SU-SEL-70-057

Analysis of Parallel Systems

by

T. H. Bredt

August 1970

Technical Report No. 7

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-70-172 SEL-70-057

ANALYSIS OF PARALLEL SYSTEMS

by

T. H. Bredt

August 1970

Technical Report No. 7

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 7
August, 1970

ANALYSIS OF PARALLEL SYSTEMS

by

T. H. Bredt

ABSTRACT

A formal analysis procedure for parallel computer systems is presented. The flow table model presented in a earlier paper* is used to describe a system. Each component to the system is described by a completely specified fundamental-mode flow table. All delays in a parallel system are assumed to be finite. Component delays are assumed to be bounded and line delays unbounded. The concept of an output hazard is introduced to account for the effects of line delay and the lack of synchronization among components. Necessary and sufficient conditions for the absence of output hazards are given.

The state of a parallel system is defined by the present internal state and input state of each component. The operation of the system is described by a system state graph which specifies all possible state transitions for a specified initial system state. A procedure for

* 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).

constructing the system state graph is given. The analysis procedure may be summarized as follows. A problem is stated in terms of restrictions on system operation. A parallel system is said to operate correctly with respect to the given problem if the associated restrictions are always satisfied. The restrictions specify either forbidden system states, which are never to be entered during the operation of the system, or forbidden system state sequences, which must never appear during system operation. The restrictions are tested by examining the system state graph. A parallel system for the two-process mutual exclusion problem is analyzed and the system is shown to operate correctly with respect to this problem. Finally, the conditions of determinacy and output functionality, which have been used in other models of parallel computing, are discussed as they relate to correct solutions to the mutual exclusion problem.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

A major concern in computer science is the development of formal procedures for the analysis of programs and algorithms [1, 1a, 8, 12, 18, 19, 20, 21]. There is also much interest in the development of a common basis for the description of programs and circuits, that is, software and hardware. In [2], we have defined a flow table model for parallel computer systems which uses fundamental-mode flow tables to describe the operation of each system component. Procedures for synthesizing and analyzing sequential circuits using flow tables are well known [22]. In [3], analogous procedures are developed for a class of sequential programs, allowing flow tables to be used as a common link between programs and circuits.

The purpose of the flow table model is to aid in the study of the interactions of system components which are operated concurrently. Algorithms which control these interactions usually never terminate in the way that an algorithm for sorting or inverting a matrix does and as a result different analysis methods are required.

A classic problem in parallel systems is the mutual exclusion problem stated below for two components.

Problem: (Mutual Exclusion)*

Given two components, which operate concurrently and which contain "critical sections", control these components so that the following two restrictions are always satisfied:

Restriction 1: It is impossible for two components to be in their critical sections simultaneously.

Restriction 2: If a component wants to enter a critical section, it is eventually allowed to do so.

The components in this problem usually represent the process of executing a program. The exact content or nature of the critical sections is not important in the development of a solution to this problem. Typically, a critical section contains an access to a common memory location, modification of a system table, etc.. Solutions to this problem usually assume the exclusive execution of certain primitive operations [4, 5, 6, 7, 12, 14]. Components which use these primitive operations communicate by accessing common memory locations. In the flow table model, components communicate by changing values on lines (physical wires) which interconnect them. The lines carry binary level signals and it is assumed that there is no bound on the time for value changes to propagate along the lines. The primitive operations in the flow table model are the change of the value on an interconnecting line and the recognition of a value change on one of these lines. We do not assume exclusive execution of these primitives.

---

* This is a slightly different version of the problem considered by Dykstra [5, 7]. Dykstra did not require that a given program must enter its critical section but rather that the decision as to which program would enter its critical section could not be postponed indefinitely.

In [2, 3] a solution for the two-process mutual exclusion problem was designed using the flow table model. In this paper, formal analysis procedures for the flow table model are given and these procedures are used in the analysis of this solution for the mutual exclusion problem.

PARALLEL SYSTEMS

We begin by giving a definition of a parallel system.

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 distinct input variables called the component input set $I_i = \{x_{i_1}, x_{i_2}, \ldots, x_{i_n}\}$ $1 \leq i_j \leq M$; $j = 1, \ldots, n$ and a set of distinct output variables called the component output set $O_i = \{X_{i_1}, X_{i_2}, \ldots, X_{i_m}\}$, $1 \leq i_j \leq M$; $j = 1, \ldots, m$. 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 binary 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 variable input. The operation of each component is described by a completely specified flow table [22] 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 values of component input and output variables define the component input state and output state, respectively. The assumptions about physical delays present in a parallel system are the following:

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.

The parallel system designed in [2] as a solution for the two-process mutual exclusion problem is shown in Fig. 1. The initial internal state for each component is internal state 1. Components C_1 and C_2 contain critical sections which are entered and left exactly once when the component is in internal state 2 with the 1 input state.

