3.3 SPECIAL CLASSES OF SEQUENTIAL MACHINES

3.3.1 Information Lossless Sequential Machines

A sequential machine can be considered to be an input-output device which encodes input sequences into output sequences. An interesting class of machines are those for which the input information can be decoded from the output information and some information about the initial and/or final state of the machine. Such machines will be referred to as information lossless (IL). We shall consider four types of information losslessness. If the input sequence can be determined from the output sequence and the initial state the machine is IL-I. If the input sequence can be determined from the output sequence and the final state the machine is IL-II. If the input sequence can be determined from the output sequence and the initial and final states the machine is general information lossless (GIL). Finally, if the first input of the input sequence can be determined from the first k+1 outputs of the output sequence and the initial state, the machine is information lossless finite of degree k (ILF(k)). Note that IL-I is identical to ILF(0). We shall now prove a series of lemmas which can be used to determine what IL properties are possessed by a given machine.

Lemma 3.8 A sequential machine M is IL-I if and only if \( Z(q, I_j) \neq Z(q, I_k) \) for all \( I_j \neq I_k \), and all states \( q, I \in Q \).

Proof: Assume that M is IL-I and \( Z(q, I_j) = Z(q, I_k) = z_k \) for some input \( I_j \neq I_k \) and state \( q \). If the machine is initially in state \( q \) and produces the output \( z_k \) for some input, the input cannot be determined uniquely, and the machine is not IL-I. This contradiction proves the necessity of the condition.

Assume that \( Z(q, I_j) \neq Z(q, I_k) \) for all \( I_j \neq I_k \) and all \( q, I \in Q \). Then given any initial state and single output the input can be uniquely determined. Hence any output sequence can be decoded, one input at a time (i.e., the input sequence can be determined uniquely), and the machine is IL-I. \( \square \)

Example 3.11 The machine \( M_1 \) of Figure 3.30 is IL-I. However \( M_2 \) is not IL-I since the initial state cannot be determined if the output 0 is produced from initial state 1. \( \square \)

\[
\begin{array}{c|cc|c|cc}
0 & x & 1 & 0 & x & 1 \\
1 & 1.0 & 2.1 & 1 & 1.0 & 2.0 \\
2 & 2.0 & 3.1 & 2 & 2.0 & 3.1 \\
3 & 4.1 & 1.0 & 3 & 4.1 & 1.0 \\
4 & 2.0 & 4.1 & 4 & 2.0 & 4.1 \\
\end{array}
\]

\[
M_1 \quad M_2
\]

\begin{eqnarray*}
\text{Figure 3.30} & \text{Sequential machines of Example 3.11} & \\
\text{Lemma 3.9} & \text{A sequential machine } M \text{ is IL-II if no next state-output entry } (q, z) \text{ appears more than once in the state table.} \\
\text{Proof:} & \text{Assume that no entry } (q, z) \text{ appears more than once. Then given a final state } q, \text{ and output sequence ending in } z, \text{ we can uniquely determine the last input and the next to last state. Iterating this procedure the entire input sequence can be determined, going backwards one input at a time.} \quad \square \\
\text{Example 3.12} & \text{The machine } M_1 \text{ of Figure 3.31 is not IL-II since the output sequence 0 final state 1 cannot be decoded. However, } M_2 \text{ is IL-II.} \quad \square \\
\end{eqnarray*}

\[
\begin{array}{c|cc|c|cc}
0 & x & 1 & 0 & x & 1 \\
1 & 1.0 & 2.1 & 1 & 1.0 & 2.0 \\
2 & 2.0 & 3.1 & 2 & 2.1 & 3.1 \\
3 & 4.1 & 1.0 & 3 & 4.1 & 1.1 \\
4 & 2.0 & 4.1 & 4 & 4.0 & 3.0 \\
\end{array}
\]
The following lemma gives a necessary condition for a machine to be IL-II.

Lemma 3.10 If some next state-output entry \((q_i, z_j)\) appears in a state table more than once, the machine is IL-II only if \((q_i, z_j)\) does not appear in different columns of the state table.

Proof If \((q_i, z_j)\) appears in two columns, say \(I_k\) and \(I_l\), the input cannot be determined uniquely if the output is \(z_j\) and the final state is known to be \(q_i\).

The following procedure may be used to determine whether a machine that satisfies Lemma 3.10 is IL-II.

Procedure 3.5
1. Construct a test table with rows corresponding to individual states of the machine \(M\) and columns representing all possible outputs. The entry in row \(q_i\), column \(z_j\) will be the set of states which have a next state output entry \((q_i, z_j)\) for some input \(I_j\). That is,

\[Q_{zi} = \{q_k | N(q_k, I_j) = q_i, Z(q_k, I_j) = z_j \text{ for exactly one input } I_j\}\]

If there exists two inputs \(I_j \neq I_m\), such that \(N(q_k, I_j) = N(q_n, I_m) = q_i\) and \(Z(q_k, I_j) = Z(q_n, I_m) = z_j\), then the machine is not IL-II and the procedure terminates. Note that if the condition of Lemma 3.10 is satisfied, this cannot occur in the first step of the procedure.

2. For every set of states \(Q_{ii}\) entered in the test table in Step 1, add a row to the test table. In every column \(z_k\) of the row \(Q_{ii}\), enter the set of states that lead to states in \(Q_{ii}\) and produce the output \(z_k\) for some input. If there exist two inputs \(I_j \neq I_m\), such that \(N(q_n, I_j) \subseteq Q_{ii}, N(q_n, I_m) \subseteq Q_{ii}\) and \(Z(q_n, I_j) = Z(q_n, I_m) = z_k\), then the machine is not IL-II. Note that \(q_n\) and \(q_i\) need not be distinct.

3. Repeat Step 2 for new sets generated as entries that are not already rows of the table, until no new rows have to be added.

The machine is IL-II, if two different inputs never lead to the same set of states in the test table, and produce the same output.

Example 3.13 Consider the machine of Figure 3.32(a) with outputs \(z_1\) and \(z_2\). We shall now determine whether this machine is IL-II with respect to the output \(z_1\) alone (i.e. the \(z_1\) output sequence and the final state uniquely determine the input sequence). The test table is shown in Figure 3.32(b). The entries \((3,0)\) and \((4,1)\) are repeated in the state table, but satisfy the necessary condition of Lemma 3.10. The top section of the test table is constructed in Step 1 of Procedure 3.5 and the bottom section during Step 2. No further sets of states need be added since all entries in the table also appear as row labels. The machine \(M_4\) is IL-II with respect to \(z_1\) since the necessary condition was satisfied throughout the procedure.

Since the machine \(M_4\) is IL-II, we can uniquely determine the input sequence corresponding to any final state and output sequence. For example, the \(z_1\) output sequence \(0000\) and final state \(3\) is generated by the input sequence \(1110\) from initial state \(1\) or \(2\). The \(z_1\) output sequence \(1100\) and final state \(3\) is generated by the input sequence \(0110\) from initial state \(3\) or \(4\).

We will now present a test procedure for GIL and then prove its validity.

Procedure 3.6 (Test for General Information Losslessness)
1. Construct a test table whose rows represent the individual states of the machine \(M\), and whose columns represent the set of possible outputs of \(M\). The entry in row \(q_i\), column \(z_j\) is the set of possible next states if the present state is \(q_i\) and the output \(z_j\).

2. Add rows to the test table corresponding to sets of states that appeared as entries in the table if such rows do not already exist. The entry in the row corresponding to the set of states \(Q_i\) and
output $z_i$ is the set of next states if the present state is in $Q_i$ and the output is $z_i$. This procedure is repeated until no new rows are added to the test table.

