STAN-CS-70-173 SU-SEL-70-058

The Mutual Exclusion Problem

by

T. H. Bredt

August 1970

Technical Report No. 9

This work was supported in part by the Joint Services Electronic Programs U.S. Army, U.S. Navy, and U.S. Air Force under Contract N-00014-67-A-0112-0044 and by the National Aeronautics and Space Administration under Grant 05-020-377.

DIGITAL SYSTEMS LABORATORY

STANFORD ELECTRONICS LABORATORIES

STANFORD UNIVERSITY • STANFORD, CALIFORNIA

STAN-CS-70-173 SEL-70-058

THE MUTUAL EXCLUSION PROBLEM

by

T. H. Bredt

August 1970

Technical Report No. 9

DIGITAL SYSTEMS LABORATORY

Stanford Electronics Laboratories Computer Science Department

Stanford University

Stanford, California

This work was supported in part by the Joint Services Electronic Programs U.S. Army, U.S. Navy, and U.S. Air Force under Contract N-00014-67-A-0112-0044 and by the National Aeronautics and Space Administration under Grant 05-020-337.

STANFORD UNIVERSITY
Digital Systems Laboratory
Stanford Electronics Laboratories Computer Science Department

Technical Report Number 9
August, 1970

THE MUTUAL EXCLUSION PROBLEM

by

T. H. Bredt

ABSTRACT

This paper discusses how n components, which may be programs or circuits, in a computer system can be controlled so that (1) at most one component may perform a designated "critical" operation at any instant and (2) if one component wants to perform its critical operation, it is eventually allowed to do so. This control problem is known as the mutual exclusion or interlock problem. A summary of the flow table model* for computer systems is given. In this model, a control algorithm is represented by a flow table. The number of internal states in the control flow table is used as a measure of the complexity of control algorithms. A lower bound of n + 1 internal states is shown to be necessary if the mutual exclusion problem is to be solved. Procedures to generate control flow tables for the mutual exclusion

* Bredt, T.H. and McCluskey, E.J. A model for parallel computer systems. Technical Report No. 5, SEL Digital Systems Laboratory, Stanford University, Stanford, California (April 1970).

problem which require the minimum number of internal states are described and it is proved that these procedures give correct control solutions. Other so-called "unbiased" algorithms are described which require **2.n!** internal states but break ties in the case of multiple requests in favor of the component that least recently executed its critical operation. The paper concludes with a discussion of the tradeoffs between central and distributed control algorithms.

TABLEOFCONTENTS

LIST OF TABLES

LIST OF **FIGURES**

INTRODUCTION

In recent years, there has been a great deal of interest in the mutual exclusion or interlock problem in multiprocessor and multiprogrammed systems [4,5,6,7,8,9,10,15,16]. This problem occurs in an environment where several system components (programs or circuits) are operated concurrently. The components are assumed to contain critical operations or instructions (critical sections), whose simultaneous execution must be avoided. Typically, critical sections represent references to a common memory location or possibly the modification of a common system table. A more detailed statement of the mutual exclusion problem is given below.

The Mutual Exclusion Problem:*

Given two or more components in a parallel computer system, which are operated concurrently and contain critical sections, control these components so that the following two restrictions are always satisfied.

Restriction 1:

A most one component is in a critical section at any instant.

Restriction 2:

If a component desires to enter a critical section, it is eventually allowed to do so.

---

* Dijkstra [8,9,10] has solved a slightly different version of this problem. He did not require that a given program must enter its critical section but rather that the decision as to which program enter its critical section next not be postponed indefinitely.

For components which are programs, this problem is often solved by defining special hardware instructions and assuming the exclusive execution of these instructions [7,9,10,15,16]. In [4] a model for parallel computer systems was proposed in which fundamental-mode flow tables are used to describe the operation of each component. The purpose of this model is to study control problems such as the mutual exclusion problem. In [5], we have shown how flow tables can be used for the analysis and synthesis of sequential programs. The application of flow tables in the design of sequential circuits is well known [24]. As a result, flow tables and the flow table model provide a basis for the study of both hardware and software solutions to the mutual exclusion problem. A flow table solution for the mutual exclusion problem in the case where two components are controlled is given in [4]. This solution was shown to be correct in [6] using a general analysis method based on the construction of a directed graph representing the state transitions undergone by the entire system. This type of analysis is not feasible as the number of system components becomes large. The notion of correct operation we use is the following. Given a problem, such as the mutual exclusion problem, which is stated in terms of restrictions on system operation, we say that a parallel system is correct with respect to the given problem if the problem restrictions are always satisfied during system operation.

In this paper, the solution of the mutual exclusion problem when arbitrary numbers of processes must be controlled is considered. Several different types of control structures are discussed and

various solutions or control algorithms given and proved to be correct. These control algorithms are optimal in the sense that they require the minimum number of internal states in a fundamental-mode flow table.

In the next section, a short summary of the flow table model is given. This is followed by the specification of a parallel system for the mutual exclusion problem and the characterization of different control mechanisms.

THE FLOW TABLE MODEL FOR PARALLEL SYSTEMS

A detailed description of this model is given elsewhere [4,6]. The essential features are described here. The definition of a parallel system in the flow table model is given below.

Definition 1:

A parallel system is a finite collection of components

$$\mathcal{C} = \{C_1, C_2, \ldots, C_N\}$$

and a finite collection of lines

$$\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_M\}$$

Each component C_i has a set of input variables called the input set

$$Ii = \{x_{i_1}, x_{i_2}, \ldots, x_{i_m}\}$$

$$1 \leq i_j \leq M, \; j = 1, \ldots, m$$

and a set of output variables called the output set

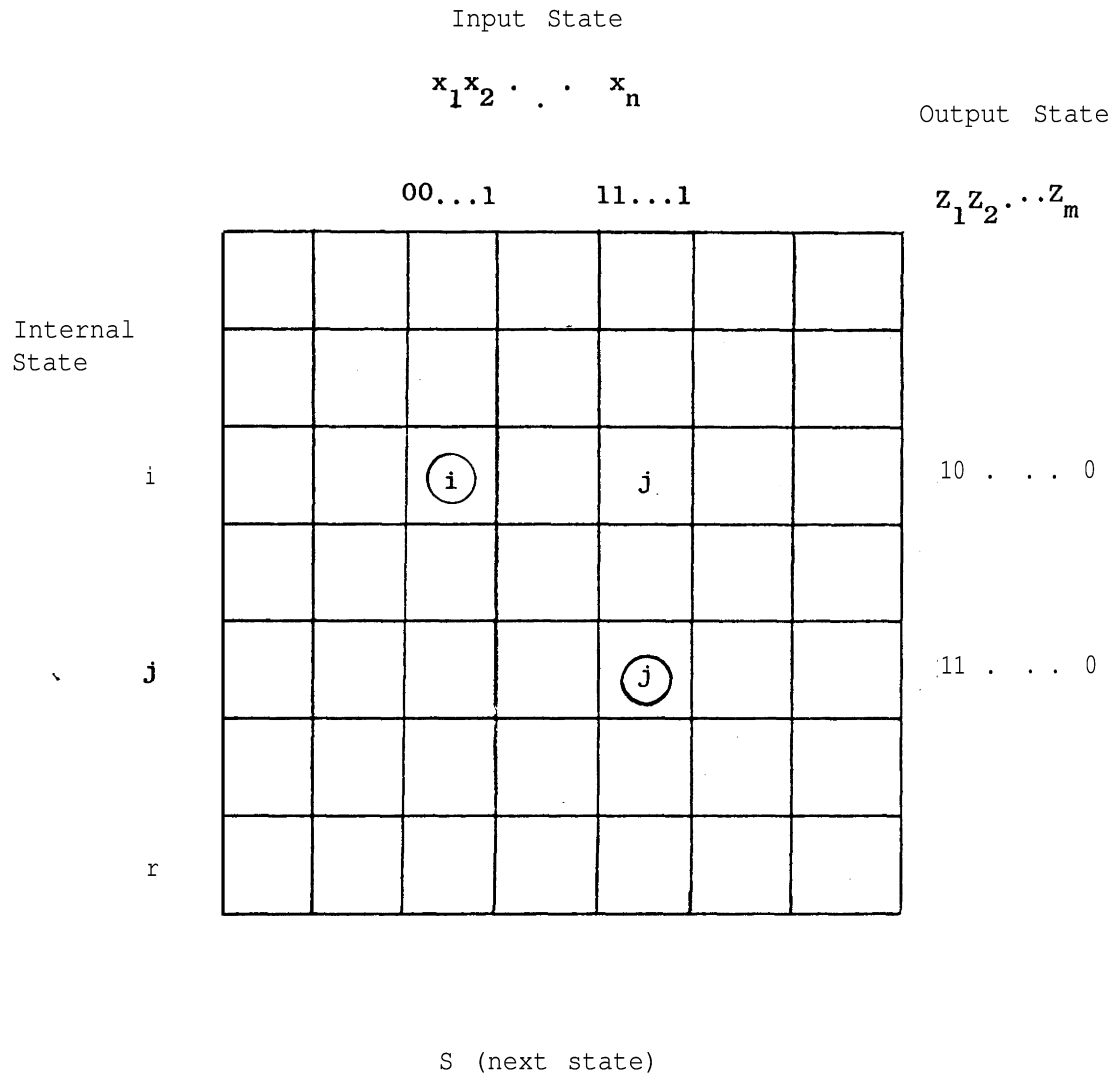
$$O_i = \{X_{i_1}, X_{i_2}, \ldots X_{i_m}\}$$

