On Information Lossless Automata of Finite Order

SHIMON EVEN, MEMBER, IEEE

Abstract—**A coding graph is a model which contains all the types of finite automata and codes as special cases. A test for information losslessness and for information losslessness of finite order of a coding graph is described. Efficient methods of computation are given which make the calculation simple and mechanizable. The application of the tests to finite deterministic automata is discussed and a method of constructing a decoder for a given finite automaton that is information lossless of finite order, is described.**

I. Introduction

IN HIS PAPER on information lossless (IL) automata [1] Huffman has defined IL automata (he called them IL finite state logical machines) as automata for which the knowledge of the initial state, the output sequence, and the final state of an experiment [2] uniquely determine the input sequence. He has described an efficient test to determine whether a given automaton is IL.

If an automaton is used as an encoding device for sequences, its information losslessness ensures that the coded sequence may be deciphered provided that the initial state and the final state are known. The major disadvantage of such a system is that the encoded message may have to be stored in full before the decoding may start. In view of this fact Huffman has defined an information lossless automaton of Nth order as one for which the knowledge of the initial state and the first N letters of the output sequence permits the decipherment of the first input letter. Since the state diagram (or the transition table) of the automaton is known, this means that the next state may be computed, and upon receiving one more output letter, one more input letter may be deciphered.

Huffman describes a test which determines, for any given automaton and any given positive integer N, whether or not the automaton is information lossless of the Nth order. The test is exhaustive in nature, in the sense that for each one of the states as an initial state all the possible experiments of length N are considered and searched for ambiguity with respect to the first input. The first objection to this procedure is that it is not efficient. The second is that if we just ask whether a positive integer N exists for which the automaton is IL of Nth order we have to keep testing for increasing N. Actually this testing may always be terminated since $n(n-1)/2$ (where n is the number of states of the automaton) is an upper bound for N.

Manuscript received November 14, 1963; revised May 8, 1964. This work was done while the author was with Sperry Rand Research Center, Sudbury, Mass.

The author is with Technion, Israel Institute of Technology, Haifa, Israel.

In Section II a general theory of information lossless coding graphs will be discussed. A coding graph is a model which contains deterministic automata, nondeterministic automata, variable length codes, and generalized automata [3] as special cases.

We shall describe a modified test for information losslessness which may be carried on any coding graph, and which is extended to test for information losslessness of finite order. This is to say that we shall be able to test whether for a given coding graph a positive integer N exists, such that the coding graph is Nth order IL or that no such N exists, in which case we say that the coding graph is not ILF (information lossless of finite order). Efficient methods in computation are described to make the calculation simple and mechanizable.

In Section III the application of the tests to finite automata is discussed and a method of constructing a decoder for a given finite automaton, that is, ILF, is described. Effort is made to make this construction as efficient as possible. It should be mentioned that Huffman has partially described in his paper [1] such a construction, but there is very little in common in the two presentations. First, he did use the information gained in his test for Nth order IL, which is different from our test. Second, his construction is directly in terms of hardware; compared to ours, which is presented in terms of state machines, and, therefore, is more easily analyzed and may be minimized before realization.

(Part of the material in this paper has been published in Russian [4], with an English abstract. Most of it was included in a doctoral thesis [6] the contents of which have been used in subsequent research [3], [6], [7]. The present publication makes available to the American reader those results which have not previously appeared in English.)

II. A General Theory of Information Losslessness

A. Statement of the Problems

Let G be a finite directed graph with n vertices $V_1, V_2, \cdots, V_n$ and labeled arcs satisfying the following conditions: There are m labels $x_1, x_2, \cdots, x_m$, and each of the arcs has one of these labels attached to it. Each one of the labels may be used any number of times, also no restrictions are imposed on the number of arcs leaving a vertex or the way they are labeled. From now on, a graph of this type will be called a *coding graph*. We shall always assume that the structure of the discussed coding graph is known.

Consider the following problem: Assume a finite path is selected on the coding graph G, and assume that the

vertex the path starts from is known and so is the vertex where the path ends, and the labels of all the arcs included in the path are given in the order they take place on the path. If the preceding information is always sufficient to determine the path uniquely, we shall say that the coding graph G is *information lossless* (IL).

Example 1: Let G be given by Fig. 1. To show that G is not IL, let V_1 be the initial vertex and V_4 be the final vertex, and let the labels of the path-arcs be $x_3x_1x_1x_1$. There are two paths which fulfill these conditions, namely:

$$V_1 \quad x_3 \quad V_4 \quad x_1 \quad V_4 \quad x_1 \quad V_4 \quad x_1 \quad V_4$$

and

$$V_1 \quad x_3 \quad V_3 \quad x_1 \quad V_1 \quad x_1 \quad V_2 \quad x_1 \quad V_4.$$

Our first problem is to find an algorithm to determine whether or not a given coding graph is IL.