3. List the sets of states of $M$ which contain identical next-state-output $(N,Z)$ entries. If no state of $M$ contains two (or more) identical $(N,Z)$ entries and any set of states of $M$ which contains identical $(N,Z)$ entries is not contained in a set of states derived in Step 1 or Step 2, the machine is GIL. Otherwise, it is not GIL.

Example 3.14 A state table $M_s$ and its test table are shown in Figure 3.33. The entry $(3,1)$ appears in the set of rows $\{2,4\}$. The entry $(4,1)$ appears in the set of rows $\{1,4\}$. Since neither of these sets is contained in any row of the test table, $M_s$ is GIL.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.0</td>
<td>4.1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>3.1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1.0</td>
<td>2.0</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>3.1</td>
<td>4.1</td>
<td>4</td>
</tr>
</tbody>
</table>

$M_s$

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>34</td>
<td>34</td>
<td>12</td>
</tr>
</tbody>
</table>

Figure 3.33 (a) State table, (b) test table

Since $M_s$ is GIL, we should be able to determine the input sequence for any output sequence, provided the initial and final states are known. The test table is also useful for decoding an output sequence (i.e., determining the input sequence) generated by a GIL machine for given initial and final states.

Consider decoding the output sequence 00010 generated by $M_s$ with the initial state 1 and final state 2. From the test table, the state after the first output must be 3 (entry in row 1, column $z = 0$). Similarly, the next state can be 1 or 2 (entry in row 3, column $z = 0$). The next two states can be 3 or 4. This information may be represented as follows:

Output: 0 0 0 1 0
State: 1 3 12 34 34 2

The last input must produce an output of 0 and the next state of 2, from the state of 3 or 4. From the state table it can be determined that this can only occur if the previous state was 3 and the input was $x = 1$. Now, the next to last transition must be from state 3 or 4 to state 3 with output 1. From $M_s$, we can conclude that the input must be 0 and the state must be 4. Continuing in this manner, the entire input sequence can be obtained as 01001.

Example 3.15 In the state table $M_b$, the entry (2,0) appears in the set of rows $\{1,3\}$ and the entry (3,0) appears in the set of rows $\{1,4\}$. Since the set $\{1,3\}$ is a row of the test table, $M_b$ is not GIL. The output sequence 00010 with initial state 1 and final state 2 can be generated by input sequence 00101 or 11000 and hence cannot be uniquely decoded.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.0</td>
<td>3.0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>2.1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3.1</td>
<td>2.0</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1.1</td>
<td>3.0</td>
<td>4</td>
</tr>
</tbody>
</table>

$M_b$

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>34</td>
<td>34</td>
<td>12</td>
</tr>
</tbody>
</table>

Figure 3.34 (a) State table, (b) test table

Lemma 3.11 A sequential machine $M$ is general information-lossless if and only if no state has two or more identical entries and no two states with the same $(N,Z)$-entry (next state and output) are in the same set of the test table.

Proof: Necessity: Assume that $(q_i, z_i)$ appears as the next state-output entry of states $q_a$ and $q_b$ which are in the same set of the test table.
of $M$. Since $q_a$ and $q_b$ are in the same set of the test table, there is an initial state $q_0$ and two input sequences $X_0, X_1$ which lead to states $q_a$ and $q_b$, respectively, with the same output sequence $Z$. Therefore, the initial state $q_0$, the final state $q$, and the output sequence $Z_{2j}$ do not uniquely determine the input sequence and hence the state table is not information lossless.

**Sufficiency:** For any output sequence, given the initial state and final state, we can list the possible set of states which generate each output in the output sequence. Each of these corresponds to a set of states in the test table. Let $q_j$ be the final state and let $z_j$ be the last output of the sequence. Then, from the state table, we can determine the possible set of next to last states. If the conditions of the lemma are satisfied, no two states with the entry $(q_i, z_j)$ is contained in the same set of states of the test table. Therefore, the next to last state and hence the last input can be determined uniquely. This procedure can be repeated until the entire input sequence is determined. □

To determine the ILF property, it is helpful to define an **IL implication graph** of a state table $M$. The graph contains a node for every pair of states $(q_i, q_j)$ (where may equal $q_j$) such that for some state $q$ and inputs $I_k, I_m, I_k \neq I_m, N(q, I_k) = q_j, N(q, I_m) = q_i$, and $Z(q, I_k) = Z(q, I_m)$. If the graph contains a node $(q_i, q_j)$ and $N(q_i, I_k) = q_m, N(q_j, I_m) = q_m$, and $Z(q_i, I_k) = Z(q_j, I_m)$ then the node $(q_i, q_j)$ to a node labeled $(q_m, q_m)$.

**Example 3.16** The IL implication graph of $M_3$ of Example 3.14 is shown in Figure 3.35.

![Figure 3.35 Implication graph of $M_3$](image)

For table $M$, of Figure 3.36(a), the IL implication graph is shown in Figure 3.36(b).

**Lemma 3.12** A table is ILF($k$) if and only if the IL implication graph has no cycles, no repeated state nodes $(q_i, q_j)$, and the longest chain is of degree $k$ (i.e., has $k$ nodes and $k - 1$ arcs).

**Proof:** Necessity: If the IL implication graph has a repeated state node $(q_i, q_j)$ there is an initial state $q_i$ and an output sequence $Z$ which can be generated by two input sequences $X_1, X_2$ and result in the same final state $q_j$. If the machine is in state $q_i$, the input sequence $X_1, X_2$ will produce the same output sequence where $X_1$ is any input sequence. Therefore, the machine is not ILF($k$), for any $k$. If the IL implication graph has a cycle, there is an initial state and an arbitrarily long output sequence which can be generated by two or more input sequences. Therefore $M$ is not ILF($k$). If the longest chain in the implication graph is of length $m > k$, then there exist an initial state and two different input sequences of length $m + 1$ which produce the same output sequence.

**Sufficiency:** If the longest chain in the IL implication graph is of degree $k$, then given any initial state and output sequence of length $k + 1$, it is impossible to generate this output sequence by two input sequences which differ in the first input. Thus, the first input can always be determined from the initial state and the first $k + 1$ outputs, and the machine is ILF($k$). □

**Example 3.17** From the implication graph of $M_4$ (Figure 3.35), $M_4$ is ILF(2) since the longest chain is of degree 2. The output sequence 001 from initial state 3 can be decoded to have the first input equal to 1. The output sequence 010 from initial state 3 has the first input equal to 1. However, the output sequence 00 from initial state 3 can be produced by either of the input sequences 00 or 10. From Figure 3.36(b), $M_4$ is not ILF. The output sequence 0111 ... from initial state 3 cannot be decoded. □
Sequential Circuits—Selected Topics

It is interesting to consider the relationships between the various types of information losslessness. From the definitions IL-I is equivalent to ILF(0), IL-I implies GIL, and IL-II implies GIL. Table $M_1$ is IL-I but not IL-II. Table $M_2$ is IL-II but not IL-I. Therefore, in general, IL-I and IL-II are incommensurate. Table $M_3$ is GIL but not IL-II and Table $M_4$ is IL but not ILF. Finally if a table $M$ is not GIL, then there is some initial state $q_0$ and output sequence $Z$ which results in the same final state for two different input sequences and hence $M$ is not ILF. These relationships between the various types of information losslessness can be represented by the lattice of Figure 3.37, which shows that if a machine is IL-I (or ILF(0)) it is ILF(k) which in turn implies it is GIL. Similarly IL-II implies GIL.

![Figure 3.37 Lattice of information lossless relations](image)

For an ILF machine $M$, it is possible to define an inverse machine $M^{-1}$ whose inputs are the outputs of $M$. If the input sequence $I$ applied to $M$ produces an output sequence $Z$, then $M^{-1}$ will produce an output sequence $I$ with some inherent delay, if the input sequence $Z$ is applied to it. (See Problem 3.24.)