$$1 \leq ij \leq M, \quad j = 1,\ldots,m$$

Each line $\ell_j = (X_j, x_j)$ connects a component output variable X_j with a component input variable x_j. The lines carry level values and value changes propagate from component output to component input. Each output variable must be connected by a line to exactly one input variable and each input variable must be connected by a line to exactly one output variable. The operation of each component is described by a completely specified fundamental-mode flow table with a designated initial internal state. The initial value for each line is the value specified for the output variable associated with the line.

The general form of a fundamental-mode flow table is shown in Table 1. Each row in the flow table represents an internal state of the component whose operation the flow table describes. The present values of the component input and output variables define the component input state and output state, respectively. The total state of a component is defined by its present internal state and input state. The total system state or system state is defined by the N-tuple consisting of the present total state of each of the components. The parallel system designed in [4] to solve the mutual exclusion problem for two programs is shown in Fig. 1. The initial system state for this parallel system is written (1-0,1-0,1-00). The interpretation of the line values is given in the next section. The initial total state of component C_3

Table 1. General Form of a Flow Table



$\mathcal{C} = \{C_1, C_2, C_3\}$

$\mathfrak{L} = \{\ell_1, \ell_2, \ell_3, \ell_4\}$

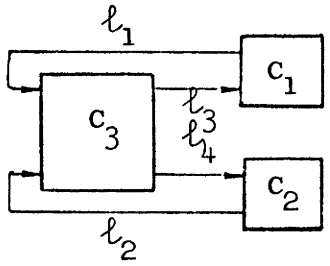
$\ell_1 = (x_1, x_1)$, $\ell_2 = (x_2, x_2)$

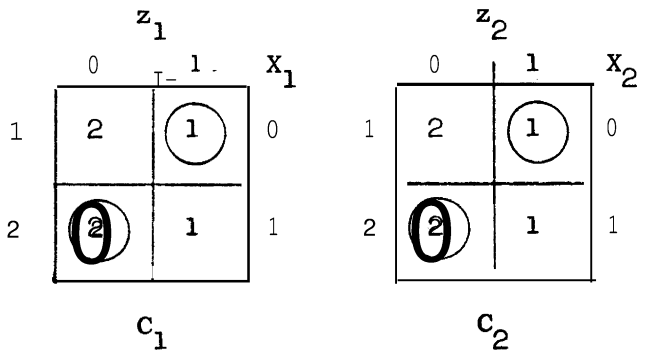
$\ell_3 = (z_1, z_1)$, $\ell_4 = (z_2, z_2)$

$O_1 = \{X_1\}$, $I_1 = \{z_1\}$

$O_2 = \{X_{2t}$, $I_2 = \{z_{2t}$

$O_3 = \{z_1, z_2\}$, $I_3 = \{x_1, x_2\}$





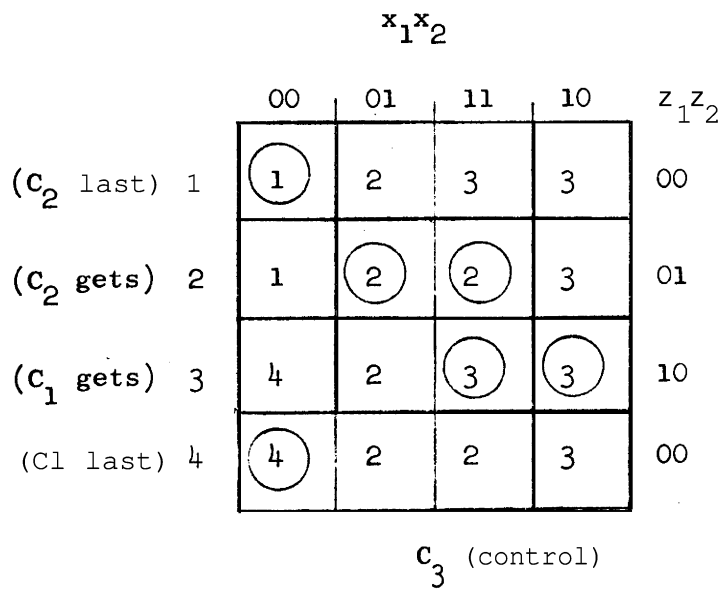


Figure 1. Parallel system for the two-component mutual exclusion problem,

in Fig. 1 is written 1-00. If the present internal state is the same as the next-state entry determined by the component total state, the component is said to be stable; otherwise, it is unstable. For a flow table, we require that each unstable entry specifies a stable entry, a table which does not satisfy this condition is called a state table.

The assumptions about delays in a parallel system are as follows

Assumption 1:

The time for a value change to propagate from a component output to a component input-(line delay) is finite and unbounded.

Assumption 2:

Within a component, delays are finite and bounded.

Line delays need not represent "pure" delays and each component is assumed to have no knowledge of the duration of delays in any other component.

The use of flow tables rather than functions to describe component operation distinguishes this model from others [1,2,3,12,13,14,17,18, 19,20,21,22,23,25,26,27]. Flow tables provide a direct, formal correspondence between the model and the implementation of the model whether the implementation is a program or a circuit. Our model resembles the model of Muller [25] for speed independent circuits. He restricts components to have single outputs and assumes line delays are zero and component delays are unbounded. In other models components communicate by sharing memory cells rather than by wired connections. These models also assume that line delays are zero. The consideration

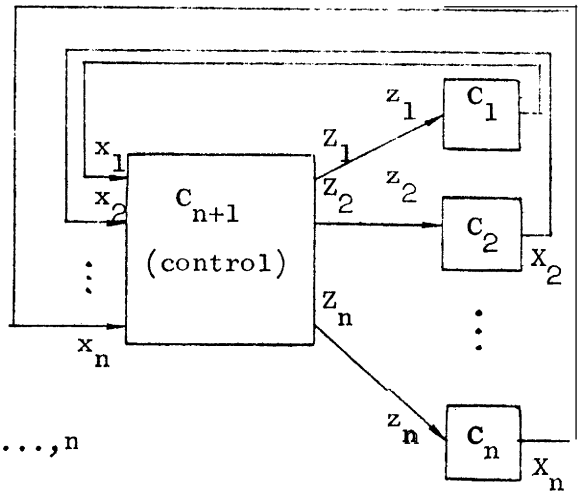
of line delays is particularly important in the mutual exclusion problem as is the possibility that two or more components may make simultaneous requests. That is, multiple-input changes may occur at a component. In [4], a mode of operation is defined such that each component input change results in a unique internal-state transition. This mode of operation proceeds in two phases which can be described as follows. When a component enters a stable total state, it determines the present input state by recording in a rank of flip-flops the current input state. This is done using an internal clock signal. The present input state determines whether the component remains stable or undergoes an unstable transition to a new stable internal state. During unstable transitions, computations such as the execution of a critical section may be carried out. This response to the present input state is the second phase of component operation. During this second phase, all input transitions are isolated from the component by the input rank of flip-flops. This two-phase operation defines the basic cycle of operation for a component. We say that a component has recognized an input-variable transition, if the new input value is recorded in an input flip-flop. Because of our line delay assumption, it can be guaranteed that when a component produces an output variable transition, the new value propagates to the input at the other end of the line and is recognized if and only if either the component never changes the output value again or before the output value is changed, the component must recognize an input change produced in recognition of the propagation of its output value to a component input. A further discussion of these considerations is given in [6].