We say that a coding graph G is *information lossless of finite order* (ILF) if, and only if, a positive integer $N(G)$ exists such that the knowledge of the initial vertex of a path and the first $N(G)$ labels uniquely determine the first arc of the path. However, it is assumed that the same statement does not hold for $N(G)$-1, that is, there exist an initial vertex and a sequence of $N(G)$-1 labels for which there exist two or more paths differing in at least their first arcs. We are using the least integer in order to make $N(G)$ unique for each G. Clearly, once this arc has been determined, the next vertex is known, and if one more label on the path is given, one more arc on the path can be determined. Basically this is a deciphering procedure with a delay of $N(G)$-1 arcs.

Our second problem is to find an algorithm to determine whether or not a given coding graph is ILF.

There exist coding graphs which are ILF but not IL, but they are of no practical interest.

Example 2: Let G' be given by Fig. 2. Clearly, G' is not IL, but is ILF with $N(G')=2$ by the fact that there are no arcs of length 2. There are less trivial cases of ILF coding graphs that are not IL, but in most practical cases there are no terminal vertices, so this situation does not arise. (A terminal vertex is one from which no arcs emanate.) This is shown in Theorem 1.

Theorem 1: If a coding graph is ILF and has no terminal nodes, then it is also IL.

Proof: Assume that there exists a coding graph G that is ILF but is not IL, then there must exist an initial vertex V_i, a final vertex V_j and a finite sequence of labels for which more than one path exists. Since G has no terminal nodes, there must be at least one way of constructing an infinite path with an initial vertex V_j. This means that there is an initial vertex V_i and an infinite sequence of labels for which at least two paths exist. This contradicts the assumption of ILF. Q.E.D.

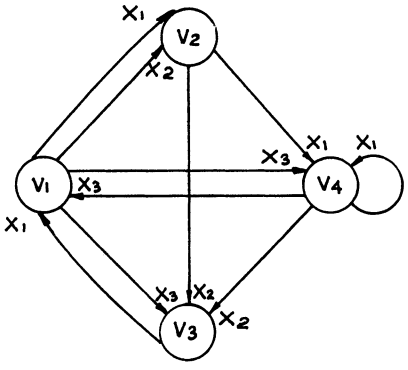


Fig. 1. The coding graph G.

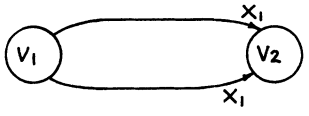


Fig. 2. An ILF coding graph which is not IL.

B. Solution of the Problems

An unordered pair of vertices (V_i, V_j) is said to be *compatible* if there exists a vertex V_k such that there exists an arc $V_k \rightarrow V_i$ and on arc $V_k \rightarrow V_j$ with the same label, or if there exists a compatible pair of vertices (V_k, V_l) such that there exist arcs $V_k \rightarrow V_i$ and $V_l \rightarrow V_j$ with the same label. No other pairs are compatible.

Let G be a coding graph. Define a directed graph $T(G)$, called the testing graph for G, in the following way:

1) The nodes of $T(G)$ correspond to compatible pairs of vertices of G.

2) If there exist an arc $V_k \rightarrow V_i$ and an arc $V_l \rightarrow V_j$ with the same label, and if (V_k, V_l) is a compatible pair, then $T(G)$ has a directed arc leading from the node corresponding to (V_k, V_l) into the node corresponding to (V_i, V_j) with the same label. No other arcs exist in $T(G)$.

Let us demonstrate the construction of $T(G)$ on Example 1. G (given by Fig. 1) may be presented in

TABLE I
REPRESENTATION OF G BY A TABLE

G	x_1	x_2	x_3
V_1	V_2	V_2	V_3, V_4
V_2	V_4	V_3	—
V_3	V_1	—	—
V_4	V_4	V_3	V_1

Table I where the vertices are given in the first column, and an arc $V_i \overset{x_k}{\rightarrow} V_j$ (read: from V_i to V_j with label x_k) is represented by V_j in the row headed by V_i and the column headed by x_k. For the sake of brevity we shall use tables with indices only. Therefore, G is given by Table II.

TABLE II
SIMPLIFIED TABLE OF G

G	1	2	3
1	2	2	3, 4
2	4	3	—
3	1	—	—
4	4	3	1

As may be seen in the table, some of the entries are empty, and some have more than one entry. If the coding graph has no terminal vertices, no row is empty for all labels.

The first compatible pair is (3, 4) because $1 \xrightarrow{3} 3$ and $1 \xrightarrow{3} 4$. Next, we have (1, 4) because $3 \xrightarrow{1} 1$ and $4 \xrightarrow{1} 4$. We proceed in the same manner and get $T(G)$ which is represented by Table III. The process is terminated when all the compatible pairs have been used as row headings.

TABLE III
TABLE REPRESENTATION OF $T(G)$

$T(G)$	1	2	3
(3, 4)	(1, 4)	—	—
(1, 4)	(2, 4)	(2, 3)	(3, 1), (4, 1)
(2, 4)	(4, 4)	(3, 3)	—
(2, 3)	(4, 1)	—	—
(1, 3)	(2, 1)	—	—
(4, 4)	(4, 4)	(3, 3)	(1, 1)
(3, 3)	(1, 1)	—	—
(1, 2)	(2, 4)	(2, 3)	—
(1, 1)	(2, 2)	(2, 2)	(3, 3), (3, 4), (4, 4)
(2, 2)	(4, 4)	(3, 3)	—

The graph $T(G)$ is shown in Fig. 3. In this example, G is isomorphic to a subgraph of $T(G)$ [the upper right part of $T(G)$], but for our purposes this is of no importance, and, therefore, we shall not discuss this point any more.