Our definition of a parallel system is similar to the definition of a circuit given by David Muller in his study of speed independence [23]. Muller assumes that each component (element) has a single output and allows lines to take integer values which are not restricted to be binary. The delay assumptions made in Muller's model are

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

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

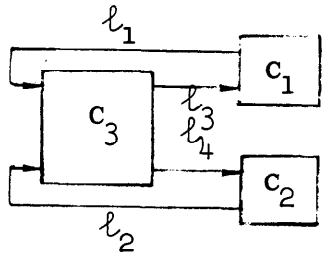
$\ell_1 = (X_1, x_1)$, $\ell_2 = (X_2, x_2)$

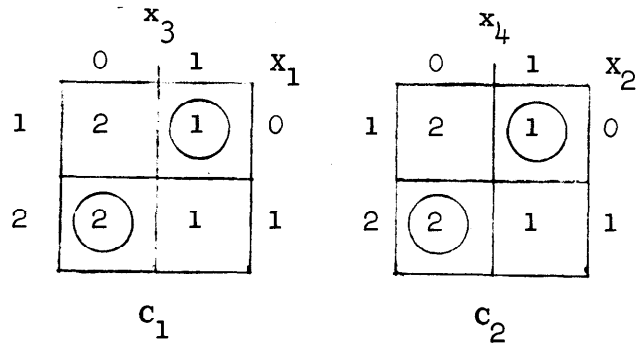
$\ell_3 = (X_3, x_3)$, $\ell_4 = (X_4, x_4)$

$O_1 = \{X_1\}$, $I_1 = \{x_3\}$

$O_2 = \{X_2\}$, $I_2 = \{x_4\}$

$O_3 = \{X_3, X_4\}$, $I_3 = \{x_1, x_2\}$





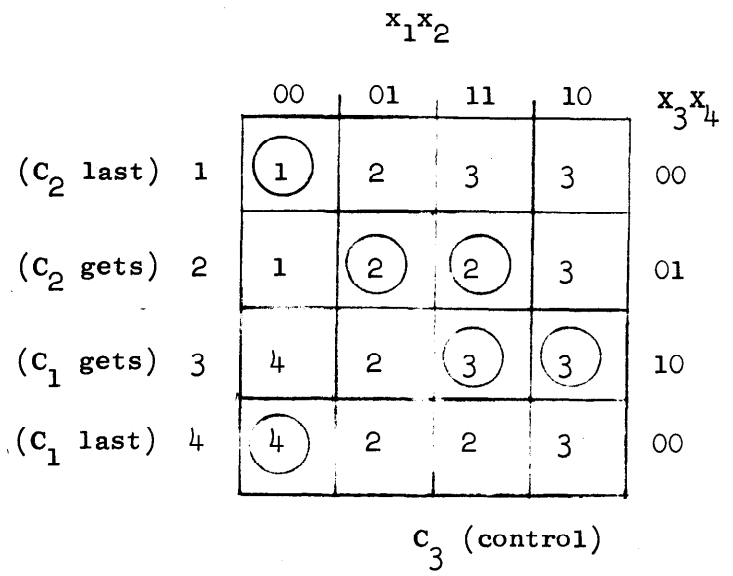


Figure 1. Parallel system for the two-component mutual exclusion problem.

different from those made here. Muller assumes that all line delays are zero. In cases where line delays can affect system operation, he assumes that special delay components (elements) are specified in the model. In our model, line delays are explicitly accounted for.

Our parallel systems are also similar to the definition of a finite-state computational schema given by Luconi [15]. Luconi uses functions rather than flow tables to describe component behavior and assumes that delays are present only in components and not in lines. In Luconi's model, components communicate by means of values stored in memory cells rather than by level signals on interconnecting lines. Luconi assumes that component operations are never performed simultaneously.

The intent of our line delay assumption is that line delays cannot be controlled. It is not assumed that, when a component C_i changes the value of an output variable, the value change necessarily propagates to the component C_j at the other end of the interconnecting line or, if the value does propagate, that it is recognized by C_j. When components operate in this manner, it is not clear what it means for a component to "recognize" an input change. The following discussion is intended to clarify this point. Basic component operation, as described in [2], consists of two phases. In the first phase, the present input values (input state) are recorded in a rank of flip-flops called the input rank. In the second phase, these input values and the present internal state of the component determine

the component response. When this response is complete, the two-phase cycle of operation begins again. A definition of the recognition of an input value is given below.

Definition 2:

> Given a value change for a component input variable, the new value is said to be recognized by the component if the new value is recorded in the appropriate flip-flop of the input rank during the first phase of some cycle of component operation as described above.