A PARALLEL SYSTEM FOR THE MUTUAL EXCLUSION PROBLEM

In the n-process or n-component mutual exclusion problem, there are n components which contain critical sections. Each component is assumed to enter, leave and then re-enter its critical section in an infinite loop. We first consider solutions to the mutual exclusion problem with a single control component or control mechanism. The complete specification of a parallel system, with the exception of the control flow table, for the n-process mutual exclusion problem is shown in Fig. 2. To clarify the description of the operation of the system, we have deviated slightly from the form specified for a parallel system in Definition 1 in labelling the lines. The interpretation of the variable values for this parallel system is given in Table 2. The operation of each component C_i, $1 \leq i \leq n$, is as follows. Unless specifically stated otherwise, the component is initially in total state 1-0 and is unstable. In this state, the component does not want to enter its critical section and is not in its critical section. Eventually, the component enters total state 2-0 where X_i is set to 1. The component now wants to enter its critical section and will remain in this total state until it recognizes the enabling value $Z_i = 1$. In total state 2-1 the component enters and leaves its critical section (exactly once). After it has left, the component enters total state 1-1 where X_i is set to 0. This value propagates to the control component which presumably acknowledges the arrival of the 0 value for x_i by setting Z_i to 0. When z_i becomes 0, the cycle begins again.

$\mathcal{C} = \{C_1, C_2, \ldots, C_{n+1}\}$

$\mathcal{L} = \{\ell_1, \ldots, \ell_{2n}\}$



$\ell_i = (X_i, x_i)$, $i = 1, \ldots, n$

$\ell_i = (Z_i, z_i)$, $i = n+1, \ldots, 2n$

$O_i = \{X_i\}$, $i = 1, \ldots, n$

$I_i = \{z_i\}$, $i = 1, \ldots, n$

$O_{n+1} = \{Z_1, Z_2, \ldots, Z_n\}$

$I_{n+1} = \{x_1, x_2, \ldots, x_n\}$

	Z_i = 0	Z_i = 1	X_i
1	2	(1)	0
2	(2)	1	1

Flow Table for C_i, $1 \leq i \leq n$

Figure 2. Parallel system for the n-component mutual exclusion problem without the control flow table.

Table 2. Interpretation of Variable Values for Fig. 2

$(1 \leq i \leq n)$

$x_i = 0$	C_i is not in its critical section and does not want to enter its critical section.
$x_i = 1$	C_i is in its critical section or C_i wants to enter its critical section.
$Z_i = 0$	C_i may not enter its critical section.
$Z_i = 1$	C_i may enter its critical section.

For two components with critical sections, a control flow table is given in Fig. 1.

In [5], it is shown how to obtain a sequential program from a flow table and how for a certain class of programs, it is always possible to construct a flow table. Program implementations of the flow tables for C1 and a two-process control program are given in Table 3. The assignment statements change values on interconnecting lines. The wait statements are used to test the component input state and to transfer to the appropriate next statement when an input transition occurs. Each pair, e.g. (0,3), specifies an input state and the number of the next statement to be executed if the input state is recognized. There are no restrictions on the exclusiveexecution of any statements in these programs.

CONTROL FLOW TABLES

In this section, the phrase control flow table refers to a flow table for the control component (C_{n+1}) in Fig. 2. A control flow table is said to be correct if the parallel system of Fig. 2, with that flow table for the control component, is correct with respect to the mutual exclusion problem. We say that a component C_i ($1 \leq i \leq n$) is enabled to enter its critical section when Z_i has the value 1.

The following definitions serve to partition the class of correct control flow tables.

Table 3. Sequential Programs for Component C1 and a Two-Component Control Mechanism.

```
     INPUT Z1;
     OUTPUT X1;  (initially X1 = 0)
1:   DUMMY; (computation outside critical section)
2:   WAIT (0,3);
3:   X1 := 1;
4:   WAIT (1,5);
5:   DUMMY; (critical section)
6:   X1 := 0;
7:   GO TO 1.
```

(a) Program for Component C_1

```
     INPUT X1,X2;
     OUTPUT. Z1,Z2; (initially Z1 = Z2 = 0)
1:   Z2:=0;
2:   WAIT (01,4), (11,7), (10,7);
3:   Z1:=0;
4:   Z2:=1;
5:   WAIT (00,1), (10,6);
6:   Z2:=0;
7:   Z1:=1;
8:   WAIT (00,9), (01,3);
9:   Z1:=0;
10:  WAIT (01,4), (11,4), (10,7).
```

(b) Control Program for Two Components

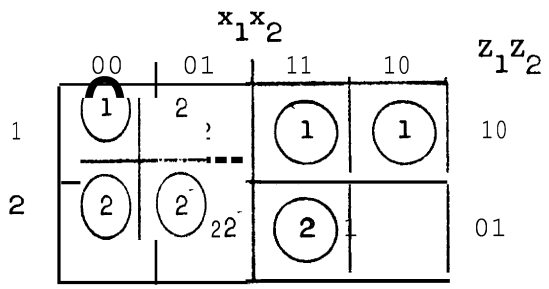
Definition 2:

A correct control flow table is said to be unbiased if components are enabled in the order in which their requests are first recognized and if, when multiple requests are recognized simultaneously for the first time, components are enabled in the reverse of the order determined by their most recent access to their critical sections.

The control flow table in Fig. 1 is unbiased. If a correct control flow table is not unbiased, it is said to be biased.

Restriction 2 of the mutual exclusion problem states that if a component C_i ($1 \leq i \leq n$) wants to enter its critical section, it is eventually allowed to do so. We will consider this restriction to be violated if it is possible for one or more components to halt outside their critical sections (with X_i equal to 0) such that some other component , say C_j, is prevented from entering its critical section when presumably it wants to do so. For example, the control flow table shown in Table 4 will correctly control two components C_1 and C_2 in the parallel system of Fig. 2 as long as both C_1 and C_2 run forever. The initial system state with this control flow table is (1-1, 1-0, 1-00). Component C_1 is assumed to have just left its critical section and will not be permitted to enter again until after C_2 is enabled. If C_2 should halt, C_1 will never be enabled again. Dijkstra [8,9,10] also does not allow control solutions which fail if one or more programs halt.

Table 4. A Possible Control Flow Table for n =2



The following theorems establish necessary conditions for correct control flow tables.

Theorem 1:

If a control flow table is correct, there must be at least one internal state with an output state for which Z_i has the value 1, i = 1,...,n.

Proof:

If there is no output state for which Z_i has the value 1, then z_i, the input to component C_i, will never equal 1. Therefore component C_i, will never enter its critical section violating Restriction 2 of the mutual exclusion problem.

Theorem 2:

The output state for the initial total state of a correct control flow table must have Z_i equal to 0 for all i, i = 1,...,n.

Proof:

Suppose there exists a correct control flow table with Z_i equal to 1 for some i in the initial total state. Initially, lines are assumed to be stable and components C_i are assumed to be in internal state 1. Thus component C_i is in the stable total state 1-1. Component C_i may not request to enter its critical section until z_i becomes 0. The control should be stable initially since all x_i are 0. The control will not

leave its stable initial state until a different input state is recognized. Since x_i cannot become 1, some other component must request access to a critical section before Z_i can be changed to 0. This is not allowed in a correct control flow table.

Theorem 3:

If a control flow table is correct, Z_i is changed from 0 to 1 only if x_i is 1.

Proof: (By contradiction) -

Suppose Z_i is changed from 0 to 1 when x_i is 0. The output variable X_i of component C_i may be 0 or 1. If X_i is 0, component C_i could be trapped in total state 1-1 and either never released or released only after a transition for another control input. Neither case is allowed for a correct control flow table. If X_i is 1, C_i is in internal state 2. As soon as z_i becomes 1, C_i may enter its critical section. It is possible that before the input variable x_i becomes 1, a different input, say x_j, becomes 1. If the control does not set Z_j to 1 until after x_i becomes 1 then the enabling of C_j depends on the operation of C_i which is not allowed. If Z_j is set to 1 before x_i becomes 1, both C_i and C_j may enter their critical sections simultaneously which violates Restriction 1 and the control flow table is not correct.

Theorem 4:

If a control flow table is correct, it must never enter a total state for which the output state has more than one output variable with the value 1.

Proof:

Suppose such a total state is entered and Z_i and Z_j have the value 1. By Theorem 3, x_i and x_j must be equal to 1. It is possible that X_i and X_j are also 1 since X_i and X_j are not set to 0 until the inputs z_i and z_j have the value 1 and line delays cannot be controlled. Therefore C_i and C_j may both be in internal state 2 and may simultaneously enter their critical sections. This violates Restriction 1 and the control flow table cannot be correct.

Theorems 1-4 enable us to prove the following theorem which establishes a lower bound on the number of internal states required for a correct control flow table.

Theorem 5:

For a given n, every correct control flow table must have at least n + 1 internal states.

Proof:

By Theorem 2, the output state for the initial internal state must have Z_i equal to 0 for all i, i = 1,...,n. By Theorems 1 and 4, there must be at least one internal state for which Z_i has the value 1 and Z_j has the value 0, $j \neq i$ for each i, i = 1,...n.