3.3.2 Linear Sequential Circuits

Another important class of circuits is linear sequential circuits. These circuits have important coding theory applications for encoding and decoding of data. We shall first consider linear combinational circuits that were introduced in Section 2.5.

A function $f(x_1, x_2, ..., x_n)$ was defined to be linear if it could be expressed in the form

$$f(x_1, x_2, ..., x_n) = a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus ... \oplus a_n x_n,$$

where $a_i = 0, 1$, $0 \leq i \leq n$. An alternative representation of linear functions is useful for our discussions here. A function $f(x_1, x_2, ..., x_n)$ is said to be linear if it satisfies the principle of superposition

$$f(a_1 x_1, a_2 x_2, ..., a_n x_n) = a_1 f(x_1, 0, 0, ..., 0) + a_2 f(0, x_2, 0, 0, ... \oplus a_n f(0, 0, ..., 0, x_n),$$

where $a_i, 1 \leq i \leq n$ are constants. For the binary case, $a_i = 0$ or 1 and the arithmetic is modulo 2 (the sum of an even number of 1's is 0, the sum of an odd number of 1's is 1). Hence, for binary functions, linearity implies

$$f(0, 0, ..., 0) = 0 \text{ and }$$

$$f(x_1, x_2, ..., x_n) = f(x_1, 0, ..., 0) \oplus f(0, x_2, 0, ..., 0) \oplus ... \oplus f(0, 0, ..., 0, x_n).$$

It may appear that there is an inconsistency between the two definitions when $a_0 = 1$ in the first case. This apparent inconsistency can be resolved by expressing the function of $n$ variables in the first definition as a function of $n + 1$ variables for applying superposition. Let the added variable be $x_0$. Then

$$f(x_0, x_1, x_2, ..., x_n) = f(x_0, 0, 0, ..., 0) \oplus f(0, x_1, 0, 0, ..., 0) \oplus ... \oplus f(0, 0, ..., 0, x_n),$$

where $f(x_0, 0, ..., 0) = a_0$ (a constant).

(Note that the same technique is used in the analysis of other types of linear systems.) Thus, complementation is a linear function by the first definition, since $\bar{x} = 1 \oplus x$, but has to be treated as a function with two inputs, namely, 1 and $x$, in order for the superposition principle to be applicable. The AND and OR are easily shown to be nonlinear. For the AND of two inputs, $f(x_1, x_2) = x_1 x_2$. However, $f(x_1, 0) \oplus f(0, x_2) = 0 \oplus 0 = 0 \neq f(x_1, x_2)$ since $f(x_1, x_2) = 1$ for $x_1 = x_2 = 1$. Similarly, an OR gate is a nonlinear element since $f(x_1, x_2) = x_1 + x_2$, but $f(0, x_2) \oplus f(x_1, 0) = x_2 \oplus x_1 \neq x_1 + x_2$.

The only linear combinational functions are:

$$f(x_1, x_2, ..., x_n) = a_1 x_1 \oplus a_2 x_2 \oplus ... \oplus a_n x_n,$$

where $a_i = 0, 1$, $0 \leq i \leq n$, and

$$x_i \oplus x_k = \bar{x}_i x_k + x_i \bar{x}_k.$$
Thus, a linear combinational circuit can be considered to be composed exclusively of modulo 2 adders and is represented as shown in Figure 3.38.

The operation of modulo 2 addition is commutative and associative. In addition, \( f(x) \oplus f(x) = 0 \) and \( x_i = 1 \oplus x_i \).

![Diagram](image)

**Figure 3.38** Representation of linear combinational circuits

A sequential circuit is linear if the output logic and the combinational logic for the next state function using D flip-flops as memory elements are linear. Such a circuit can be represented as shown in Figure 3.39. In the rest of this section, we shall use the + symbol to represent modulo 2 addition whenever there is no ambiguity.

![Diagram](image)

**Figure 3.39** Representation of linear sequential circuit

A linear sequential circuit with \( n \) binary inputs, \( k \) state variables and \( m \) binary outputs can be described by the following set of equations.

\[
Y_i = \sum_{j=1}^{k} \alpha_{ji} x_j + \sum_{j=1}^{k} \beta_{ji} y_j, \quad 1 \leq i \leq k
\]

\[
z_i = \sum_{j=1}^{n} \gamma_{ji} x_j + \sum_{j=1}^{n} \delta_{ji} y_j, \quad 1 \leq i \leq m
\]

Using row matrices to represent inputs, states and outputs, a linear sequential circuit can be defined by the following matrix equations:

\[
Y^T = Ax^T + By^T
\]

\[
z^T = Cx^T + Dy^T
\]

where \( Y = [Y_1 \ Y_2 \ldots Y_k] \), \( x = [x_1 \ x_2 \ldots x_n] \), \( y = [y_1 \ y_2 \ldots y_k] \), \( z = [z_1 \ z_2 \ldots z_m] \) and \( x^T \) is the transpose of \( x \).

\[
A = [\alpha_{ij}] \text{ is a } k \times n \text{ matrix}
\]

\[
B = [\beta_{ij}] \text{ is a } k \times k \text{ matrix}
\]

\[
C = [\gamma_{ij}] \text{ is an } m \times n \text{ matrix}
\]

\[
D = [\delta_{ij}] \text{ is an } m \times k \text{ matrix}
\]

The behavior of linear sequential circuits can be studied using linear algebra. Alternatively, the transfer function approach which is widely used in the study of linear systems, can also be used for linear sequential circuits. We shall use the latter method.

We will first consider linear sequential circuits without feedback. Let the circuit have \( n \) inputs, \( k \) state variables, and \( m \) outputs. Since there is no feedback in the circuits, the state variables can be ordered so that any next state variable \( Y_i \) depends only on the inputs and the state variables \( y_j, j > i \). Such a circuit is defined by the following set of equations:

\[
Y_k = \alpha_{k1} x_1 + \alpha_{k2} x_2 + \ldots + \alpha_{kn} x_n = \sum_{j=1}^{n} \alpha_{kj} x_j
\]

\[
Y_{k-1} = \sum_{j=1}^{n} \alpha_{(k-1)j} x_j + \beta_{(k-1)k} y_k
\]

\[
Y_i = \sum_{j=1}^{n} \alpha_{ij} x_j + \sum_{j>i}^{n} \beta_{ij} y_j
\]

\[
z_i = \sum_{j=1}^{n} \gamma_{ij} x_j + \sum_{j>i}^{k} \delta_{ij} y_j
\]
If we define a delay operator \( D \) such that \( Dx(t) = x(t-1) \) and \( D^k x(t) = x(t-k) \), then we can derive an analytic expression for the output of a linear sequential circuit without feedback. It is easily shown that this operator is linear. Since \( y_j = D(Y_j) \) each state variable can be expressed as a (mod 2) sum of input variables \( x_i \) and delayed input variables \( D^k x_i \). Therefore, each output can be similarly expressed. For the circuit of Figure 3.40, which is the most general single input, single output feedback free linear sequential circuit,

\[
\begin{align*}
z &= a_0 x + y_1 \\
    &= a_0 x + D y_1 \\
    &= a_0 x + D(a_1 x + y_2) \\
    &= \ldots \\
    &= a_0 x + a_1 D x + a_2 D^2 x + \ldots + a_k D^k x.
\end{align*}
\]

The \( D \) operator is a linear operator and hence expressions in this operator can be factored, multiplied and, in general, manipulated in the usual manner, recalling that all sums are modulo 2. The transfer function \( T = \frac{z}{x} \) specifies the relation between output and input as a polynomial in the operator \( D \)

\[
T = \frac{z}{x} = a_0 + a_1 D + a_2 D^2 + \ldots + a_k D^k.
\]

The derivation of the transfer function for a circuit without feedback and the synthesis of such a circuit from the transfer function are easily done by comparison of the circuit or the transfer function with the canonical representation (Figure 3.40).

