (56)
2) (34) (36) (56)
3) (234) (36) (56) (25)
4) (1234) (36) (56) (25).

Note that the largest maximal incompatibles have two members, so the lower bound on the number of rows is 2. The upper bound is 4.

The table can be reduced to two rows only if a closed collection of compatibles can be found such that (ignoring members common to both compatibles) there are 3 members in each, or 2 members in one and 4 in the other, or 5 in one and 1 in the other. Since no MC has 5 members, we immediately eliminate the last-mentioned combination. The only 4-member compatible is (1234), and it implies (25), (56), and (36), so that it cannot be part of a 2-member collection. There are no 3-member compatibles other than those obtainable from (1234), and we cannot get two disjoint 3-member sets from (1234). Hence, at least 3 compatibles are required and the lower bound cannot be met.

Let us start by choosing two disjoint pairwise compatibles that imply no other sets, namely (25) and (36). We now need sets with states 1 and 4. The compatible (14) implies only (25), and so a solution which meets the new lower bound of 3 rows is (14) (25) (36). Table XVII is the reduced version with rows A, B, and C corresponding to the 3 compatibles. Another solution is (12) (34) (56). It is interesting to note that we cannot obtain a minimum solution if we use the largest MC.

TABLE XVII

	1	2	3	4
A	–	C, 1	B, 1	B, 1
B	B, 0	C, 0	A, 0	A, 1
C	C, 0	C, 1	B, 0	C, 1

VI. Conclusion

The problem considered here is one in which, given a finite, but very large set, the object is to find those members which are optimal in a specified sense. (The set consists of all flow tables with less than n rows that cover a given n-row flow table, and we are looking for the members with the smallest number of rows.) Since, for such problems, it is always possible to describe what might be termed a completely enumerative solution, in which all members of the set are examined, there is a sense in which all problems of this type are trivial. However, when the sets are extremely large, the completely enumerative procedure may be hopelessly impractical. It is then particularly desirable to develop direct procedures which generate the desired results without any enumeration. Between these extremes there may lie many partially enumerative methods, and in these cases it is frequently difficult to decide when a satisfactory procedure has been found. This will usually depend upon the complexity of the particular examples that are of interest.

The minimization of incompletely specified flow tables is an exceedingly difficult problem, and we have found no approach that suggests a direct procedure. Our results are as follows:

1) an analysis of the problem which we feel sheds considerable light on its general nature, and which provides a framework for partially enumerative procedures;
2) methods for directly generating the maximal compatibles, which seem to play an important role in the problem;
3) the application of the above ideas to a partially enumerative procedure which seems to be practical for minimizing flow tables containing up to about fifteen rows (without using a computer);
4) methods for obtaining useful upper and lower bounds on the number of rows in the minimum-row tables.

Appendix I

Proofs of Theorems

Proof of Theorem 1

Assume P covers Q. If P contains any state p which does not cover any state of Q, then p can be omitted, and the resulting table will still cover Q. Assume therefore that every state p_i in P covers at least one state of Q.

Let C_i be the set of all states of Q that are covered by p_i $(i=1, 2, \cdots, n)$. We shall now show that the collection of sets (C_i) is a closed collection, hence verifying the first part of the theorem. The second requirement for closure (definition 7) is clearly met, since if q_1 and q_2 belong to C_i and if p_i covers both q_1 and q_2, then for each I_k, $Z(p_i, I_k)=Z(q_1, I_k)$ and $Z(p_i, I_k)=Z(q_2, I_k)$, if both $Z(q_1, I_k)$ and $Z(q_2, I_k)$ are defined. Therefore, $Z(q_1, I_k)$ and $Z(q_2, I_k)$ are equal whenever they are both defined.

If q_1 is a member of C_j, then starting Q in q_1 and P in p_j, the application to both P and Q of any input sequence starting with I_k (and applicable to q_1) must result in P and Q giving the same output sequence (wherever the output of Q is defined), since by definition p_j *covers* q_1. After the first input I_k of the input sequence, P will be in some state p_v, and Q (if its next state is defined) will be in state q_2, which must be a member of C_v, because when the remainder of the input sequence (which can be any input sequence applicable to q_2) is applied to both P and Q, the outputs from both will be the same (wherever the output of Q is defined). Hence by the definition of "covers," p_v covers q_2, and by the definition of C_v, q_2 is a member of C_v. In summary then, for all states q_1 of C_j we have that if $N(q_1, I_k)$ is defined, then it is in C_v. Or C_j *implies* a set which is included in C_v. This is the first condition for closure (definition 7), and completes the proof of the first part of the theorem.

For the second part of Theorem 1 we are given the closed collection $(C_1, \cdots, C_j, \cdots, C_n)$. Using the synthesis procedure described in step 4) of Process A, we construct an n-state flow table P. We intend to prove that $P \geq Q$. In order to do this, we must show that for each state q of Q there is at least one state p of P such that p covers q. In particular, we will show that if q is a member of C_j, then p_j will cover q. We do this with an inductive argument.