The number of internal states required by a correct control flow table provides a measure of the complexity or cost of the control function or algorithm for the mutual exclusion problem which is independent of whether the algorithm is implemented as a program or as a circuit.

The following theorem establishes another necessary condition for a correct control flow table. This condition is not required to determine the lower bound on the number of internal states but will be used later.

Theorem 6:

If a control flow table is correct, output variable Z_i is changed from 1 to 0 only if x_i is 0.

Proof:

Suppose Z_i is changed from 1 to 0 when x_i is 1. By Theorems 2 and 4 either the output state for the next internal state has no Z_j equal to 1 or exactly one Z_j equal to 1. Suppose no Z_j is equal to 1. Now if some x_k becomes 1, it must be recognized in a finite time (Assumption 2) and the control flow table must enter a stable state for which Z_k is 1. If it does not, the enabling of C_k depends on the activity of some other component, which is forbidden. The enabling of C_k, allows C_k to enter its critical section. During this interval, from the moment the state for which Z_i is 1 is left until C_k enters its critical section, the 1 value produced on Z_i may be propagating to the input of

component C_l. It is possible that C_i will recognize this 1 value and enter its critical section. Both C_i and C_k would be in critical sections simultaneously violating Restriction 1 and the control flow table is incorrect.

Suppose that when the control flow table left the state for which Z_i was 1, it went directly to a state for which Z_j is 1. If x_j is 0, the flow table is incorrect by Theorem 3. If x_j is 1, then an argument similar to one given for component C_k above shows that C_i and C_j can be in critical sections simultaneously and the solution is again incorrect.

Thus in both cases, when we leave an internal state for which Z_i is 1 when x_i is 0, the control flow table is incorrect.

Biased Control Flow Tables

For a given value of n, the following procedure constructs a biased control flow table with the minimum number of internal states.

Procedure B

(Biased Control Flow Table With n + 1 Internal States)

1. Define a flow table with 2^n columns, one column for each possible input state, and n + 1 internal states (rows), which are numbered 0, 1, 2, ...n.
2. Let the initial internal state be state 0 with output state such that Z_i has the value 0 for all i.
3. Let Z_i have the value 1 and Z_j the value 0 ($j \neq i$) in the output state for row i.

4. Define the table entries as follows.

   a. In row 0

      In each column, the entry is the same as the least subscript of an input variable with the value 1. The entry is 0 if all x_i are 0.

   b. In row i ($1 \leq i \leq n$)

      In each column, if x_i has the value 1, the entry is (i). If x_i has the value 0, the entry is the least j such that $j > i$ and x_j has the value 1 or, if no such j exists, the least j such that x_j has the value 1. If all input variables are 0, the entry is 0.

The flow tables generated by Procedure B for n = 2 and n = 3 are shown in Table 5. These flow tables are biased because, in the case of a particular multiple request, in each row the same component is enabled regardless of which component last executed a critical section.

Theorem 7:

For each n, the flow table obtained from Procedure B is correct.

The proof of Theorem 7 will be a direct consequence of a later theorem.

Table 5. Biased Control Flow Tables From Procedure B

	x_1x_2 00	01	11	10	Z_1Z_2
0	(0)	2	1	1	00
1	0	2	(1)	(1)	10
2	0	(2)	(2)	1	01

(a) n = 2

	$x_1x_2x_3$ 000	001	011	010	110	111	101	100	$Z_1Z_2Z_3$
0	(0)	3	2	2	1	1	1	1	000
1	0	3	2	2	(1)	(1)	(1)	(1)	100
2	0	3	(2)	(2)	(2)	(2)	3	1	010
3	0	(3)	(3)	2 1		(3)	(3)	1	001

(b) n = 3

With the control algorithm defined by the control flow table, a component must wait for at most n-1 other components to execute critical sections before it executes its own critical section.

We will now consider the general class of biased control flow tables which are correct and use the minimum number of internal states.

Definition 3:

Two flow tables are said to be distinguishable if when the same input sequence is recognized by each flow table, different output sequences are produced.

If two flow tables are not distinguishable, they are indistinguishable. We will determine the number of distinguishable control flow tables that are correct and have n + 1 internal states. Next we give conditions sufficient to guarantee that Restriction 1 of the mutual exclusion problem is satisfied.

Theorem 8:

If a control flow table is such that the following conditions hold:

1. The flow table is initially stable with initial output state in which Z_i is equal to 0 for all i, i = 1, . . ., n.
2. Each output state has at most one Z_i with the value 1.

3. The value of output variable Z_i is changed from 0 to 1 only if x_i is 1.

4. The value of output variable Z_i is changed from 1 to 0 only if x_i is 0.

Then, at most one component C_i $(1 \leq i \leq n)$ may be in a critical section at any instant (Restriction 1 is satisfied).

Proof:

By condition 1 and the definition of the parallel system in Fig. 2, no component is initially in its critical section. By condition 3 and the flow table specification of operation for component C_i $(1 < i \leq n)$ in Fig. 2, a component is enabled to enter its critical section (Z_i is set to 1) only if that component is in the stable total state 2-1. If Z_i is set to 1, condition 4 ensures that it is not set to 0 until x_i becomes 0. But the flow table for component C_i shows that x_i cannot become 0 until after C_i has left its critical section and entered the stable total state 1-1. Component C_i cannot re-enter its critical section until z_i becomes 0, which only happens after x_i becomes 0 at the control input. It follows that if component C_i is in its critical section, then Z_i must have the value 1 or, equivalently, if Z_i has the value 0, component C_i is not in its critical section. The fact that at most one component may be in a critical section at any instant follows from condition 2.

The following theorem establishes the output state requirements for a correct control flow table with n + 1 internal states.

Theorem 9:

If a control flow table is correct and has n + 1 internal states, then

1. The output state for the initial internal state must have Z_i equal to 0 for all i, i = 1,...,n.
2. For each i, i = 1,...,n, there must be exactly one output state with Z_i equal to 1 and Z_j equal to 0 for all j, $j \neq i$.

Proof:

If condition 1 does not hold, the flow table is incorrect by Theorem 2. If condition 2 does not hold, then either some Z_i is never equal to 1 in any output state, which is not allowed by Theorem 1, or more than one Z_i is 1 in some output state, which is not allowed by Theorem 4.

In the remainder of this section the phrase "control flow table" refers to a control flow table with n + 1 internal states, numbered 0,1,..., n, and 2^n columns. The initial internal state is state 0 with output state in which Z_i is 0 for all i. The output state for row i, i = 1,...,n, is Z_i equal to 1 and Z_j equal to 0, $j \neq i$.

The selection of table entries in row 0 is covered by the following two theorems.

Theorem 10:

Consider a control flow table with n + 1 internal states as just defined. If this flow table is correct, the entries in row 0 must satisfy the following conditions.

1. In the column in which all input variables have the value 0, the entry is 0.
2. In the other columns, the entry is j where j is the subscript of an input variable x_j that has the value 1 in that column.

Proof:

If condition 1 does not hold, some Z_i is set to 1 when x_i is 0. By Theorem 3, the flow table is incorrect. If condition 2 does not hold, either the entry is 0, a stable entry, or the entry specifies a row with Z_i equal to 1 in a column with x_i equal to 0. The latter case is ruled out by Theorem 3. If the entry is 0, the enabling of a component depends on an input change produced by another component, which is not allowed.

Theorem 11:

Consider a control flow table with n + 1 internal states as defined earlier. If the entries in rows 1 through n are specified correctly and

1. In the column with all x_i equal to 0, the entry is 0.
2. In the other columns, the entry is j where j is the subscript of an input variable x_j that has the value 1 in that column.

then the flow table is correct and each choice of the entries in row 0 results in a distinguishable control flow table.

Proof:

Each choice of an entry in a column with some x_i equal to 1 specifies an internal state with a different output state; therefore, each choice of the row 0 entries results in a distinguishable flow table. The correctness of the flow table follows from the fact that when a 1 input value is recognized, exactly one of the components which produced a 1 input value is enabled and also from the assumption that rows 1 through n are correctly specified.

There are $2^n - (n + 1)$ entries in row 0 in columns where more than one input variable has the value 1 (n entries have exactly one input variable equal to 1 and one has no input variables equal to 1).

There are $\binom{n}{p}$ entries with exactly p input variables equal to 1*. By Theorem 9, for each of these entries there are p ways to select that entry and each selection gives a distinguishable flow table, assuming rows 1 through n are correct. Therefore, the total number of distinguishable flow tables which can be produced on the basis of row 0 alone is

$$\prod_{p=2}^{n} p^{\binom{n}{p}} = 2^{\binom{n}{2}} \cdot 3^{\binom{n}{3}} \cdot \ldots \cdot n^{\binom{n}{n}}$$

where $p^{\binom{n}{p}} = \underbrace{p.p.\ldots.p}_{\binom{n}{p} \text{ times}}$

For n = 3, Table 6 shows the possible entries in row 0.