Theorem 2: A coding graph G is IL, if, and only if, no compatible pair of $T(G)$ has the same vertex repeated.

In our example we have: (1, 1), (2, 2), (3, 3), (4, 4), and; therefore, it is not IL. (This has been demonstrated before.)

Proof: Assume that some pair (e, e) is compatible. This means that there exists some compatible pair (a, b) such that $a \xrightarrow{i} e$ and $b \xrightarrow{i} e$ for some label i. Since all compatible pairs are reached from a single vertex by two (or more) paths with the same sequence of labels, the addition of i to this sequence will complete a sequence that corresponds to at least two paths that start in the same vertex and end in the same vertex. Therefore, G is not IL.

Now assume that G is not IL. Then, by definition, there must exist an initial vertex, a sequence of labels and a final vertex e which do not uniquely determine a path, i.e., there must exist at least two paths which satisfy these conditions. This allows two ways (or more) of assigning the intermediate vertices, if the length of the sequence is larger than one. In this case, the corresponding pairs of vertices are clearly compatible and, therefore, the pair (e, e) is compatible. Also if the length of the sequence is one, the pair (e, e) is compatible.

Q.E.D.

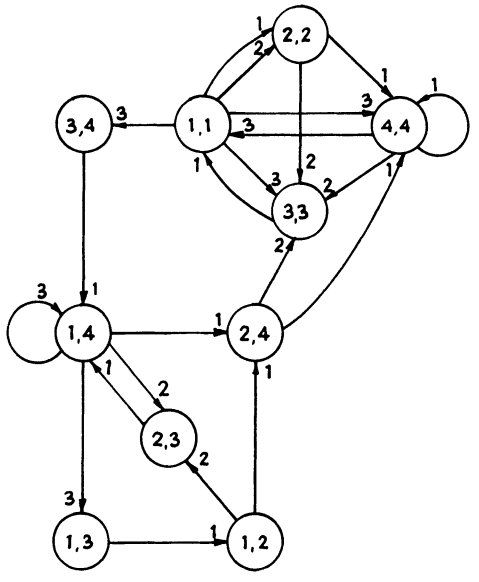


Fig. 3. The graph $T(G)$.

The proof suggests a simple way of constructing a minimum-length ambiguous path. This is done from right to left, and for our example will proceed as follows. The first pair generated in the table (Table III) which has the same vertex repeated is (4, 4). The corresponding label is 1, and the pair generating (4, 4) is (2, 4). The pair (2, 4) is generated by (1, 4) with the label 1, the pair (1, 4) is generated by (3, 4) with the label 1, and the pair (3, 4) is generated from 1 with the label 3. This is recorded in the following:

$$1 \xrightarrow{3} 3 \xrightarrow{1} 1 \xrightarrow{1} 2 \xrightarrow{1} 4$$

$$1 \xrightarrow{3} 4 \xrightarrow{1} 4 \xrightarrow{1} 4 \xrightarrow{1} 4.$$

The sequence of labels is 3111, the initial vertex is 1 and the final vertex is 4.

Theorem 3: A coding graph G is ILF if, and only if, $T(G)$ is loop free.

Proof: Assume that $T(G)$ is not loop free. By definition, every one of the compatible pairs is accessible from a single vertex of G by a pair of finite paths with different first arcs. Thus, we can find a pair of paths with the same label sequence that will lead into a loop in $T(G)$. Repeating the label sequence that causes the loop as many times as we wish, we may construct a pair of arbitrarily long different paths in G that start in the same vertex with different first arcs, and produce the same label sequence. Thus, G is not ILF.

Assume now that $T(G)$ is loop free. If G is not ILF then there exist arbitrarily long different paths with the same label sequence that start from the same vertex with different first arcs. This means that there is an arbitrarily long path in $T(G)$. Since the number of possible vertices in $T(G)$ is bounded [the bound is $n(n+1)/2$], arbitrarily long paths in $T(G)$ are possible only if a loop exists. Q.E.D.

Corollary: A coding graph G with no terminal vertices is ILF if, and only if, it is IL and $T(G)$ is loop free.

This follows immediately from Theorems 1 and 3. Since in most cases we shall deal with coding graphs with no terminal vertices, and the problem will be to find if they are ILF, it seems as though an efficient approach is to test first for IL.

If the order of an ILF coding graph G is $N(G)$, then there exist two different paths of length $N(G)$-1 emanating from the same vertex with the same sequences of labels and different first arcs. This means that one can find in $T(G)$ a path of length $N(G)$-2, but no path of length greater than or equal to $N(G)$-1. (The numbers are reduced by one, since it takes one step to get into the first compatible pair of such a path.)

Thus we have:

Theorem 4: If the length of a maximal path in $T(G)$ is m ($m \geq 0$) then $N(G) = m+2$.

The case $N(G) = 1$ is detected by absence of compatible pairs.