Suppose that Q and P are in some pair of states q and p respectively, such that q is a member of the C-set covered by p. Then we will say that a *covering condition* exists.

1) If, with the two machines initially in a covering condition, any *length-one* input sequence applicable to Q is applied to both P and Q, then it is evident from step *b*) of the synthesis procedure that the two machines will again be in a covering condition.
2) Hypothesize that if, with P and Q initially in a covering condition, any *length-m* input sequence applicable to Q is applied to both machines, then they will again end up in a covering condition.
3) Now assume that, with the machines initially in a covering condition, any *length-m+1* input sequence applicable to Q is applied to both P and Q. Then, according to our hypothesis 2), after the first m members of the input sequence have been applied, the machines will again be in a covering condition. The last member of the length-$m+1$ input sequence can be considered as a length-one sequence applied with the machines in a covering condition. According to 1), this again leaves the machines in a covering condition. Therefore, given the hypothesis 2), it follows that if the machines are initially in a covering condition, the application to both of any length-$m+1$ sequence applicable to Q will leave them again in a covering condition. Since the hypothesis has been established for $m=1$ [in step 1], it follows that when P and Q are initially in a covering condition, they will remain in a covering condition after any input sequence applicable to Q is applied to both machines.
4) When the two machines P and Q are in any covering condition, the output of P is the same as the output of Q where the latter is specified. Hence, when P and Q are in any covering condition (and there is at least one for each state of Q), the output sequences from P will be the same as the specified portions of the corresponding output sequences from Q when any input sequence applicable to Q is applied to both machines. Therefore, P covers Q and the theorem is proved.

Proof of Theorem 2

First we wish to show that every subset of a compatible is a compatible. Let S be the set consisting of all compatibles of flow table Q. S is obviously closed. Let T be the collection of all subsets of compatibles. Let $S \cup T$ (the union of S and T) be the collection of all sets which are either in S or T. If we can show that $S \cup T$ is closed, we will have reached our objective. Certainly any set s in S meets the two conditions required for closure. If set t is in T, it will also meet the closure conditions since:

1) if both q_1 and q_2 are members of t, since t is a subset of some s in S, and each s meets the closure condition, then $Z(q_1, I_k) = Z(q_2, I_k)$ (if both are defined);
2) furthermore, since t in T is included in some s in S, any set *implied* by t (say b) is included in a set *implied* by s. Since a set implied by s is included in some s, b must be included in some s, so b is a member of T.

Therefore $S \cup T$ is a closed collection, hence every member of T is a compatible, as was to be proved. In particular, every pair of states included in a compatible is a compatible.

Now we shall show that if every 2-member subset of a is a compatible, then a is a compatible. Let S be the collection of all compatibles in flow table Q. S is closed. Let T be the collection of those sets, all of whose 2-member subsets are in S. Now we wish to show that $S \cup T$, the collection of all sets either in S or T, is closed.

If s is a member of S, it obviously meets the closure condition.

1) If t is a member of T, and q_1 and q_2 are in t, then by construction of T, the set (q_1, q_2) is in S, and therefore $Z(q_1, I_k) = Z(q_2, I_k)$ when both exist. This is condition 2 for closure.
2) If t implies the set b, then b^2 (any 2-member subset of b) is implied by t^2 (a 2-member subset of t); but t^2 is in S, so if b^2 is implied by t^2, b^2 is in S. Therefore, every 2-member subset of b is in S. Therefore, b is in T.

Proof of Theorem 3

If there is no set X in the chain generated by B such that for some k, $Z(x_1, I_k)$ and $Z(x_2, I_k)$ are defined but not equal (x_1 and x_2 being in X), then the chain generated by B with the addition perhaps of a number of single states forms a closed collection. It obviously meets the output condition for closure [condition 2) of definition 7]. Similarly, the "implication" condition for closure [condition 1)] is met, because the recursive process which generates the chain (definition 10) guarantees that if X is in the chain, every set implied by X is also in the chain. Since B is a member of the chain, and the chain is closed, B is a compatible.

On the other hand, if there is some set X in the chain and some k for which $Z(x_1, I_k)$, $Z(x_2, I_k)$ are both defined and unequal, then B is not a compatible. To show this, we define a *backward chain generated by X*.

1) X is in the backward chain.
2) If Y is in the backward chain, then every set which implies Y is added to the backward chain.

This backward chain has the following two properties:

1) If X is not a compatible then no member of the backward chain generated by X is a compatible.

Assume that at some step in the generating process all members of the backward chain are not compatibles. Then all sets added to the backward chain must imply sets which are not compatible. If Y implies Z (Z being a member of the backward chain), then Y is not compatible, since any closed collection which contains Y must also contain a set which includes Z. But since Z is not a compatible, there can be no such closed collection, so Y is not a compatible. Therefore, if at some point in the generating process all the sets thus far incorporated in the backward chain are not compatible, then all the sets of the backward chain are not compatibles. Since X is not a compatible, no member of the backward chain generated by X is a compatible.