---

* $\binom{n}{p} = \frac{n!}{p!\,(n-p)!}$

Table 6. Possible Entries in Row 0 for n = 3

$x_1x_2x_3$

	000	001	011	010	110	111	101	100
0	0	3	2,3	2	1,2	1,2,3	1,3	1

$$\prod_{p=2}^{3} p^{\binom{3}{p}} = 2^{\binom{3}{2}} \cdot 3^{\binom{3}{3}}$$

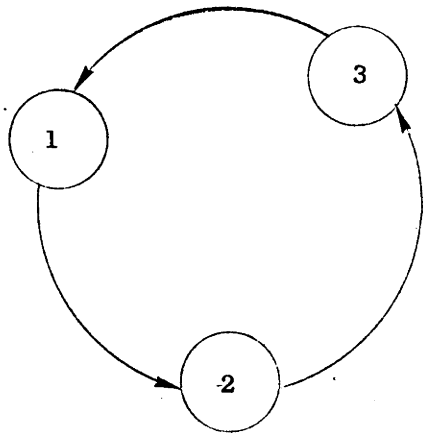
$$= 2^3 \cdot 3 = 24$$

It remains to consider how the entries in rows 1 through n may be chosen to give correct, distinguishable flow tables. The flow tables produced by Procedure B use a fixed "rule" to determine, in the case of multiple requests, which component is to be enabled next. This rule is stated in step 4b of the procedure. This rule can be restated in more general terms. For each n, there is a "chain" consisting of a circular ordering of the n integers 1,2,...,n. In the case of Procedure B, the next integer after integer i in the chain is given by the sum i (mod n) + 1. The chain for n = 3 is shown in Fig. 3a. The rule to determine the entries in row i can be restated in terms of the "chain rule" below.

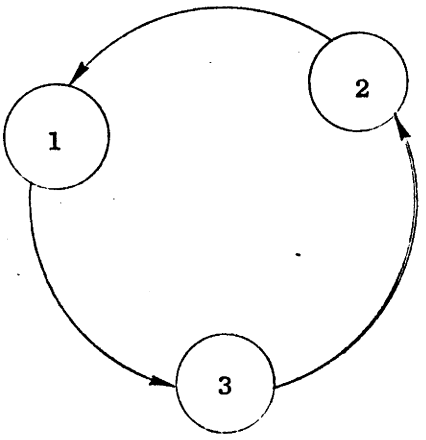
Chain Rule:

Consider row i ($1 \leq i \leq n$). The next-state entry in each column is speoified by the next integer j in the chain after or including the integer i such that x_j has the value 1 in that column. If all input variables have the value 0, the entry is 0.

The chain of Fig. 3a and the chain rule produce the same next-state entries for n = 3 as the Procedure B. For n = 3, there is one other chain that is different from the chain in Fig. 3a. This chain is shown in Fig, 3b. In general, for each n, there are (n - 1)! distinct chains since the present position is always fixed by the row number and there are (n - 1)! possible arrangements of the other n - 1 integers.



(a) Chain used in Procedure B



(b) Another chain

Figure 3. Chains which determine which component to enable for n = 3.

The importance of chains and the chain rule in the determination of the entries in rows 1 through n is demonstrated by the following two theorems.

Theorem 12:

Assume the entries in row 0 are chosen correctly. If the selection of entries in rows 1 through n of a control flow table with n + 1 internal states as defined earlier does not follow the chain rule for any chain, the flow table is incorrect.

Proof:

There are two ways to violate the chain rule. One way is to have a non-zero entry in the column with all input variables equal to 0. As a result, some output variable Z_i is set to 1 when x_i is 0 and the flow table is incorrect by Theorem 3. The other way involves the selection of next-state entries in columns with at least one input variable equal to 1. Suppose the rule is violated in the case exactly one input variable is equal to 1. Then the flow table is incorrect by Theorem 3. Next, consider the violation of the chain rule when more than one input variable has the value 1. If, in row i, the-violation occurs such that row i is left when x_i has the value 1, the flow table is incorrect by Theorem 6. Suppose this is not the case. There must be an entry in some row, say row i, such that for some input state with at least two input variables, x_j and x_k, $(j,k \neq i)$ equal to 1, the

rule specifies that the entry should be j and the entry is k instead. Thus, component C_k enters its critical section next instead of C_j. Suppose that, while in row k, input variable x_i becomes equal to 1 again (component C_i wants to re-enter its critical section). When x_k becomes 0, both x_i and x_j are equal to 1 along with all other input variables that were equal to 1 in row i or became equal to 1 while the control component was in internal state k. The relative positions of i, j, and k in the chain are shown in Fig. 4. From row k, component C_i must be enabled (row i must be entered) before component C_j is enabled. Otherwise, the chain rule is not violated. Suppose that while in row i and when x_i becomes 0, the input state recognized is exactly the same as the previous time the flow table was in row i. As before, row k is entered next. For this pattern of requests, row j is never entered and component C_j never enters its critical section. Restriction 2 is violated and the flow table is incorrect.

Theorem 12 is equivalent to saying that if the entries in row 0 are specified correctly and. the flow table is correct, then the entries in rows 1 through n must satisfy the chain rule for some chain. This establishes the necessity for using the chain rule. An example of the violation of the chain rule for all chains is shown in Fig. 5. The 2-101 entry is incorrect for the chain of Fig. 3a (it should be 3). For the chain of Fig. 3b, the 1-011 entry should be 3 and the 3-110 entry should be 2. The undesired transitions are also shown in Fig. 5.

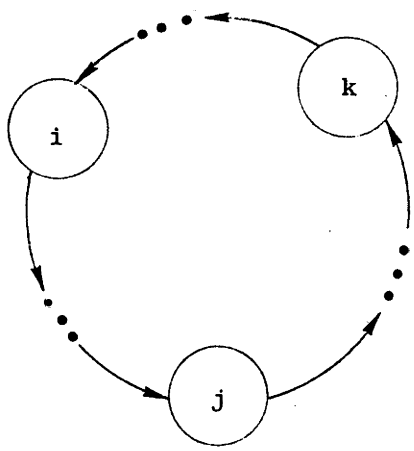


Figure 4. Relative chain positions of i, j, and k.

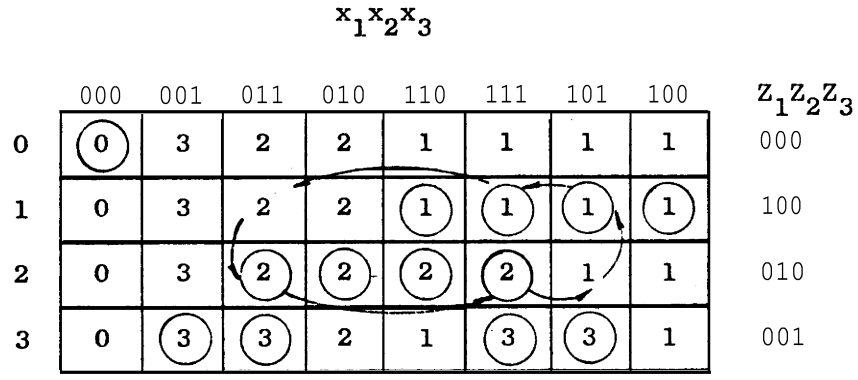


Figure 5. Incorrect flow table due to the violation of the chain rule (n = 3).

Either i = 2, j = 3, and k = 1 with initial input state 111 or i = 1, j = 3, and k = 2 with initial input state 111.

No two chains result in exactly the same specification of next-state entries and each assignment gives a distinguishable flow table for some input sequence. Therefore the total number of distinguishable, correct flow tables which have n + 1 internal states is

$$(n-1)! \prod_{p=2}^{n} p^{\binom{n}{p}} .$$

The values of this expression for n = 2, 3, and 4 are given in Table 8.

The following theorem establishes that following the chain rule with some chain to fill in the entries in rows 1 through n is sufficient to solve the mutual exclusion problem if the entries in row 0 are specified correctly.

Theorem 13:

Given a control flow table with n + 1 internal states as described earlier. If the entries in row 0 are chosen correctly and the entries in rows 1 through n are chosen using the chain rule with a fixed chain, the flow table is correct.