$T(G)$ and Theorems 2 and 3 suggest, then, an algorithm to determine if G is IL, and if it is ILF. In Section C we shall discuss methods to determine if a given graph is loop free, and if so, determine a longest path on the graph.

C. Remark on Practical Methods in Performing the Tests

Given a coding graph, we first generate for it the corresponding testing-graph. (The testing-graph is a coding graph by itself, but this is of no significance.) The testing-graph is a directed graph, and we want an efficient method of testing whether it is loop free or not.

In case of a directed graph with a small number of vertices this can easily be settled by inspection. However, even then one would actually draw the graph. When the number of nodes becomes large, this is not so easy to carry out; a different and more systematic way is desirable, especially if one wants to use a computer.

Let G be a directed graph with n vertices and let $C(G)$ be the connection matrix of G, i.e., $C(G)$ is a $n \times n$ matrix whose (i, j)th entry is 1 if there is an arc from vertex i to vertex j, and is 0 if otherwise.

Lemma 1: If a directed graph G is loop-free, then it has a terminal vertex.

Proof: If G has no terminal vertex, then for every vertex there exists at least one arc leading from it to some vertex of G; therefore, infinitely long paths may be constructed in G. Since G is finite, this implies that G has a loop. Q.E.D.

Lemma 2: If G is a directed graph, and i is a terminal vertex of G, then the graph G^i resulting from the deletion of vertex i and all arcs leading to it from G is loop free if, and only if, G is loop-free.

The proof is obvious.

It is also clear that the graph G^t that results from G by removal of *all* terminal vertices and arcs leading to them is also loop free, if and only if G is loop free. This is a simple inductive consequence of Lemma 2.

In the matrix representation the deletion of a vertex and all arcs leading to it (and from it) is achieved by the deletion of the row and column corresponding to this vertex from the matrix. We are now ready to describe the algorithm:

I) Given a directed-graph G, construct $C(G)$.
II) Find all rows of $C(G)$ that have only zeros in them. If there are none, go to step IV).
III) Delete all those rows and the corresponding columns from the matrix.
Repeat II) and III).
IV) If the matrix has not vanished completely, that is, if the result is a $m \times m$ matrix where $m > 0$, then G is not loop free. If the matrix has vanished, G is loop free.

Example 3: Let G be a directed graph with seven vertices labeled 1 through 7, and let its arcs be represented by the following set of ordered pairs: (1, 2), (2, 3), (4, 3), (4, 5), (4, 6), (5, 7), (6, 1), (6, 2), (6, 3), (6, 5), (6, 7), (7, 1). Let us now demonstrate the algorithm on G:

I)
$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

II), III) Row 3 has only zeros. Delete row 3 and column 3.

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

II) Row 2 has only zeros.

III)
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The following matrices result:

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad [0]$$

On the next step the matrix vanishes completely.

IV) G is loop free.

As we have shown, vertices from which paths of length zero emanate (terminal vertices) are eliminated at the first application of step III).

Lemma 3: If the length of a maximal path in G with vertex i as the beginning of the path is m, then i will be eliminated from the graph at the $(m+1)$st application of step III).

Proof: The proof is by induction. Assume the Lemma is true for $m-1$. Let m be the length of a longest path emanating from a given vertex i. Let j be the first vertex following i on such a path. Clearly, the length of a longest path following j is $m-1$. Therefore, j will be eliminated at the mth application of step III). At the next application of II) the row corresponding to vertex i will have only zeros in it, and in the $(m+1)$st application of III) it will be eliminated. Q.E.D.

Corollary: The length of a maximal path in a loopless graph G is equal to one less than the number of applications of III) in the algorithm.

As a consequence, not only does the test show whether G is loop free, but without any additional labor it determines the length of a maximal path in G in case it is loop free. (If the labels of the rows and columns are preserved during the procedure, a maximal path can be easily traced.)

In our example, step III) was applied seven times; therefore, the length of a maximal path in the graph is 6. (There is only one path, and it is 4-6-5-7-1-2-3.)

III. Application of the Theory to Finite Automata

A. Theoretical Explanation

In this section the tests developed in Section II will be applied to finite automata [8] which we shall simply call automata. The automata considered are of the Mealy model [9], where the output is a function of both the state and the input.

Definition: An automaton A is a quintuple $(K, \Sigma, \Delta, \delta, \lambda)$, where K is a finite nonempty set of "states"; Σ is a finite nonempty set of "inputs"; Δ is a finite nonempty set of "outputs"; δ is the "next state" function; and λ is the "output" function. δ maps $K \times \Sigma$ into K and λ maps $K \times \Sigma$ into Δ.

We may discuss the properties of an automaton in terms of *state diagram* of A. The state diagram is a directed graph where K is the set of vertices (the states), and for each vertex S ($S \in K$) and input x ($x \in \Sigma$) the graph has a directed arc from S to $\delta(S, x)$ labeled with $\lambda(S, x)$. Usually, the input also is preserved as the part of the label; namely, the arc is labeled by $x/\lambda(S, I)$, and such a state diagram is equivalent to the specifications of K, Σ, Δ and the functions δ and λ. However, in this section we have no use for the first part of the extended label (the input associated with the state transfer) and, therefore, it is deleted.