Example 3.18  (a) For the circuit of Figure 3.41, \( a_4 = a_3 = 1, \ a_0 = a_2 = a_9 = 0 \). Therefore, the transfer function is given by:

\[
T = \sum_{i=0}^4 a_i D^i = D^2 + D^4.
\]

(b) To realize a circuit with the transfer function, \( T = 1 + D + D^4 \), we note that \( a_0 = a_1 = a_4 = 1 \) and \( a_2 = a_3 = 0 \). The circuit of Figure 3.42 is obtained from the general form of Figure 3.40 by deleting the connection in the modulo 2 adder whenever \( a_i = 0 \).

Figure 3.40  Canonical linear sequential circuit without feedback

Figure 3.41  A linear sequential circuit of Example 3.18(a)

Figure 3.42  Feedback free circuit for Example 3.18(b)

Transfer functions are also useful for the analysis and synthesis of linear circuits with feedback. Given a linear circuit with state variables and outputs defined by the equations

\[
\begin{align*}
Y_j &= \sum_{i=1}^n \alpha_{ij} x_i + \sum_{i=1}^k \beta_{ij} y_j \\
z_i &= \sum_{j=1}^n \gamma_{ij} x_i + \sum_{j=1}^k \delta_{ij} y_j.
\end{align*}
\]

every output \( z_i \) may be expressed as a linear function of the inputs \( x_i \), the previous values of \( x \) and the previous values of \( z_i \). For a linear circuit with a single input, single state variable and single output,

\[
\begin{align*}
Y &= \alpha x + \beta y \\
z &= \gamma x + \delta y
\end{align*}
\]

Applying the \( D \) operator to (1), we obtain
Sequential Circuits—Selected Topics

\[ DY = y = \alpha Dx + \beta Dy \]
\[ : y + \beta Dy = \alpha Dx \]
\[ y = \frac{\alpha D}{1 + \beta D} x. \]

Substituting in (2), we obtain

\[ z = \gamma x + \delta \left( \frac{\alpha D}{1 + \beta D} \right) x \]
\[ z(1 + \beta D) = z(1 + \beta D) = \gamma x(1 + \beta D) + \delta \alpha Dx \]
\[ z + \beta Dz = \gamma x + \gamma \beta Dx + \delta \alpha Dx \]
\[ z = \gamma x + \gamma \beta Dx + \delta \alpha Dx + \beta Dz. \]

If the linear circuit has \( k \) state variables, there will be \( k \) simultaneous linear equations similar to (1), which can be solved to obtain equations for the state variables in terms of the inputs \( x \) and their past values. Each output \( z_i \) can then be expressed as a linear function of the inputs \( x \), their previous values and the previous value of \( z_i \). For the remainder of this section, we shall consider only single input, single output linear circuits. Multiple outputs can be treated by considering each output separately and the principle of superposition can be used for treating multiple inputs.

![Figure 3.43 Canonical linear sequential circuit with feedback](image-url)

Special Classes of Sequential Machines

A single input, single output linear sequential function has a transfer function of the form

\[ T = \frac{z}{x} = \frac{a_0 + a_1 D + a_2 D^2 + \ldots + a_n D^n}{1 + b_1 D + b_2 D^2 + \ldots + b_n D^n}, \]

where

\[ a_1, b_i = 0,1. \]

For the circuit of Figure 3.43

\[ z = a_0 x + y_1 \]
\[ = a_0 x + D(a_1 x + b_1 z + y_2) \]
\[ = \ldots \]
\[ = a_0 x + D a_1 x + D^2 a_2 x + \ldots + D^n a_n x \]
\[ + Db_1 z + D^2 b_2 z + \ldots + D^n b_n z. \]

The transfer function is

\[ \frac{z}{x} = \frac{a_0 + a_1 D + a_2 D^2 + \ldots + a_n D^n}{1 + b_1 D + b_2 D^2 + \ldots + b_n D^n}. \]

Thus, this circuit is a canonical representation of a single input, single output linear sequential circuit. (Note that the canonical feedback free circuit corresponds to the special case \( b_i = 0 \) for all \( i \).) Given the transfer function of a binary linear function, a circuit to realize this function can be derived from the canonical realization. Thus, if \( T = (D^2 + 1)/(D^3 + D + 1) \), \( a_1 = a_2 = 0 \), \( a_2 = a_0 = 1 \), \( b_1 = 1 \), \( b_2 = 0 \), \( a_i = b_i = 0, \) for \( i \geq 4 \). With these values of \( a_i \) and \( b_i \), the canonical circuit realization of \( T \) is as shown in Figure 3.44.

Some transfer functions may not correspond to physically realizable circuits. For instance, if \( T = z/x = (D + 1)/D \), then \( Dz = (D + 1)x = Dx + x \). Therefore, the value of \( z \) at \( t - 1 \) depends on the value of \( x \) at \( t \). Such a circuit is impossible to realize since no value of \( z \) can depend on future values of \( x \) in any physical circuit. For a transfer function \( T \) to be physically realizable if \( T = P_1(D)/P_2(D) \) where \( P_1(D) \) is the numerator polynomial in \( D \) and \( P_2(D) \) is the denominator poly-
mial, the smallest nonzero power of $D$ in $P_2(D)$ must not exceed the smallest power of $D$ in $P_1(D)$.

If the output of a circuit $C$ with transfer function $T = P_1(D)/P_2(D)$ is input to a circuit $C'$ with transfer function $T' = P_3(D)/P_4(D)$, the transfer function of the composite circuit is equal to the product $T \cdot T'$. (Exercise). If $P_2(D) = P_2(D)$ and $P_4(D) = P_4(D)$, then $T \cdot T' = 1$. Hence, the output of the composite machine is equal to the input (if both are physically realizable) and $T'$ is said to be the inverse of $T$, and is denoted as $T^{-1}$.

The transfer function of a given circuit is easily derived as illustrated in the following example.

Example 3.19 For the circuit of Figure 3.45.

$$
\begin{align*}
T & = A + B \\
& = A + DA \\
& = DC + D^2C \\
& = D^2E + D^3E.
\end{align*}
$$

But

$$
E = C + B + x.
$$

Therefore

$$
C = DE = DC + DB + Dx
= DC + D^2C + Dx.
$$

Consequently

$$
C + DC + D^3C = Dx
$$

and

$$
C = \frac{Dx}{D^3 + D + 1}.
$$

Therefore

$$
\begin{align*}
z & = \frac{D^2x}{D^3 + D + 1} + \frac{D^3x}{D^3 + D + 1} \\
T & = \frac{D^3 + D^2}{D^3 + D + 1}
\end{align*}
$$

The transfer functions of circuits with several binary inputs and/or outputs can be similarly derived.

Another important method of characterizing a function in the theory of linear systems is its impulse response. The impulse response of a single input, single output, linear binary circuit is defined as the output sequence produced in response to the input sequence 100...0, assuming that the outputs of all delays are initially 0. The impulse response of a circuit can be determined directly from the transfer function. For a single input feedback free circuit, the transfer function is of the form

$$
T = a_0x + a_1Dx + a_2D^2x + \ldots + a_kD^kx
$$

which corresponds to the circuit of Figure 3.40. For this circuit, it is apparent that the impulse response is

$$
a_0a_1a_2\ldots a_{k-1}a_k00\ldots0
$$
For the circuit of Figure 3.41, the impulse response is

\[ a_0a_1a_2a_3a_400 \ldots = 00101000 \ldots 0. \]

For a general binary linear sequential circuit, the transfer function is of the form

\[ T = \frac{a_0 + a_1D + a_2D^2 + \ldots + a_nD^n}{1 + b_1D + b_2D^2 + \ldots + b_nD^n} \]

which corresponds to the canonical circuit of Figure 3.43. The impulse response \( h \) of this canonical circuit is defined by a sequence of symbols \( C_0C_1C_2C_3 \ldots C_i \ldots \) where

\[
\begin{align*}
C_0 &= a_0 \\
C_1 &= a_1 + b_1C_0 \\
C_2 &= a_2 + b_2C_0 + b_1C_1 \\
C_3 &= a_3 + b_3C_0 + b_2C_1 + b_1C_2 \\
C_i &= a_i + \sum_{j=1}^{i} b_jC_{i-j}, \quad i \leq n \\
C_k &= \sum_{j=1}^{n} b_jC_{k-j}, \quad k > n
\end{align*}
\]

Note that the symbols \( C_k, k > n \), will define a periodic steady state response.