Table 7. Number of Distinguishable, Correct Flow Tables With n + 1 Internal States

n	$(n-1)! \prod_{p=2}^{n} p^{\binom{n}{p}}$
2	2
3	48
4	124, 416

Proof:

The flow table definition and the entries determined by the chain rule satisfy conditions 1 - 4 of Theorem 8; therefore, Restriction 1 of the mutual exclusion problem is satisfied. Suppose Restriction 2 can be violated. That is, some component C_i is such that x_i is 1 but Z_i is never set to 1. If there are not multiple requests (more than one input variable is 1), C_i will be enabled. Therefore more than one input variable must be 1. Rut, the definition of the chain rule and the structure of a chain guarantee that after at most n - 1 components enter their critical sections, component C_i must be enabled.

Combining the results of Theorems 10 - 13, we have the following theorem which establishes necessary and sufficient conditions for a correct control flow table with n + 1 internal states.

Theorem 14:

Consider a control flow table with n + 1 internal states as defined earlier. This flow table is correct if and only if

1. In row 0
   a. In the column with all x_i equal to 0, the entry is 0.
   b. In the other columns, the entry is j where j is the subscript of an input variable x_j that has the value 1 in that column.

2. In rows 1 through n, the entries are chosen by using the chain rule with a fixed chain.

The proof of Theorem 7 follows as a corollary of Theorem 14. Notice that Theorem 14 defines a general procedure which can be used to construct a correct flow table with n + 1 internal states. This procedure can be used instead of the Procedure B defined earlier.

This completes our discussion of control flow tables that have the minimum number of internal stat-es.

Unbiased Control Flow Tables

Let us consider unbiased control flow tables (Definition 2). By Theorem 2, there must be at least one row with output state for which Z_i is 0 for all i, i = 1,...,n. By Theorem 1, there must be at least one row for which Z_i is 1 and Z_j is 0 for all j, $j \neq i$, $1 < i,j < n$. There must be n! rows with all output variables equal to 0. Each row must correspond to a unique past history'of critical section executions. This must be done in order to

ensure that when multiple simultaneous requests are recognized, the component enabled is the component which has been out of its critical section the longest. There must be (n-1)! rows with exactly one Z_i equal to 1 for each i, i = 1,...,n. This is also required to decide which component to enable when component C_i leaves its critical section. Thus in order to be unbiased, the control flow table must have "perfect memory". The total number of internal states is $n! + n\,(n-1)! = 2n!$. The following theorem has been established.

Theorem 15:

For a given n, an unbiased control flow table must have at least $2n!$ internal states.

Theorem 5 and Theorem 15 show that the cost, in terms of the number of internal states, of providing unbiased service is rather high.

The following procedure can be used to construct an unbiased control flow table for any specified value of n.

Procedure U

(Unbiased Control Flow Table With $2n!$ Internal States)

1. Define a flow table-with 2^n columns and $2n!$ internal states.
2. Let the first $n!$ internal states (rows) have the output state with all Z_i equal to 0. Give each of these rows a unique tag which is one possible order of the subscripts of the n components $C_1,\ldots,C_n$*.

* We adopt the convention that the leftmost element of the tag is the subscript of the component which was most recently (or currently) in its critical section. The rightmost tag element refers to the component least recently in its critical section.

3. Divide the remaining n! rows into n groups of (n-1)! rows. In group i, i = 1,...,n, let the output state be Z_i equal to 1 and Z_j equal to 0, $j \neq i$. Give each row in group i a unique tag which has i in the first position and the subscripts of the other components in the other positions. (A control flow table as specified thus far is given in Table 8 for n = 3)

4. The table entries are determined as follows:

   a. In rows with all Z_i equal to 0,

      (1) there is a stable entry in the column with all input variables equal to 0. Other columns have unstable entries.

      (2) For entries which are unstable, compute a sub-order from the present tag which is the tag positions which are the subscripts on an input variable which is equal to 1. Form a new tag using the last element of the sub-order as the first element of the new tag, the remaining elements of the sub-order as the final elements, and the remaining elements of the original tag as the middle elements (For tag 2, 1, 3, 4, the input states and new tags are shown in Table 9). The unstable entry is the number of the row with the new tag which has an output variable equal to 1.

Table 8. Flow Table Set-Up for n = 3

		$x_1x_2x_3$								
tag	s	000	001	011	010	110	111	101	100	$Z_1Z_2Z_3$
1,2,3	1									000
1,3,2	2									000
2,1,3	3									000
2,3,1	4									000
3,1,2	5									000
3,2,1	6									000
1,2,3	7									100
1,3,2	8									100
2,1,3	9									010
2,3,1	10									010
3,1,2	11									001
3,2,1	12									001

t

S

least recent

most recent

Table 9. Tag for Next Row Given 2,1,3,4 as Present Tag in a Row With all Z_i Equal to 0

Input State $(x_1x_2x_3x_4)$	Sub-Order	New Tag
0001	4	4,2,1,3
0010	3	3,2,1,4
0011	3,4	4,2,1,3
0100	2	2,1,3,4
0101	2,4	4,1,3,2
0110	2,3	3,1,4,2
0111	2,3,4	4,1,2,3
1000	1	1,2,3,4
1001	1,4	4,2,3,1
1010	1,3	3,2,4,1
1011	1,3,4	4,2,1,3
1100	2,1	1,3,4,2
1101	2,1,4	4,3,2,1
1110	2,1,3	3,4,2,1
1111	2,1,3,4	4,2,1,3

b. In rows with exactly one output variable, say Z_i, equal to 1, the first element of the tag must be i.

(1) If all input variables are 0, the new row is the row with the same tag and all output variables equal to 0.

(2) If input variable x_i is equal to 0, compute a new tag as in Step **4a(2).** The unstable entry is the number of the row with the new tag.

(3) If p input variables are equal to 1, $1 \leq p \leq n$ and X_i is equal to 1, find the sub-order of elements in the present tag for which the corresponding input variables have the value 1. If the sub-order consisting of the p - 1 elements which are not equal to i is exactly the same as the sub-order consisting of the final p - 1 elements in the present tag, the entry is a <u>stable entry</u>. If not, form a new tag using the sub-order corresponding to the p-1 input variables which are 1 as the suffix of the new tag and the remaining elements of the present tag as the prefix. The unstable entry is the number of the row with the new tag which has an output variable equal to 1 (For n = 4 and present tag **1,4,2,3** there are stable entries in columns $x_1x_2x_3x_4$ = **1000,** 1010, 1110, 1111. Each row has n stable entries. A complete list of the new tags for the row with tag **2,1,3,4** and Z_2 equal to 1 is shown in Table **10.**

Table 10. Tags for Next Row Given **2,1,3,4** as the Present Tag and Z_2 Equal to 1 for the Present Row

Input State	Sub-Order	New Tag
0000	———	**2,1,3,4**
0001	4	**4,2,1,3**
0010	3	**3,2,1,4**
0011	**3,4**	**4,2,1,3**
0100	2	**2,1,3,4** (stable)
0101	**2,4**	**2,1,3,4** (stable)
0110	**2,3**	**2,1,4,3**
0111	**2,3,4**	**2,1,3,4** (stable)
1000	1	**1,2,3,4**
1001	**1,4**	**4,2,3,1**
1010	**1,3**	**3,2,4,1**
1011	**1,3,4**	**4,2,1,3**
1100	**2,1**	**2,3,4,1**
1101	**2,1,4**	**2,3,1,4**
1110	**2,1,3**	**2,4,1,3**
1111	**2,1,3,4**	**2,1,3,4** (stable)

The control flow tables generated by this procedure for n = 2 and n = 3 are shown in Tables 11 and 12, respectively.

The key steps in Procedure U are those which determine the next row to be entered. This is always done in such a manner that the component which has been out of its critical section the longest will be enabled next, the modification of the tags in Step **4a(2)** and **4b(3)** ensures that in the **presence** of multiple requests, the control will first enable the component which has been out of its critical section the longest and make the other components with pending requests next in line after the first component enabled executes its critical section. Procedure U satisfies conditions 1-4 of Theorem 8, so Restriction 1 of the mutual exclusion problem must be satisfied. The fact that Restriction 2 is also satisfied follows directly from the way the procedure determines the next component to be enabled. It follows that:

Theorem 16:

For any given n, Procedure U gives a correct control flow table which is unbiased.

A direct consequence of the definition of an unbiased control flow table is the fact that for a given row with its past history of critical section executions (tag) and for a given input state,the flow table entry is always uniquely specified.