Theorem 5: The state diagram of an automaton A is a coding graph with no terminal vertices.

Proof: Since Σ is a nonempty set the state diagram will have arcs emanating from each of the vertices. Q.E.D.

As a matter of fact, if p is the number of inputs (the number of elements in Σ), then each vertex has exactly p arcs emanating from it. In this respect a state diagram is a proper special case of a coding graph in the sense that some coding graphs are not state diagrams. This is a result of the fact that in the definition of automata the case of "don't care" is excluded [10]. Actually, the tests for automata with "don't care" conditions are identical, except that there are many ways to define information losslessness for such automata, and the method of testing depends on the convention. For this reason we shall not pursue the subject any further.

A direct consequence of Theorem 5 is that our tests for IL and ILF are valid for automata, and we shall illustrate this consequence by two examples.

B. Examples

Example 4: Let $K = \{S_1, S_2, S_3, S_4, S_5\}$, $\Sigma = \{0, 1\}$, $\Delta = \{0, 1\}$ and $\delta(x, S)$, and $\lambda(x, S)$ are given in the entry in the row headed by S and the column headed by x of Table IV.

TABLE IV
Description of Automaton A_1

A_1	$x=0$	$x=1$
S_1	S_1, 1	S_3, 1
S_2	S_5, 0	S_2, 1
S_3	S_4, 0	S_1, 0
S_4	S_3, 0	S_2, 0
S_5	S_2, 1	S_1, 0

The first step is to find the compatible pairs, as shown in Table V. The derivation of the graph $T(A_1)$ is carried

TABLE V
GENERATION OF FIRST COMPATIBLE PAIRS

	$y=0$	$y=1$	Comp. Pairs
S_1	—	S_{13}	(13)
S_2	S_5	S_2	—
S_3	S_{14}	—	(14)
S_4	S_{23}	—	(23)
S_5	S_1	S_2	—

TABLE VI
GENERATION OF $T(A_1)$

	$y=0$	$y=1$
(13)	—	—
(14)	—	—
(23)	(15)(45)	—
(15)	—	(12)(23)
(45)	(12)(13)	—
(12)	—	(12)(23)

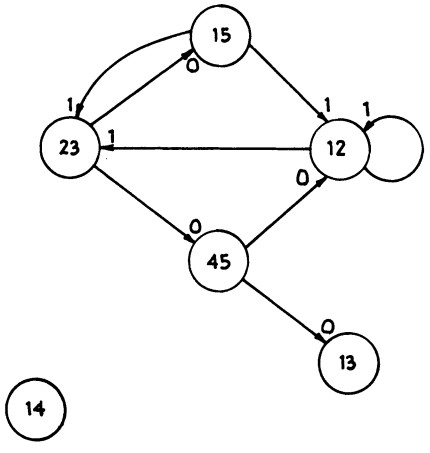


Fig. 4. $T(A_1)$.

TABLE VII
DESCRIPTION OF A_2

A_2	$x=0$	$x=1$
S_1	S_1, 0	S_2, 0
S_2	S_3, 0	S_4, 0
S_3	S_4, 1	S_3, 1
S_4	S_2, 1	S_1, 1

out in Table VI, where just the subscripts are used. The symbol y is used for the output.

The graph $T(A_1)$ is shown in Fig. 4.

Since no pair with the same symbol repeated twice has been generated in Table V or Table VI, the automaton A_1 is IL. However $T(A_1)$ contains loops, and therefore A_1 is not ILF. Using Fig. 4, one such loop is $(2, 3) \xrightarrow{0} (1, 5) \xrightarrow{1} (2, 3)$, and an infinite output sequence of the form 0 0 1 0 1 0 1 $\cdots$ with S_4 as initial state cannot be deciphered.

TABLE VIII
GENERATION OF THE COMPATIBLE PAIRS

	$y=0$	$y=1$	Comp. Pairs
S_1	S_{12}	—	(12)
S_2	S_{34}	—	(34)
S_3	—	S_{34}	(34)
S_4	—	S_{12}	(12)

(a)

	$y=0$	$y=1$
(12)	(13)(14)(23)(24)	—
(34)	—	(13)(14)(23)(24)
(13)	—	—
(14)	—	—
(23)	—	—
(24)	—	—

(b)

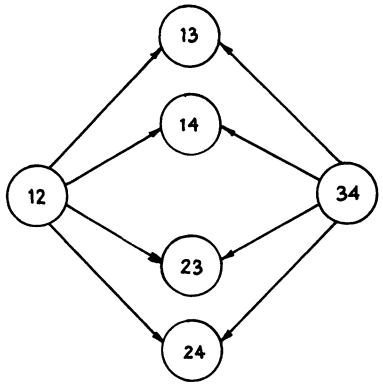


Fig. 5. $T(A_2)$.

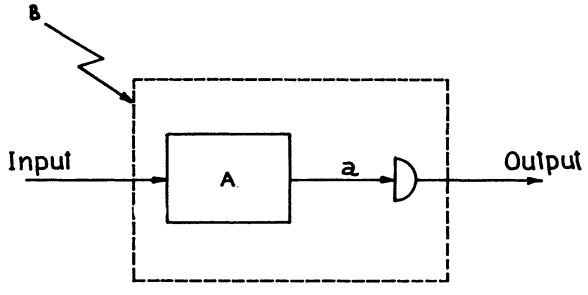


Fig. 6. Demonstration of a nonprime ILF automaton.