The impulse response of a linear circuit can be obtained directly from the transfer function, as illustrated in the following example.

**Example 3.20** Let the transfer function of a circuit be

\[ T = \frac{D^2 + 1}{D^1 + D^2 + 1}. \]

Therefore

\[
(D^2 + 1)x = (D^1 + D^2 + 1)z
\]

\[
z = x + D^2x + D^3z + D^2z.
\]

Assuming all previous values of \( x \) and \( z \) to be 0, the impulse response is given by the sum of individual terms in the sequences, \( x, D^ix, D^2z, \) and \( D^3z \), where \( D^iz \) is 0 during the first two inputs and \( D^2z \) is 0 during the first three inputs.

\[
\begin{align*}
x &= 1000000000000000 \\
D^ix &= 0010000000000000 \\
D^2z &= 0010010111001010 \\
D^3z &= 0010010111001011 \\
z &= 1001011100101111 \\
\end{align*}
\]

The input sequence \( x \) is as defined on line 1, and \( D^ix \) is derived from \( x \). The first two outputs can now be derived as the sum of \( x + D^2x \) + \( D^3z \) + \( D^2z \). The first output can then be used to derive the third value of \( D^2z \) and the fourth value of \( D^3z \). This process is repeated to generate the impulse response which becomes cyclic with repeated sequence 1110010 of period 7.

**Theorem 3.2** The response of a linear sequential circuit to an arbitrary input sequence \( a_0a_1a_2 \ldots a_n00 \ldots 0 \) can be expressed in terms of the impulse response \( h \) as

\[ \sum_{i=0}^{n} a_iD^i(h) \]

where \( D^i(h) \) is \( h \) delayed by \( i \) units and \( D^0(h) = h \).

**Proof:** Consider the circuit of Figure 3.46 and assume that all delay outputs are initially 0. This circuit, in response to an input sequence \( a_i \), generates an input sequence to \( C \) equal to \( 00 \ldots 0a_i00 \ldots \). That is it generates an impulse delayed by \( i \), if \( a_i = 1 \). Therefore, if \( x = a_i, z = a_iD^i(h) \). If an input combination \( (a_0, a_1, \ldots, a_n) \) is applied to \( (x_0, x_1, \ldots, x_n) \) the sequence \( a_0a_1 \ldots a_n000 \ldots \) will appear at the input to \( C \). If the circuit \( C \) is linear, the entire circuit is also linear, and the output response to any input combination \( (a_0, a_1, \ldots, a_n) \) applied to it is identical to the output of \( C \) for the input sequence \( a_0a_1 \ldots a_n00000 \ldots 0 \) applied to \( C \). Let \( H(x_0, x_1, \ldots, x_n) \) be the function realized by the
Figure 3.46 Circuit for proof of Theorem 3.20

The impulse response of a circuit with a complex transfer function $T$ can sometimes be expressed in terms of the impulse responses of the subfunctions of $T$.

**Lemma 3.13** If $h_i$ is the impulse response of a circuit with transfer function $T_i$, then a circuit with transfer function $T = \sum_{i=1}^{n} T_i$ has impulse response

$$h = \sum_{i=1}^{n} h_i.$$  

**Proof:** Exercise.

Theorem 3.2 and Lemma 3.13 are useful in deriving a transfer function with a given impulse response.

**Example 3.22** Let the desired impulse response be

$$h = 10010111010110110...$$

Recall that the impulse response $h$ of a circuit with feedback, is equal to $h_s + h_t$ where $h_t$ is the transient component of $h$ and $h_s$ is the periodic steady state component of $h$. For the above case,

$$h_s = 1101101110...$$

and

$$h_t = 0101000...$$

From Lemma 3.13, if $T_i$ has impulse response $h_i$, and $T_s$ has impulse response $h_s$, then $T = T_s + T_i$ has impulse response $h = h_s + h_i$. The impulse response $h_s$ can be generated by a feedback free machine with transfer function $T_s = D^3 + D$. The impulse response $h_s$ consists of the repeated subsequence 110 of period 3. This can be produced by a machine with transfer function $T_s = T_1 \cdot T_2$, where $T_1$ produces a single pulse every three units of time and $T_2$ produces the output
sequence 11000 ... in response to a single input pulse (Figure 3.47). Therefore, \( T_1 = 1/(D^3 + 1) \) and \( T_2 = D + 1 \) and

\[
T = T_1 + T_2 = D^3 + D + \frac{D + 1}{D^3 + 1} = \frac{D^6 + D^4 + D^3 + 1}{D^3 + 1}.
\]

Therefore

\[
T = \frac{D + 1}{D^3 + 1}.
\]

Suppose \( D^3 x = D^2 x = D x = 1 \) initially. Then the first input \( x = D^3 x + D^2 x = 1 + 1 = 0 \). At \( t = 2 \), \( D^3 x = D^2 x = 1 \), \( D x = 0 \) and the second input is also 0. Repeating this procedure, we obtain the sequence shown below:

\[
\begin{array}{cccccccccccc}
D^3 x & D^2 x & D x & x \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1
\end{array}
\]

Note that the sequence will cycle after the initial conditions reappear in the sequence.

(b) Consider a circuit with feedback, whose transfer function is \( T = (D^3 + 1)/(D^3 + D^2 + 1) \). Then,

\[
z = (D^3 + 1)x + D^3 z + D^2 z.
\]

To generate a null sequence, we set \( z = 0 \) and solve the equation

\[
x = D^2 x + D^3 z + D^2 z \quad \text{for} \quad x.
\]

Since \( z = 0 \) throughout the sequence, we must satisfy the condition \( D^3 z = D^2 z = D z = 0 \). Therefore, a null sequence must satisfy \( x = D^2 x \). If the initial conditions are \( D^3 x = D^2 x = D x = 1 \), the null sequence is 11 ... 1.

A polynomial \( P \) of degree \( k \) is irreducible if \( P \) cannot be factored into a product of polynomials of degree less than \( k \). If a transfer function \( T \) is an irreducible polynomial of degree \( k \) and \( T \) is not a factor of \( D^q + 1 \), for \( q < 2^k - 1 \), then \( T \) has a cyclic null sequence of period \( 2^k - 1 \), containing all subsequences of length \( k \), except the all-0 sequence. Such a null sequence is called a maximal length null sequence. Tables of irreducible polynomials which can be used to generate such null sequences can be found in many books on coding theory [16]. Such polynomials are of importance in the application of linear circuits to error correction.

**Application of Linear Circuits to Error Correction**

A simple example of the application of linear circuits to error correction is illustrated in Figure 3.48. Suppose that a message \( M \) consisting of a sequence of bits is encoded by applying \( M \) as an input sequence to a linear circuit with transfer function \( T \). The output of the linear
circuit is a coded message \( C \), which is then transmitted over a noisy channel. The received message is \( C^* = C + N \), where \( N \) represents the noise produced errors. We wish to design an encoder and a decoder such that the transmitted message \( M \) can be uniquely determined from the received signal \( C^* \).