The following example illustrates a situation in which an input change may not be recognized by a component. The parallel system is defined in Fig. 2. Initially, component C_3 is unstable and will enter internal state 2, setting X_2 and X_3 to 1. At this point the following sequence of events may occur. x_2 becomes 1. C_1 recognizes this input change and enters internal state 2, setting X_1 to 1. x_1 becomes 1. C_3 recognizes this input change and enters internal state 1, setting X_2 and X_3 to 0. This entire sequence may be completed such that either x_3 never becomes 1 or is equal to 1 for such a short time that C_2 never recognizes the change in the value of x_3 from 0 to 1. The possibility of spurious input value transitions is undesirable under all normal circumstances and should be avoided. Before investigating these spurious transitions further, let us consider the nature of the difficulties encountered in this example.

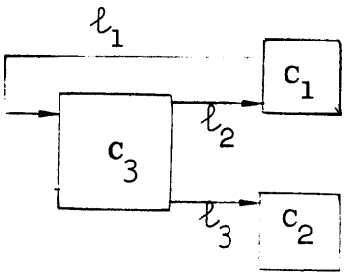
$\mathcal{C} = \{C_1, C_2, C_3\}$

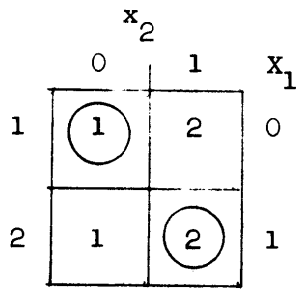
$\mathcal{L} = \{\ell_1, \ell_2, \ell_3\}$

$O_1 = \{X_1\}$, $I_1 = \{x_2\}$

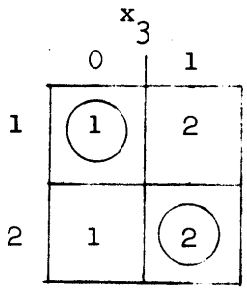
$O_2 = \phi$, $I_2 = \{x_3\}$

$O_3 = \{X_2, X_3\}$, $I_3 = \{x_1\}$



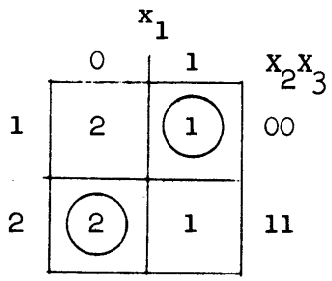


a) C_1



b) C_2

(initial internal state is 1 for C_1, C_2, and C_3)



c) C_3

Figure 2. Example parallel system.

An apparent source of trouble is the lack of an interconnection from component C_2 to component C_3. However, the existence of such a connection is not sufficient to guarantee that C_2 recognizes all input changes. In particular, C_3 can ignore all inputs from C_2 in deciding when to change the value of X_3. A precise statement of sufficient conditions for the 1 value for X_3 to be recognized is given later. While a connection from C_2 to C_3 can result in the recognition of the output value for X_3, such a connection is not always necessary if the component flow tables are modified slightly. Consider the three flow tables in Table 1 as alternatives to those shown in Fig. 2. Now when C_3 sets X_3 to 1, the value of X_3 is not changed for as long as the system continues to operate, presumably an infinite time. Since line and component delays are finite, C_2 must eventually recognize the value change for x_3.

Output Hazards

The following definitions and theoretical results are introduced to develop a more formal understanding of the difficulties present in this example. Consider components C_i and C_j and a line $\ell_k = (X_k, x_k)$ from C_i to C_j.

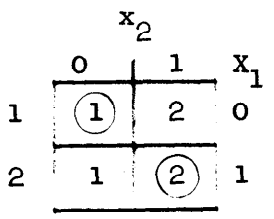
Definition 3:*

Let a and b be internal states of C_i for which output variable X_k has the values 1 and 0, respectively. An output 1 hazard is the possibility of a transition from a to b when input variable x_k has the value 0.

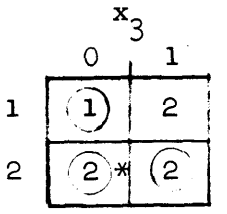
---

* Output hazards are examples of transformation-losses in lines in the terminology of Luconi [15, 16, 17]. They are violations of the semi-modularity condition of Muller [23]. These hazards have been called "resolution hazards" by Wood [26].

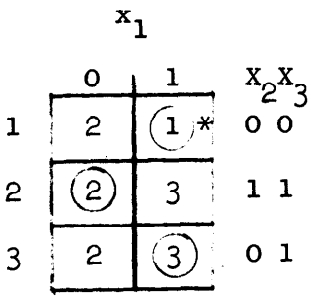
Table 1. Alternative Flow Tables for the Parallel System of Fig. 2



a) Component 1



b) Component 2



c) Component 3

* Never entered during the operation of the component.