Example 5: The automaton A_2 is given by Table VII and its compatible pairs are generated in Table VIII (a) and VIII (b).

Figure 5 shows $T(A_2)$. Since no compatible pair contains the same symbol twice, A_2 is IL; and since $T(A_2)$ is loop free, A_2 is also ILF. The longest path in $T(A_2)$ is of length 1, therefore, A_2 is of the third order.

C. Prime and Nonprime ILF Automata

Let A be an ILF automaton and B be another automaton obtained from A by feeding the output of A to a delay element. (See Fig. 6.)

Let $N(A)$ denote the order of IL of automaton A.

Theorem 6: B is ILF and $N(B)=N(A)+1$.

Proof: Knowledge of the state of B guarantees the knowledge of the state of A. The first output of B is dependent only on the state of B. If the initial state of B and $N(A)+1$ output letters of the output sequence are known, then the initial state of A and the first $N(A)$ output letters of A are known, and, therefore, the first input letter may be determined. Thus, B is ILF and its order is at most $N(A)+1$.

Assume that $N(B)<N(A)+1$. This means that the knowledge of the initial state of B and $N(A)$ output letters will suffice to determine the first letter. The first output letter is included in the first state; therefore, we may get the same information by reading the signal at a (see Fig. 6) and in this case $N(A)$-1 letters will suffice This contradicts the fact that A's order is $N(A)$. Q.E.D.

An ILF automaton B for which an A may be found such that B is equivalent to A with a delay on the output is called a *nonprime* ILF automaton. If this is not the case the automaton is called a *prime* ILF automaton.

If for all states of an ILF automaton B, the output is independent of the input, B is nonprime, and an automaton A may be found which produces the same output one time unit earlier. (The first output of B is deleted without any loss of information.)

Example 6: Let B be given by Fig. 7. Since the output is independent of the input we may associate the output with the state. (See Fig. 8.) Now we want to remove one time unit as is done in the conversion from the Moore model to the Mealy model. This is done by associating the output with the input that causes transfer into this state. (See Fig. 9.) Finally, the reduction procedure may be applied to yield the resulting automation A. (See Fig. 10.) Let us test the automata B and A for ILF. The test table for B is given in Table IX and it is clear that B is ILF and its order is 2. Automation A is clearly of first order. (Also, it is identical to its inverse.)

TABLE IX

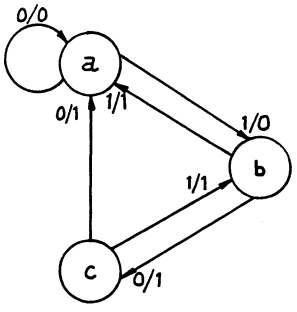
GENERATION OF THE COMPATIBLE PAIRS FOR B

	$y=0$	$y=1$
a	(a, b)	—
b	—	(a, c)
c	—	(a, b)
(a, b)	—	—
(a, c)	—	—

D. On the Inverse of ILF Automaton

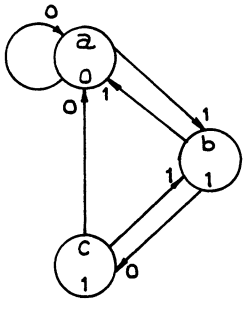
If the order of an automaton is 1, the inverse may be constructed by interchanging roles of input and output. (See Class I of information lossless circuits in Huffman's paper.) An example is given in Fig. 11 where (a) and (b) are inverses of each other.

If B is a nonprime ILF automaton and A is its corresponding prime ILF version (that is, B is the result of transmitting the output of A through a finite delay),



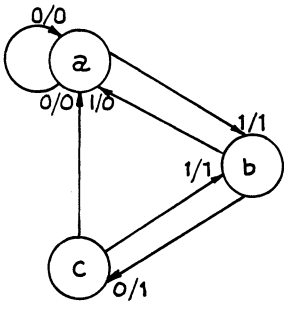
B	$x=0$	$x=1$
a	a, 0	b, 0
b	c, 1	a, 1
c	a, 1	b, 1

Fig. 7. Description of automaton B.



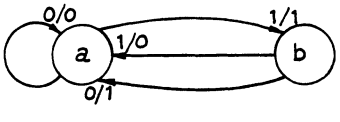
	$x=1$	$x=1$	y
a	a	b	0
b	c	a	1
c	a	b	1

Fig. 8. An equivalent Moore model of B.



	$x=0$	$x=1$
a	a, 0	b, 1
b	c, 1	a, 0
c	a, 0	b, 1

Fig. 9. A prototype of automaton A.



A	0	1
a	a, 0	b, 1
b	a, 1	a, 0

Fig. 10. Automaton A.

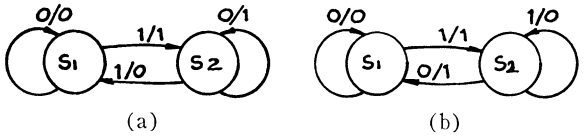


Fig. 11. Construction of an inverse for a Class I automaton.

then it is clear that the inverse of A is also the inverse of B. Therefore, if the inverse of a given automaton is required, we can build the inverse for its prime version.