Let the decoder consist of another linear circuit with transfer function \( T^{-1} \), where \( T^{-1} \) is such that \( T \cdot T^{-1} = T^{-1} \cdot T = 1 \). Denoting the output of the decoder by \( M^* \), we have

\[
C = TM \\
C^* = TM + N \\
M^* = T^{-1}C^* = T^{-1}(TM + N) \\
= M + T^{-1}N. \\
M = M^* + T^{-1}N.
\]

Thus, in order to correctly decode the message sent, we need to determine the sequence \( T^{-1}N \) by observing \( M^* \).

We shall only consider the case where the noise causes a single bit to be in error. That is, the sequence \( N \) may contain at most a single 1. Assume that \( M \) consists of a sequence of \( r \) information bits followed by \( s \) 0's (i.e., \( M = m_1m_2...m_s0...0 \)). If \( N = 00...0 \), then \( M^* = M \) and the last \( s \) bits of \( M^* \) will be 0. In order to determine the correct output from \( M^* \), we shall define \( T \) so that the sequence \( N \) (and therefore \( T^{-1}N \)) is uniquely determined from the last \( s \) bits of \( M^* \). It can be shown [16] that if \( T \) is an irreducible polynomial of degree \( k \), then \( T^{-1} \) will have the desired property if \( s = k \) and \( r = 2^k - 1 - k \).

**Example 3.24** \( T = D^3 + D + 1 \) is an irreducible polynomial of degree 3, and \( T^{-1} = 1/(D^3 + D + 1) \). For single error correction, the total number of bits in the message \( M \) is \( 2^3 - 1 = 7 \). The message \( M \) will consist of four information bits followed by three 0's, i.e., \( M = m_1m_2m_3m_4000 \). The sequence \( T^{-1}N \) for an error in any bit position \( i \) can be determined from the impulse response of the function \( T^{-1} \).

If \( z \) is the impulse response of the function, \( T^{-1}N \) for an error in the \( i \)th position is given by \( D^{i-1}(z) \) where the leftmost bit corresponds to \( i = 1 \). The impulse response of \( T^{-1} \) is defined by \( z = x + D^1z \).

\[
x = 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
\]
\[
D^1z = 0 0 0 1 1 1 0 1 0 0 0 0 ...
\]
\[
Dz = 0 1 1 0 1 0 0 1 1 ...
\]
\[
z = 1 1 1 0 1 0 0 1 1 ...
\]

The impulse response is the cyclic output sequence \( z = 111010 01111... \). The response to a noise pulse occurring at \( t = i, \ 1 \leq i \leq 7 \) is shown in the table of Figure 3.49. Note that the bits 5 to 7 of \( T^{-1}N \) are distinct for each of these cases. Therefore, these 3 bits uniquely determine \( T^{-1}N \), which can be realized by a 3-input multiple output combinational circuit. The outputs of this circuit can be added modulo 2 to \( M^* \) to obtain \( M \). For example, if \( M^* = 1110101 \), the last 3 bits indicate (from Figure 3.49) that \( T^{-1}N = 0011101 \) and

\[
M = M^* + T^{-1}N = 1111000.
\]
Note that a single error in the output of the encoder may cause several bits in \( M^* \) to be different from \( M \), but they are corrected by this method.

3.4 REGULAR EXPRESSIONS

The problem of determining the function realized by a state table, and of specifying a state table corresponding to a word description of a function can sometimes be handled effectively using regular expressions. A regular expression is a representation of a set of sequences. For a sequential machine \( M \) with a single binary output \( z \), an input sequence is said to be \textit{accepted} by \( M \) in a designated initial state if the machine produces an output \( z = 1 \) for the last input of the sequence. An input sequence that is accepted by \( M \) with initial state \( q_0 \) is called a \textit{signal sequence} of \( M \) for the initial state \( q_0 \). The set of all signal sequences of \( M \) is called its \textit{signal set}. A machine with more than one binary output will have a signal set associated with each output. For the state table \( M_1 \) of Figure 3.50(a) with initial state 1, the signal set \( S = \{1,01\} \). The signal set of the table of Figure 3.50(b) in initial state 1 consists of the infinite set of sequences of alternating 0's and 1's beginning with a 0 and ending with a 1.

\[
\begin{array}{c|c|c|c|c|c}
1 & 2,0 & 3,1 & 1 & 2,0 & 3,0 \\
2 & 3,0 & 3,1 & 2 & 3,0 & 3,1 \\
3 & 3,0 & 3,0 & 3 & 3,0 & 3,0 \\
\end{array}
\]

\( M_1 \) \hspace{1cm} \( M_2 \)

(a) \hspace{1cm} (b)

Figure 3.50 Two state tables

It is possible that the signal set \( S \) of a machine contains no sequences; that is, there is no input sequence which, when applied to the machine in the specified initial state, produces a 1-output. For example, the signal set of the state table of Figure 3.51 with initial state 2 is empty and is denoted by \( S_2 = \emptyset \).

If the machine of Figure 3.51 is initially in state 1, the output will be 1 without applying any input. This can be considered as an input sequence of length 0, denoted by \( \lambda \). For the machine of Figure 3.51 in the initial state 1, the signal set is \( S_1 = \{\lambda\} \). It is important to note the difference between the empty signal set \( \emptyset \) and the sequence of length 0. The empty signal set does not contain any sequence, not even \( \lambda \).

The set of sequences accepted by finite state sequential machines can be represented by regular expressions. The set of sequences represented by a regular expression is called a \textit{regular set}. A regular expression for an input alphabet \( I \) is defined recursively by the following rules:

1. \( i \in I \), then \( i \) is a regular expression and \( R = i \) represents the input sequence \( i \).
2. If \( A \) and \( B \) are regular expressions, then \( A \cup B \) (union) is a regular expression which represents all input sequences represented by \( A \) or \( B \).
3. If \( A \) and \( B \) are regular expressions, then \( A \cdot B = \{xy \mid x \in A, \ y \in B \} \) is a regular expression which represents the set of sequences formed by \textit{concatenation} of each input sequence \( x \) in \( A \) followed by each input sequence \( y \) in \( B \). For convenience \( A \cdot B \) will be represented as \( (A) \cdot (B) \) (or simply \( AB \) if \( A \) and \( B \) are single symbols).
4. If \( A \) is a regular expression, then \( A^* = \lambda \cup A \cup A \cdot A \cup A \cdot A \cdot A \cup \ldots \) is a regular expression.

Example 3.25 For the input alphabet \( \{0,1\} \), \( R = 0^*1[0 \cup l(0)^*1]^* \) is a regular expression. The input sequence 0 0 1 0 1 1 is represented by \( R \) as are 10 and 1. However, the input sequence 11 is not represented by \( R \). The regular expression \( 0(0 \cup 1)^* \) represents all input sequences including \( \lambda \). The regular expression \( (0 \cup 1)(0 \cup 1)^* \) represents all input sequences excluding \( \lambda \).