Table 11. Unbiased Control Flow Table From Procedure U, n = 2

		x_1x_2				
tag	s	00	01	11	10	Z_1Z_2
1,2	1	(1)	4	4	3	00
2,1	2	(2)	4	3	3	00
1,2	3	1	4	(3)	(3)	10
2,1	4	2	(4)	(4)	3	01
		S				

Table 12. Unbiased Control Flow Table From Procedure U, n = 3

tag	S	$x_1x_2x_3$ = 001	011	010	110	111	101	100	$Z_1Z_2Z_3$
1,2,3	1	(1) 11	11	9	10	11	12	7	000
1,3,2	2	(2) 11	9	9	10	9	12	8	000
2,1,3	3	(3) 12	11	9	8	12	12	7	000
2,3,1	4	(4) 12	11	10	8	7	7	7	000
3,1,2	5	(5) 11	9	10	10	10	7	8	000
3,2,1	6	(6) 12	9	10	8	8	7	8	000
1,2,3	7	1 11	11	9	8	(7)	(7)	(7)	100
1,3,2	8	2 11	9	9	(8)	(8)	7	(8)	100
2,1,3	9	3 12	(9)	(9)	10	(9)	12	7	010
2,3,1	10	4 12	9	(10)	(10)	(10)	7	7	010
3,1,2	11	5 (11)	(11)	10	10	(11)	7	8	001
3,2,1	12	6 (12)	11	10	8	(12)	(12)	8	001

S

Theorem 17:

For each n, the correct control flow table that is unbiased and has 2n! internal states is unique*.

CENTRAL VERSUS DISTRIBUTED CONTROL

To this point, all control and decision-making has been performed in a single component. As the number of components controlled (n), becomes large so does the number of inputs and outputs for this component. Fortunately, however, we have shown that the number of internal states which are necessary to obtain a correct solution for each n is just n + 1. In this section, we will briefly discuss some alternate control organizations. Rather than concentrating all decision-making in a single control component we will consider the consequences of using more than one. Before proceeding, we must introduce some terminology which will be useful in the following discussion. The question of exactly where the "control" lies in a given system is rather difficult to specify. Thus far, for the mutual exclusion problem, we have considered the control function to reside in component C_{n+1} of Fig. 2. This is natural because of the interpretation we attach to this system; however it is also conceivable that in some situation the control function might be thought to reside the components $C_1, C_2, \ldots, C_n$ which in turn drive the single component C_{n+1}.

---

* except for the numbering of the internal states.

We will resolve such ambiguities in an arbitrary way be simply stating for a given system where the control responsibility is assumed to be. In the case of the mutual exclusion problem as shown in Fig. 2 the control is assumed to be in component C_{n+1}.

The parallel system in Fig. 2 is an example of a system with a central control mechanism. More precisely we say the following:

Definition:

In a given parallel system, if the control function is performed by a single component, that system is said to have central control.

If a system does not have central control, it is said to have distributed control; that is, the control function is performed-by more than one component. For a given circuit it is often possible to partition the circuit in many different ways, making it difficult to determine if the circuit represents a single component or several components which communicate with each other. In the case of a parallel system, as defined in Definition 1, we distinguish components on the basis of the delay assumptions for a parallel system. Any circuit in which it must be assumed that delays are finite and bounded in order to ensure correct operation is considered to be a single component. On the other hand,if it is possible to partition a circuit such that the parts can communicate even though the delays in the interconnecting lines are finite and unbounded, the parts are considered to be separate and distinct components.

Linear Control

In this section, distributed control solutions to the mutual exclusion problem are considered. One type of distributed control is shown in Fig. 6. In this case each component C_i, $1 \leq i \leq n$, has its own control component C_{n+i} with which it communicates in the same manner as components communicate with the central control in Fig. 2. Before component C_i can be enabled, its control component must communicate with its left and right neighbor control components to determine if it is possible for C_i to enter its-critical section. With this type of organization it is necessary to propagate a request for permission to enter a critical section to all control components. We assume each component can communicate only with its left and right neighbors. Because line delays are unbounded, when a control component produces an output value transition it must recognize an input value change produced in recognition of the propagation of its own output value before it can proceed.* This means that pairs of lines are required, one to send a request and one to receive the reply. Furthermore a control component must not only send requests to its neighbors but receive requests from them as well. The general form of a control component-is shown in Fig. 7.

---

* A further discussion of the consequences of the line delay assumption is given in [6].

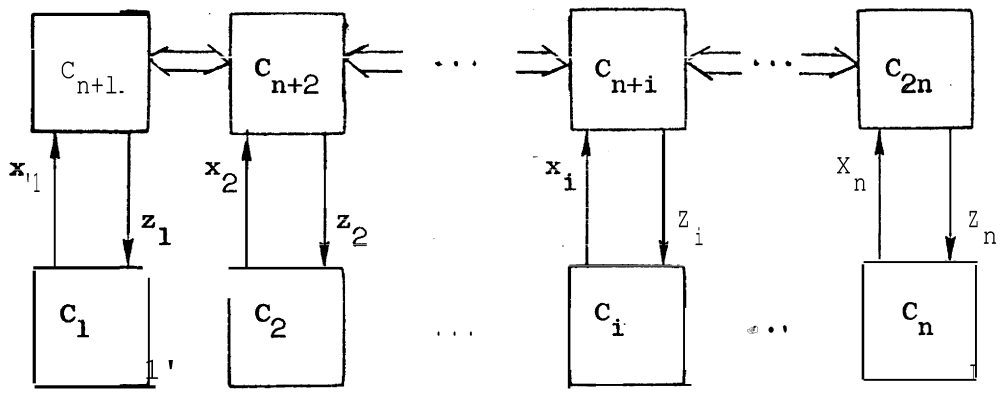
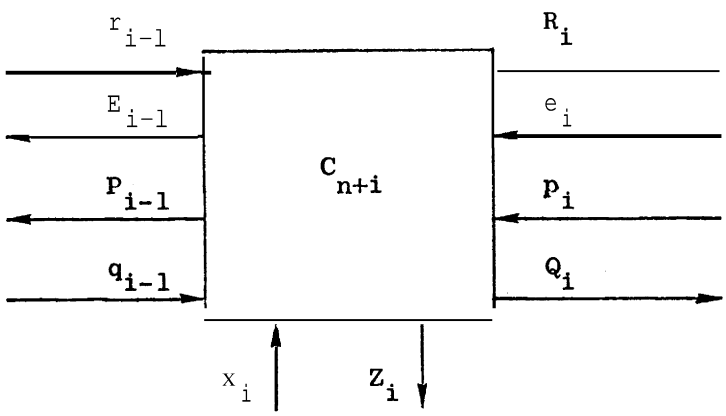


Figure 6. A linear distributed control for the mutual exclusion problem.



$(1 \leq i \leq n)$, x_i, Z_i as before

$R_i = \begin{cases} 1 & \text{request for permission to enable } C_k \ (k \leq i) \\ 0 \text{ no} & \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \end{cases}$

$e_i = \begin{cases} 1 & \text{permission to enable } C_k \ (k < i) \\ 0 \text{ no} & \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \end{cases}$

$P_{i-1} = \begin{cases} 1 & \text{request for permission to enable } C_k \ (k \geq i) \\ 0 \text{ no} & \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \end{cases}$

$q_{i-1} = \begin{cases} 1 & \text{permission to enable } C_k \ (k > i) \\ 0 \text{ no} & \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \end{cases}$

$r_{i-1} = \begin{cases} 1 & \text{request for permission to enable } C_k \ (k < i) \\ 0 \text{ no} & \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \end{cases}$

$E_{i-1} = \begin{cases} 1 & \text{permission to enable } C_k \ (k < i) \\ 0 \text{ no} & \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \end{cases}$

$p_i = \begin{cases} 1 & \text{request for permission to enable } C_k \ (k > i) \\ 0 \text{ no} & \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \end{cases}$

$Q_i = \begin{cases} 1 & \text{permission to enable } C_k \ (k > i) \\ 0 \text{ no} & \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \end{cases}$

Figure 7. General form of a control component for Fig. 6.

For correct operation, each control component must have a sufficient number of internal states to remember whether a component to his left, right,or his own component was in its critical section last. This information is necessary to resolve ties which result when multiple requests are recognized and also to know whether to pass permission to enable to another component or to wait for that component to pass permission to enable to you. Only with this information can it be guaranteed that no component is permanently excluded from its critical section. We conclude that at least three internal states are required for each general control component (the leftmost and rightmost control components need to remember only whether their own component or a neighbor was in its critical section last). The actual number of internal states is difficult to calculate and we will not do so here. However on the basis of this examination of the control structure in Fig. 6 we can conclude that the total number of internal states required for control will be at least twice as many as for a central control. Of course the number of inputs and the structure of each control component in Fig. 6 is fixed so we can add components simply by adding another control component without any redesign. Without examining actual implementations of central control components, which we do not propose to do in this paper, it is difficult to make any evaluation of either approach on a basis other than the total number of internal states required for the control function.