The basic idea in the construction of the inverse of a given automaton A is that a combinational circuit may be constructed employing the initial state and $N(A)$ output letters of A as inputs, and supplying the first input letter to A as output. The $N(A)$ output letters are made available by storage in a shift register, and a copy of A is used which is first put in the initial state. The state of this copy is used as input to the combinational circuit. The output of the combinational circuit is used as input to the copy of A. (See Fig. 12.) We shall not use this method of construction because it does not necessarily yield an economical realization. However, the main idea is the same. Let us demonstrate the method by an example.

Example 7: Let the automaton A be given in Fig. 13. The test table and $T(A)$ are given in Fig. 14. Since the longest path in $T(A)$ is of unit length, the automaton is of third order (by Theorem 4). Let us define the states of the first version of the inverse[1] as triples (in general, N-tuples) whose first member is a state of A, whose second member is one of the output letters that can follow this state, and whose third member is another output letter that may follow this state and the first output letter. This is done for all states, and for each of them, for all possible sequences of output of length 2. In our example the inverse states are:

$(a, 0, 0)$ $(b, 0, 1)$ $(c, 0, 0)$ $(d, 1, 0)$
$(a, 1, 1)$ $(b, 1, 0)$ $(c, 0, 1)$ $(d, 1, 1)$.

We do not list $(a, 0, 1)$ as a state of the inverse because after an output 0, the next state is c, and the output 1 cannot be produced at this state.

Next, for every one of the inverse-states and one more output, we find the first input that was applied to A. This is possible because A is ILF of third order. The next inverse-state is the triple whose first member is the state into which A enters if started in the state which is the first member of the present inverse-state with the input we have found. The second member is the third member of the present inverse-state, and the third member is the present output of A. The transition table for our example is shown in Table X. The reduced form of the inverse of A is shown in Fig. 15. If the initial state of A is a, start the inverse in state S_4. The correspondence for all the states of A is as follows:

[1] Number of states will be reduced later, if possible.

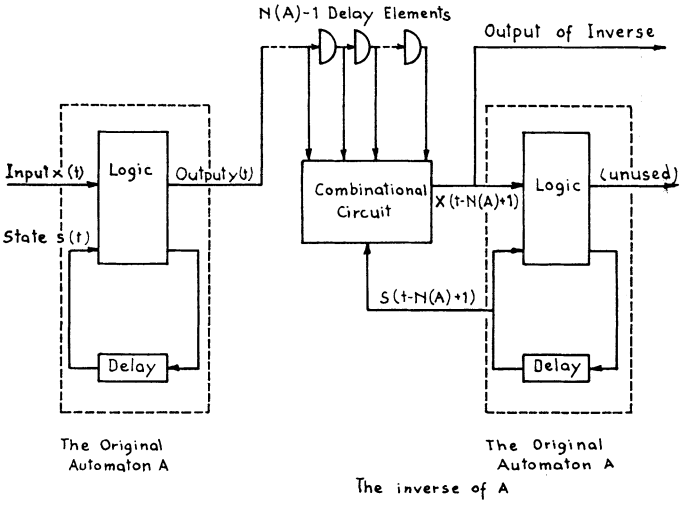
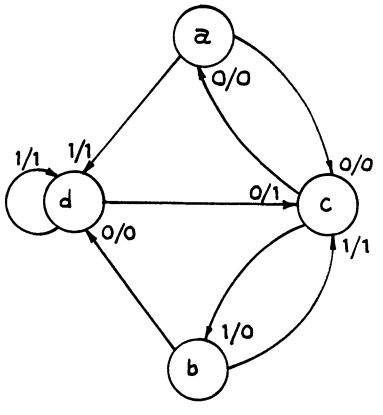
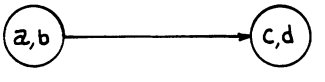


Fig. 12. A possible procedure for deciphering.



A	$x=0$	$x=1$
a	c, 0	d, 1
b	d, 0	c, 1
c	a, 0	b, 0
d	c, 1	d, 1

Fig. 13. Description of automaton A.



	$y=0$	$y=1$
a	c	d
b	d	c
c	(a, b)	—
d	—	(c, d)
(a, b)	(c, d)	(c, d)
(c, d)	—	—

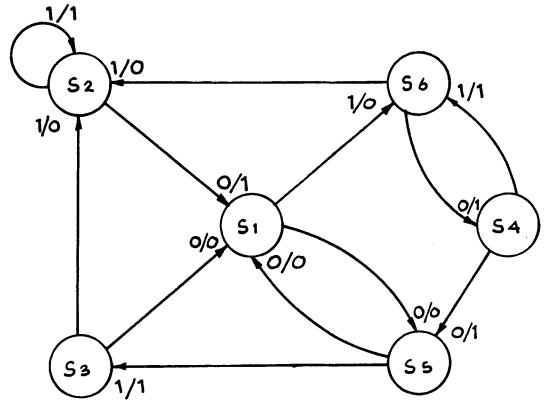
Fig. 14. Generation of $T(A)$.