A regular expression can be used as an intermediate step in the derivation of a state table from a word description of a problem. For instance, consider the derivation of a state table which generates a 1 output for any input sequence which contains an odd number of ones.
Index

Absorption, 6
Accept, 176
Acyclic circuits, 10
Adjacency graph, 211
Admissible clusters, 458
Admissible sets, 442
Akers’ array, 412
AND-OR realization, 15
Arrays
  cellular, 408–428
  collector, 423
  half adder, 410
  iterative, 408, 410–411, 556–557
  minterm, 423
  one-dimensional, 408
  programmable, 422–428
  two-dimensional, 410–422
Associative memories, 426
Associativity, 6, 573
Asynchronous sequential circuits
  flow table see Flow table
  flow table reduction, 201–208 (see also Flow table reduction)
  flow table specification, 200–201
  multiple input changes, 260–265
  pulse input, 199–200
  restricted delay assumptions, 265–270
  simultaneous input changes, 260–263
  state assignment, 209–237 (see also Asynchronous state assignment)
  unbounded gate delays, 271–275
  unrestricted input changes, 260, 263–265
  Y-matrix, 237–240
Asynchronous state assignment, 209–237
  connected row set, 212–219
  Hamming code, 222
  MOC tables 236–237
  one-shot, 222
  shared row 219–222
  single transition time (STT), 222–236 (see also STT assignment)
  universal, 216, 221, 234
Autosynchronous code, 272
Auxiliary inputs, 381
B-compatible, 207
Backplane, 472
Backplane routing, 438
Basis, 112
  strong, 112
  weak, 112
Bigraph, 441
Bilateral arrays, 424
Bipartite graph, 441
Bistables 37
Block, 287, 572
Boards, 472
Boolean algebra, 4–9
  application to switching theory, 7–9
  basic postulates, 5
  properties, 5–7
Boolean expression, 7
Boolean function, 7
Branching, 25
Bubble conserving logic, 557–560
Bubble creation, 544
Bubble destruction, 537, 544
Bubble detection, 537
Bubble generation, 536–537
Bubble logic, 537–539
Bubble operations, 540, 544
  creation, 544
  destruction, 544
  splitting, 544
  transfer, 540
Bubble propagation, 534–536
  conductor loops, 534
  T-bars, 534–535
Bubble splitting, 544
Bubble transfer, 540
Cards, 472
  assignment, 437
  placement 438
  routing, 438
Cellular arrays, 408–428
Chain, 494
Channel routing, 521
Channels, 521
Checking experiments, 148–149 (see also Experiments)
Chevron-shaped overlays, 558
Clocks, 40–43
Closed cover, 127
Clusters, 457
  admissible, 458
Collector arrays, 423
Column dominance 24
Combinational circuits, 9–32
  analysis of, 10–12
  decomposition, 79–94 (see also Decomposition)
  modular design, 31
  multiple level, 77–79
  multiple output, 65–77
synthesis of, 12-32
  tree type, 78
Combinational functions, 9
  decomposition, 79-94 (see also Decomposition)
  special classes of, 94-107
  symmetric, 95-101 (see also Symmetric functions)
  threshold, 103-107 (see also Threshold functions)
  unate, 101-103 (see also Unate functions)
Commutativity, 6, 573
Compatible sets, 122 (see also Compatibles)
  generation of, 122-127
  selection of, 127-141
Compatibles
  dominance, 130
  generation of, 122-127
  maximal, 126
  output, 122
  prime, 130
Complete sets of logic primitives, 107-113
Completeness, logical, 107
  strong, 107
  weak, 108
Completion detector, 271
Completion signals, 271
Concatenation, 177
Conductor loops, 534
Contact networks, 8, 9
Content addressable memories, 426
Counter, 51, 364-370
  controlled, 53
  Mayne, 367
  Ware, 366
Critical race, 211
Cut-points, 423
Cyclic table, 25

d-trio, 252
Data locations, 547
Decomposition
  asynchronous sequential circuits, 355-360
    combinational circuits, 79-94
  complex disjunctive, 89-92
  disjunctive, 80
  iterative disjunctive, 89
  multiple disjunctive, 89
  multiple transition time, of asynchronous sequential circuits, 357
  nondisjunctive, 92-94
  parallel, 308
  synchronous sequential circuits, 308-315
    serial, 308
  simple disjunctive, 80-89
  single transition time, of asynchronous sequential circuits, 357
  universal canonical, of sequential machines, 345-350
Decomposition map, 81
Definite events, 322
Definite machines, 322
Delay, 40-43
  inertial, 241
  inserted, 240
  pure, 241
  stray, 240
Delay element, 39 (see also Flip-flop)
Delay operator, 162
Delays and hazards, 240-260
DeMorgan’s law, 6
Dichotomy, 229
  compatible dichotomies, 231
  ordered, 231
Distinguishing sequences, 142-144
Distributivity, 6, 573
Dominance
  admissible sets, 442
  column, 24
  compatibles, 130
  row, 24
Double rank sequential circuits, 364-370
Double-rail inputs, 15
Duality, 5, 7

Easy path, 560
Element replication, 447
Equivalence, 573
Essentially equivalent, 207
Excitation matrix, 46
Experiments
  adaptive, 142
  checking, 148-149
  distinguishing, 142
  homing, 144
  preset, 142
  sequential machines, 142-149

Factorization, 79
Fanin index, 78
Fanout index, 65
Feasible cover, 441
Feedback-free circuits, 321
Feedback function, 327
  output as, 333-335
Feedback index, 324
Feedback loops, 10
Feedback shift register (FSR) realizations, 335-343
Finite state machine, 33
Flip-flop (FF), 37
  clocked, 42
D, 39
JK, 38
master-slave, 42
set-reset (SR), 38
T, 39
Flow table, 196
  multiple output change (MOC), 178
  nonfundamental mode, 199
  normal mode, 178
  normalization, 203
  single output change (SOC), 197
Flow table reduction, 201-208
  MOC, 204-208
  SOC, 202-204
  time-dependent, 204-206
  time-independent, 206-208
Fragment of graph, 497
Frame, 472
Functions, 573
  associative, 573
  commutative, 573
  distributive, 573
  domain, 573
  identity, 573
  inverse, 573
  mapping, 573
  range, 573
Graphical models of circuits, 438-441
Graphs, 572
  acyclic, 572
  connectivity matrix, 572
directed, 572
edges (arcs), 572
in-degree, 572
out-degree, 572
path, 572
undirected, 572
vertices (nodes), 572
Gray code, 366
Half-adder arrays, 410
Hamming code, 222
Hard path, 560
Hazard, 240-260
  combinational, 242-248
dynamic, 242
dynamic function, 246
essential, 250
function, 245
logic, 245
output, 248
sequential, 248-252
static, 242
static function, 246
steady state, 248
transient, 248
0-hazard, 242
1-hazard, 242
Hazard detection, 253-258
ternary algebra, 253-256
8-valued algebra, 256-258
Heuristics, 438
Homing sequences, 144-147
Hypergraph, 439
I-partition, 287
IC placement, 438
IC selection, 437
Idempotence, 6
Implicant, 15
Implication graph
  IL, 156
  of compatibles, 131
  of state table, 327
  universal, 327
Impulse response, 167
Incompatible set, 128
Incompatibles, maximal, 128
Information losslessness, 150
degree of, 150
finite (ILF), 150, 156-158
general (GIL), 150, 153-155
IL-I, 150
IL-II, 150-153
Initiate nets, 457
Input alphabet, 32
Input state, 196
Input symbol, 32
Instruction, 540
Internal states, 33
Inverse, 166
Irreducible polynomial, 173
Iterative arrays, 408, 410-411, 556-557
k-k functions, 558
Karnaugh map, 18
Kernighan-Lin algorithm, 453
Lattice, see Relations
Lee's algorithm, 511
Linear sequential circuits, 158-176
  application to error correction, 173-176
  null sequences, 172-173 (see also Null sequence)
Linearity, 108
Linearly separable functions, 105
Lin's algorithm, 226
Logically complete, 107
Lower bound, 573

m/n code, 274
Machine identification, see Experiments
Magnetic bubbles
  applications, 567
  bubble conserving logic, 557-560
  computation speedup, 534
  instruction minimization, 550
  iterative array realizations, 556-557
  logical operations, 537-539
  mathematical model, 540-546
  optimization problems, 546-555
  physical background, 533-539
  propagation, 534-536
  - resident bubble cells, 560-567 (see also Resident bubble cells)
  - time and space considerations, 547-548
Magnetic bubbles, generation, destruction and detection, 536-537
Maitra cascades, 408
Manhattan geometry, 484
Mathematical programming, 573
  - integer linear programming, 574
  - linear programming, 574
  - objective function, 574
Maximal compatibles, see Compatibles
Maxterm, 13
Maxterms, product of, 14
Mayne counter, 367
Mealy machine, 33
Memory elements, 37-40
Memory layouts, 547
  - universal, 548
Microprogramming, 428
Minimal product-of-sums, 29
Minimal spanning trees, 495-501
Minimal sum-of-products, 29
Minimum feedback realizations with SR flip-flops, 343-345
Minterm, 12
Minterm arrays, 423
Minterms, sum of 13
Mm pairs, 299
  - generation of, 302
Modular realizations
  - fixed fanout module, 407
  - three-input modules, 393
  - tree structures, 400-408
  - with fixed number of inputs, 392-400
Monotonic cost functions, 517
Moore machine 34
Multilayer routing, 519