The distributed nature of the control affects the choice of the initial internal state for each control component. That is, if each control component is started in the same

initial state it is always possible for the system to be incorrect. Suppose each control component were initialized so that it thought its own component had been out of its critical section the longest. Then it would be possible for simultaneous requests to arrive and for each control to wait indefinitely for enabling permission from its neighbors. As a result only certain combinations of initial control component states can be used. For example if n = 4 then we might initialize C_5 to think Cl was in its critical section last and C_6, C_7, and C_8 to think their left neighbor was in a critical section last. In the use of multiple requests, C_5 must give permission to C_6 before it waits for permission to enable C_1. The conclusion of this discussion is that there must be a certain "asymmetry" in the choice of the initial internal states for the control functions.

We classify the form of control in Fig. 6 as linear because it is possible to arrange the components in the manner shown in the figure where each control component communicates only with its left and right neighbors [11]. Many other forms of linear control are also possible in which groups of components could communicate with a common control which would then communicate with its neighbors. The general organization is shown in Fig. 8. It is our conjecture that in all such organizations the number of internal states for the control is always greater than the number required for a central control.

Hierarchical Control

A different distributed control structure is shown in Fig. 9. This structure is know as a hierarchical structure because in this case before a component can be enabled permission must be received

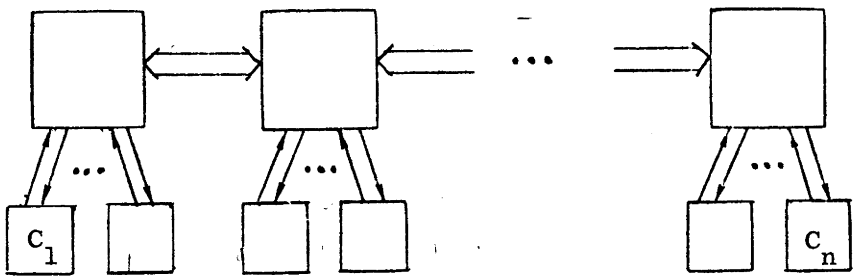


Figure 8. General organization for a linear distributed control.

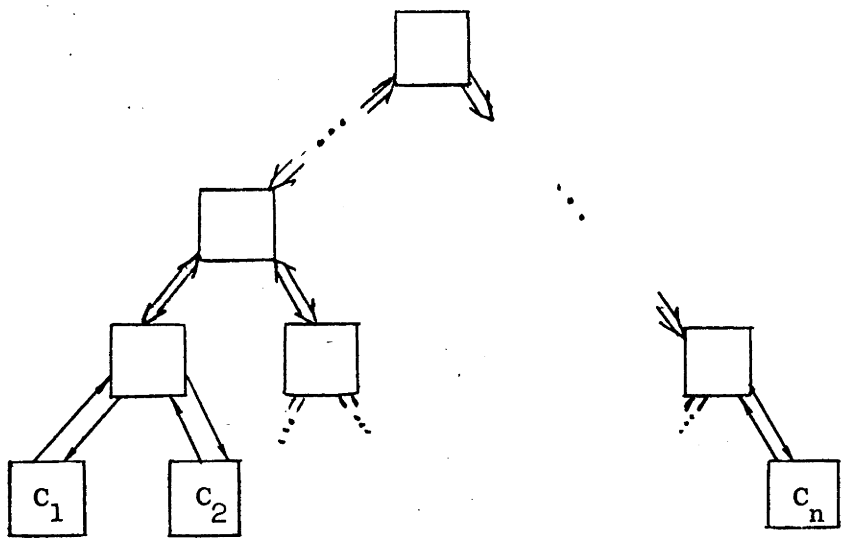


Figure 9. A binary tree hierarchical control structure,

from the control components higher in the control "tree". Fig. 9 is an example of a binary control tree where each lower level control communicates with two components and higher level control components communicate with two lower level components. It can be shown that such a tree has n-1 control components (non-terminal nodes) and by arguments similar to those in the last section it follows that (1) there must be an "asymmetry" in the choice of the initial control state and (2) the total number of internal states required for control components is greater than the number required for central control. We conjecture these conclusions are valid for hierarchical tree control structures which are not restricted to be binary.

Both linear and hierarchical control structures are biased because the control cannot store the complete history of accesses to critical sections.

CONCLUSIONS

The use of the flow table model has made it possible to characterize in a precise way the cost of a correct solution to the mutual exclusion problem as measured by the number of internal states required by the control function. In addition, procedures can be given to generate correct control flow tables.

Distinctions between central and distributed (linear, hierarchical) control can also be made in this model and the effects of one type Of control over the other evaluated. More work needs to be done in this area.

REFERENCES

[1] Adams, D.A. A computation model with data flow sequencing. CS-117 (Thesis), Computer Science Department, Stanford University, Stanford, California (Dec 1968).

[2] Adams, D.A. A model for parallel computations. Proc. Symp. on Parallel Processor Systems, Technologies, and Applications, Naval Postgraduate School, Monterey, Calif. June, 1969 (in press).

[3] Ashcroft, E. and Manna, Z. Formalization of properties of parallel programs. AIM-110, Artificial Intelligence Project, Stanford University, Stanford, Calif. (Feb 1970).

[4] Bredt, T.H. and McCluskey, E.J. A model for parallel computer systems. Technical Report No. 5, SEL Digital Systems Laboratory, Stanford University, Stanford, California (Apr 1970).

[5] Bredt, T.H. Analysis and synthesis of concurrent sequential programs. Technical Report No. 6, SEL Digital Systems Laboratory, Stanford University, Stanford, Calif. (May 1970).

[6] Bredt, T.H. Analysis of parallel systems. Technical Report No. 7, SEL Digital Systems Laboratory, Stanford University, Stanford, Calif. (to appear).

[7] Dennis, J.B. and Van Horn, E.C. Programming semantics for multiprogrammed computations. Comm. ACM 9 (March 1966). 143-155.

[8] Dijkstra, E.W. Solution of a problem in concurrent programming control. Comm. ACM 8 (Sept 1965), 569.

[9] Dijkstra, E.W. The structure of the "THE" multiprogramming system. Comm. ACM 11 (May 1968), 341-346.

[10] Dijkstra, E.W. Co-operating sequential processes.in Programming Languages, Genuys, F. (Ed.), Academic Press New York (1968).

[11] Hennie, F.C. Finite State Models for Logical Machines. John Wiley and Sons, New York, N.Y. (1968).

[12] Karp, R.M. and Miller, R.E. Properties of a model for parallel computations: determinacy, termination, queueing. SIAM J. Appl. Math., 14 (Nov 1966), 1390-1411.

[13] Karp, R.M. and Miller, R.E. Parallel program schemata: a mathematical model for parallel computation. IEEE Conference Record of the 8th Annual Symposium on Switching and Automata Theory (Oct 1967), 55-61.

[14] Karp, R.M. and Miller, R.E. Parallel program schemata. J. of Computer and System Sciences 3, 2 (May 1969), 147-195.

[15] Knuth, D.E. Additional comments on a problem in concurrent programming control. Comm. ACM, 9 (May 1966), 321-322.

[16] Lampson, B.W. A scheduling philosophy for multiprocessing systems. Comm. ACM 11 (May 1968), 347-360.

[17] Luconi, F.L. Completely functional asynchronous computational structures. IEEE Conference Record of the 8th Annual Symposium on Switching and Automata Theory (Oct 1967), 62-70.

[18] Luconi, F.L. Asynchronous computational structures. MAC-TR-49 (Thesis), Massachusetts Institute of Technology, Cambridge, Massachusetts (Feb 1968).

[19] Luconi, F.L. Output functional computational structures. IEEE Conference Record of the 9th Annual Symposium on Switching and Automata Theory (Oct 1968), 76-84.

[20] Manna, Z. Termination of algorithms. Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania (Apr 1968).

[21] Manna, Z. Properties of programs and the first-order predicate calculus. J. ACM (Apr 1969).

[22] Manna, Z. The correctness of programs. J. of Computer and System Sciences, 3 (May 1969).

[23] Manna, Z. The correctness of non-deterministic programs. Artificial Intelligence J. 1, 1 (1970).

[24] McCluskey, E.J. Introduction to the Theory of Switching Circuits. McGraw-Hill Book Co., New York, N.Y. (1965).

[25] Muller, D.E. and Bartky, W.S. A theory of asynchronous circuits. Proc. of an International Symposium on the Theory of Switching, the Annals of the Computation Laboratory of Harvard University, Vol. 29, Part I, Harvard University Press (1959), 204-243.

[26] **Rodriquez,** J.E. A graph model for parallel computations. Ph.D Thesis, MIT, Department of Electrical Engineering, Cambridge, Massachusetts (Sept 1967).

[27] Slutz, D.R. The flow graph schemata model of parallel computation. **MAC-TR-51** (Thesis), MIT, Cambridge, Massachusetts (**Sept** 1968).