$a \rightarrow S_4$
$b \rightarrow S_1$ or S_4
$c \rightarrow S_2$ or S_3 or S_5 or S_6
$d \rightarrow S_2$ or S_3 or S_5 or S_6.

TABLE X

PROTOTYPE FOR THE INVERSE OF A

	$y=0$	$y=1$
$(a, 0, 0)$	$(c, 0, 0), 0$	$(c, 0, 1), 0$
$(a, 1, 1)$	$(d, 1, 0), 1$	$(d, 1, 1), 1$
$(b, 0, 1)$	$(d, 1, 0), 0$	$(d, 1, 1), 0$
$(b, 1, 0)$	$(c, 0, 0), 1$	$(c, 0, 1), 1$
$(c, 0, 0)$	$(a, 0, 0), 0$	$(b, 0, 1), 1$
$(c, 0, 1)$	$(b, 1, 0), 1$	$(a, 1, 1), 0$
$(d, 1, 0)$	$(c, 0, 0), 0$	$(c, 0, 1). 0$
$(d, 1, 1)$	$(d, 1, 0), 1$	$(d, 1, 1), 1$



Inverse of A	$y=0$	$y=1$
S_1	$S_5, 0$	$S_6, 0$
S_2	$S_1, 1$	$S_2, 1$
S_3	$S_1, 0$	$S_2, 0$
S_4	$S_5, 1$	$S_6, 1$
S_5	$S_1, 0$	$S_3, 1$
S_6	$S_4, 1$	$S_2, 0$

Fig. 15. An inverse of A.

TABLE XI

SCHEME OF DECIPHERING A SEQUENCE

States of A:	a	c	b	d	c	a	d	d	c	b
Input sequence:		0	1	0	0	0	1	1	0	1
Output sequence:		0	0	0	1	0	1	1	1	0
States of inverse:	S_4	S_5	S_1	S_5	S_3	S_1	S_6	S_2	S_2	S_1
Output of inverse:		1	0	0	1	0	0	0	1	1

For demonstration, let the initial state of A be a, and the input sequence be 0 1 0 0 0 1 1 0 1. The resulting sequence, and the deciphering are shown in Table XI. Indeed, if we ignore the first two output letters of the inverse and the last two input letters of the original sequence, both read 0 1 0 0 0 1 1.

In our example the inverse has six states which requires three memory elements for realization. If we used the scheme of Fig. 12, the number of memory elements would be four.

It is interesting to note that the inverse of an ILF automaton is itself an ILF automaton, and every automaton is the inverse of its inverse.

REMARK

Other applications of the tests for coding graphs are given in [3] and [6]. A similar test is used in [7].

ACKNOWLEDGMENT

The author is indebted to Dr. E. F. Moore for an introduction to the subject; to Prof. H. Wang and Dr. M. Cohn for reading and criticizing the manuscript; to P. K. Hooper for a suggestion used in Part C of Section II; to Prof. D. A. Huffman, Prof. A. G. Oettinger and Prof. M. P. Schützerberger for guidance and encouragement.

REFERENCES

[1] D. A. Huffman, "Canonical forms for information-lossless finite-state logical machines," *IRE Trans. on Circuit Theory*, special Supplement, vol. CT-6, pp. 41–59, May 1959. Also in *Sequential Machines: Selected Papers*, E. F. Moore, Ed. Reading, Mass.: Addison-Wesley, pp. 132–156, 1964.

[2] E. F. Moore, "Gedanken-experiments on sequential machines" *Automata Studies*, Annals of Mathematics Studies, no. 34, C. E. Shannon and J. McCarthy, Eds. Princeton, N. J.: Princeton University Press, pp. 129–153, 1956.

[3] S. Even, "Generalized automata and their information losslessness," in *Switching Circuit Theory and Logical Design*, AIEE Special Publ. S-141, pp. 144–147, 1962.

[4] ——, "On information lossless automata of finite order," *Proc. Internatl Symp. on Theory of Switching Systems and Finite Automata*, September 1962 (Russian).

[5] ——, "On information lossless automata," Ph.D. dissertation Harvard University, Cambridge, Mass., January 1963.

[6] ——, "Tests for unique decipherability," *IEEE Trans. on Information Theory*, vol. IT-9, pp. 109–112, April 1963.

[7] ——, "Test for synchronizability of finite automata and variable length codes," *IEEE Trans. on Information Theory*, vol. IT-10, pp. 185–189, July 1964.

[8] M. O. Rabin and D. Scott, "Finite automata and their decision problems," *IBM J. Res. and Dev.*, vol. 3, pp. 114–125, April 1959. Also in *Sequential Machines: Selected Papers*, E. F. Moore, Ed. Reading, Mass.: Addison-Wesley, pp. 63–91, 1964.

[9] G. H. Mealy, "A method for synthesizing sequential circuits," *Bell Sys. Tech. J.*, vol. 34, pp. 1045–1080, September 1955.

[10] M. C. Paull and S. H. Unger, "Minimizing the number of states in incompletely specified sequential switching functions," *IRE Trans. on Electronic Computers*, vol. EC-8, pp. 356–367, September 1959.