Multiple output prime implicants, 66 (see also Prime implicant)
  - essential, 73
  - generation of 68-72
  - good, 75
  - selection, 72-77
Multirail cascades, 410
Munkres algorithm, 478
MOS (metal oxide semiconductor) circuits, 9

N-cube, 12
NAND-AND realization, 31
NAND-NAND realization 30
Nearest neighbor, 497
Nets, 439
  - multiterminal (multielement), 439
  - signal, 439
Non-decreasing overlap (NDO) theorem, 542
Noncritical race, 211
NOR-AND factoring, 268
NOR-NOR realization, 31
NOR-OR realization, 31
Null sequence, 172
  - maximal length 173

One-dimensional arrays, 408-410
One-hot code, 260
One-preservation, 108
Output alphabet, 32
Output symbol, 32

Pair chart, 123
Parity tree, 78
Partial dichotomy, see Dichotomy
Partial ordering, 572
Partition pair, 289
Partitioning, 437, 441-472
  - as covering problem, 441-445
  - as mathematical programming problem, 445-447
  - constructive heuristic procedures, 456-464
  - iterative procedure, 450-456 (see also Kernighan-Lin algorithm)
  - minimum delay, 464-471
  - minimum delay, for tree graphs, 465-467
  - minimum delay, with pin constraints, 471, 472
  - near-optimal, to minimize interconnections, 449-464
  - optimal, to minimize interconnections, 441-449
  - parallel procedure, 459
  - sequential procedure, 459
  - with element replication, 447-449
Partitions, 287
  - lattice of, 288
  - product of, 288
sum of, 293
Permutation machine, 345
Permutation-reset (PR) machine, 346
Pin assignment, 438
Placement problem 472-493
branch and bound procedure, 481
constructive methods, 489-493
iterative procedures, 483-489
lower bounds 475-481
semienumerative procedure, 481-483
Positive unateness, 108
Post's theorem, 111
Primary inputs, 381
Prime implicant, 15
covering problem, 22
essential, 23
good, 28
multiple output, 66 (see also Multiple output prime implicant)
Prime implicant table, 22
cyclic, 25
Product of maxterms, 14
Program, 540
replicating, 543
Programmable arrays, 422-428
Proper set systems, 302
Quine-McCluskey procedure, 17
Race, 211
Read-only-memory (ROM) arrays, 425
Ready signals, 271
Rectilinear distance, 484
Reduced dependence, 287
in asynchronous sequential circuits, 350-355
using JK flip-flops, 317
using SR flip-flops, 315, 318-319
using T flip-flops, 316-321
Reduced feedback, realizations with, 321-345
with SR flip-flops, 343
Reed-Muller expansion, 110
Regular expressions, 176-184
derivation from state table, 181-183
state table derivation from, 178-180
Regular set 177
Relations, 572
binary, 572
equivalence, 573
lattice, 573
partial ordering, 573
reflexive, 572
symmetric, 572
transitive, 572
Relevant partition, 351
Replicating program, 543
Reset machine, 345
Reset-identity machine, 347
Resident bubble cells, 360-367
crossover, 362
fanout/transmission, 561
NAND, 562
SR flip-flop, 566
use, 563
Routing, 493-521
channel, 521
multilayer, 519
selection of net connections, 494-511
specification of paths, 511-521
Row dominance, 24
Row set, 212
connected, 212
intermeshed, 212
Selectability, 383
Self-duality, 108
Sequences
checking, see Checking experiments
distinguishing, 142
homing, 144
synchronous, 146-147
Sequential circuits, 32-57
analysis, 54-55
asynchronous, see Asynchronous sequential circuits
decomposition, 308-315 (see also Decomposition)
double rank, 364-370
linear, see Linear sequential circuits
linear, 158
reduced dependence, see Reduced dependence synthesis, 43-53
Sequential function, 32
Sequential machines, 32-37
decomposition, 308-315 (see also Decomposition)
experiments on, 142-149
identification, 148-149
information lossless, 150-158 (see also Information losslessness)
iterative realizations, 418-422
modular realizations, 392-408 (see also Modular realizations)
special classes, 149-176
strongly connected, 148
Set pair, 292
Set systems, 292
information of, 296
proper, 302
sum of
universal, 345
Sets, 571
cardinality, 571
Cartesian product, 571
difference, 571
disjoint, 571
finite, 571
intersection, 571
null, 571
union, 571
Shannon's expansion theorem, 85, 98
Shift register, 51
Shift register realizations, 335-343, 360-364
asynchronous, 360-364
unitary feedback, 335-343
Signal sequence, 176
Signal set, 176
Single feedback loop, realizations with, 324
Single transition time (STT) state assignment, 222-236
bound, 234, 235
Hamming code, 222-225
Liu's, 225-228
multicode, 235-236
Tracey's, 228-233
unicode, 225-235
Sink trees, 465
Source trees, 467
Spacer, 260, 272
Spanning tree, 494
Speed independent circuits, 271
State assignment, 45
asynchronous, 209-237 (see also Asynchronous state assignment)
incompletely specified unicode, 292
multicode, 49
unicode, 45
State diagram, 33
State tables, 33
covering, 121
reduction of, 120-141
State transition graph, 33
State variable, 45
States, 33
covering, 121
implication, 123
input, 196
internal, 33
total
Steiner points, 501
Steiner tree, 494, 501-511
Steiner's problem, 501
Strong complete, 107
Strongly connected, 148
Subset, 571
proper, 571
Substitution property, 289
Substitution, principle of, 5
Superposition, 158
Symmetric functions, 95-101
canonical realization, 101
total symmetry, 96
Symmetry, see Symmetric functions
Synchronizing sequences, 146-147
T-bar propagation, 535
Test graph, 333
Threshold element, 104
Threshold functions, 103-107
Total state, 196
Tracks, 521
Transfer function, 162
Transit locations, 547
Transition subcube, 225
Transition time, 213
Trap state, 178
Tree
branches, 572
homing, 146
leaves, 572
minimal spanning, 495-501
Steiner, 494, 501-511
successor, 147
Tree type circuit, 78
Trivial partition, 287
Truth table, 10
Turing machine, 56
Two-dimensional arrays, 410-418
Two-level realization, 14
Two-phase clocked circuits, 370
Two-rail cascades, 409
Unate functions, 101-103
negative, 103
positive, 101
Unconnected pair, 486
Unconnected set, 486
maximal, 486
Universal logic modules 379-392
bounds on number of terminals, 389-392
with double rail inputs, 382-387
with interconnections between terminals, 387-389
with one double-rail input, 381, 382
with single rail inputs, 380
Upper bound, 573
Universal state assignment 216
Vias, 519
Ware counter, 366
Weak complete, 108

Y, z matrix, 46
0-partition, 287