2) Furthermore, the backward chain generated by X contains B (X being a member of the chain generated by B).

Assume that B is not in the backward chain. Assume further that we are building the chain generated by B and at some point none of the members of the partially constructed chain are members of the backward chain generated by X. Then any new member Y added to the chain must be implied by a set Z already in the chain. Since Z is not in the backward chain, neither is Y. Since initially the chain consists of B which is not in the backward chain by assumption, no set in the chain generated by B can be in the backward chain generated by X. This is the contradiction we seek since X is in the chain generated by B.

From 1) and 2) above we conclude that B is not a compatible.

Appendix II

Alternate Process for Generating Maximal Compatibles

1) If x_1 and x_2 are two states for which there is a k such that $Z(x_1, I_k)$ and $Z(x_2, I_k)$ are both defined and *unequal*, then the pair (x_1, x_2) are called *output incompatible*. List all pairs of states which are output incompatible.
2) Use these output incompatibles to break down the set of all states into larger and larger collections of smaller and smaller sets in such a way that the final collection of sets contains all the largest sets, none of which contain an *output incompatible* pair. This may be done in many ways, one of which is identical to Process E, assuming that the output incompatibles are listed as X's in an implication table.
3) We are now left with a collection of sets $(C_1, \cdots, C_j, \cdots, C_m)$. If there is a pair of states in C_j, x_1 and x_2, and a k such that $N(x_1, I_k)$ and $N(x_2, I_k)$ do not both appear in the same C_r, then x_1 and x_2

are called *implication incompatible*. List all implication incompatibles.

4) Again break down the sets of the collection $(C_1, \cdots, C_j, \cdots, C_m)$ with the *implication incompatibles*. Assuming these are listed as X's in an implication table, may be done with Process E.
5) Repeat step 3 on the resulting collection. Continue to cycle through steps 3), 4) and 5) until no more implication incompatibles are found. The resultant collection will consist of all maximal compatibles. The proof is fairly obvious in the light of the previous proofs.

ACKNOWLEDGMENT

The authors would like to thank S. Ginsburg of the National Cash Register Company for his comments on our manuscript, and for making available pre-publication copies[8–10] which indicated the existence of the problem. C. Y. Lee of the Bell Telephone Laboratories stimulated the authors' early work on the subject by providing a series of tailor-made counter examples which demolished a number of hypotheses (including some of his own). E. J. McCluskey, Jr. and R. H. Mayne of the Bell Telephone Laboratories supplied some helpful criticisms.

Logical Machine Design II: A Selected Bibliography[*]

DOUGLAS B. NETHERWOOD[†]

Summary—**The bibliography which appeared in the June, 1958, issue of these TRANSACTIONS is extended to a total of 777 titles. The original format is retained, but in this supplement the scope of material is restricted to technical publications pertaining to the logical design of machines.**

INTRODUCTION

ALTHOUGH this bibliography is a supplement to an earlier paper,[1] the criteria for the current selections differ from those of the original compilation. In LMD I (the original bibliography), a wide survey was attempted, with a sampling of papers from many areas related in some way to the abstract design of machines. In LMD II (the present supplement), the range of attention has been narrowed and intensified. LMD I was intended as a stimulus to a wide variety of persons, but LMD II is intended as a reference source for specialists who are working in the field of logical design.

SCOPE

The general area covered by this report has been called the Theory of Machines. However, there may scarcely be two persons who can agree precisely on definitions for the theory of machines or the limits of logical machine design. The difficulties of fixed definitions will be avoided by stating that some of the categories considered to be of the greatest relevance to this subject (listed in order of decreasing emphasis) are

1) Sequential machines
2) Algebras and models for automata,
3) Switching-net theory,
4) Combinational studies,
5) Component structures,
6) Boolean minimization,
7) Graphs and trees,
8) Neurological structures and analogies.

Disagreement may again be expected on the meaning of these categories and on their relative ranking, but it is believed that this listing will suffice as a general guide. Some other areas of lesser importance and occasional relevance are

9) Programming and coding,
10) Translation of language,
11) Reliability,
12) Memory techniques.

As in LMD I, accessibility of publications has been considered. An attempt has been made to cite every significant paper published in English language journals of wide circulation, particularly in issues of recent date. Historical interest has not been a factor in making selections; technical content alone has been considered. Some papers which are as yet unpublished have been listed, but these have existed at least in mimeographed form. A limited number of copies of them are available, in most cases, directly from the authors.

RUSSIAN PUBLICATIONS

In recent years, there has been a great increase in the number of Russian language papers on logical design, especially on the subject of switching theory. This note-

* Manuscript received by the PGEC, April 13, 1959.
† Electronic Components Lab., Wright Air Dev. Center, Wright-Patterson AFB, Dayton, Ohio.
[1] D. B. Netherwood, "Logical machine design: a selected bibliography," IRE TRANS. ON ELECTRONIC COMPUTERS, vol. EC-7, pp. 155–178; June, 1958.