Definition 4:

Let a and b be internal states of component C_i for which output variable X_k has values 0 and 1, respectively. An output 0 hazard is the possibility of a transition from a to b when input variable x_k has the value 1.

Definition 5:

A component C_i is said to be connected to a component C_j if

1. There is a line from an output of C_i to an input of C_j, or
2. There is a line from an output of C_i to some component C_k and C_k is connected to C_j.

Theorem 1: (Sufficient conditions for an output 1 hazard)

If component C_i changes output variable X_k from 0 to 1 to 0 and component C_j with input variable x_k is not connected to C_i, an output 1 hazard exists.

Proof:

By Assumption 1, there is no bound on the time for the 1 value to propagate to x_k. Since C_j is not connected to C_i, if C_i ever changes X_k from 1 to 0 it is possible that the 1 value has not arrived at x_k. The conditions of Definition 3 are satisfied and an output 1 hazard exists.

Theorem 2: (Sufficient conditions for an output 0 hazard)

If component C_i changes output variable X_k from 1 to 0 to 1 and component C_j with input variable x_k is not connected to C_i, an output 0 hazard exists.

Proof:

Similar to the proof of Theorem 1.

These two theorems, while trivial, aid somewhat in understanding the conditions in which output hazards will always exist. Of greater interest are theorems giving necessary and sufficient conditions for the absence of output hazards.

Theorem 3: (Necessary and sufficient conditions for the absence of an output 1 hazard)

Let component C_i change the value of output variable X_k from 0 to 1. There is no output 1 hazard when C_i changes X_k from 1 to 0 if and only if the value of X_k is changed to 0 only after C_i recognizes an input value produced in recognition of the 1 value for X_k.

Proof:

(Sufficiency)

Since C_i does not change X_k to 0 until it recognizes an input value produced in recognition of the 1 value for X_k, the 1 value for X_k must have propagated to a component input and x_k must have the value 1. Thus there is no output 1 hazard.

(Necessity)

We prove the contrapositive. Let C_i change X_k to 0 without first recognizing an input value produced in recognition of the 1 value for X_k. Since the line delays are unbounded (Assumption 1), there can be no guarantee that the 1 value has propagated to the

component input and it is possible that x_k is 0 when C_i changes X_k. By Definition 3, an output 1 hazard exists.

Theorem 4: (Necessary and sufficient conditions for the absence of an output 0 hazard)

Let component C_i change the value of output variable X_k from 1 to 0. There is no output 0 hazard when C_i changes X_k from 0 to 1 if and only if the value of X_k is changed to 1 only after C_i recognizes an input value produced in recognition of the 0 value for X_k.

Proof:

Similar to the proof of Theorem 3.

Suppose there are no output hazards in a parallel system. That is, every output value produced must propagate to the associated component input. Since the operation of each component is described by a completely specified flow table, each component must have a stable state in every column of the flow table and in particular every column in which the new input value appears. By Assumption 2, the component will recognize the new value in a finite time, if the new value remains present. Consider the possibility that a new input value must appear but that it is not present long enough to be recognized by the component. By Assumption 1, the time for the input value change to reach the component is arbitrary. Therefore, if it is possible for the new value to appear and not be recognized, it must be possible for the new value not to have appeared at all. That is, an output hazard

must exist. But this contradicts our assumption that no output hazards existed and hence it must be impossible for a new input value to appear but not be recognized. The result of this argument is the following theorem.

Theorem 5:

If a parallel system has no output hazards, every output change produces an input value which must be recognized.

The absence of output hazards is not only sufficient to ensure that all input changes are recognized, it is necessary as well.

Theorem 6:

If every output change produces an input value which must be recognized, the parallel system has no output hazards.

Proof:

This proof also follows from the fact that line delays are of arbitrary duration. Suppose a parallel system has an output 1 hazard in line $\ell_k = (X_k, x_k)$ which joins component C_i to component C_j. By Definition 3, it is possible for C_i to change X_k from 1 to 0 when x_k has the value 0. But by Assumption 1 line delays are arbitrary and so it is also possible that x_k will momentarily have the value 1. There is no nonzero lower bound on the duration of the 1 value for x_k. Therefore, regardless of the interval between the times C_j examines its inputs, the 1 value for x_k may not be recognized. The possibility of an output 0 hazard

can similarly be shown to always provide the possibility of a 0 input which is not recognized and the proof is complete.

In general there may be many connections from the component which recognizes a new input value back to the component which produced it. Furthermore these connections may be through many other components and require many more hazard-free interactions.

DESCRIPTION OF SYSTEM OPERATION

Given a parallel system, we now give a procedure which detects all output hazards present in the system and in the case when no such hazards exist, produces a description of the operation of the system. This description will be a directed graph. If output hazards are present, the behavior of the system cannot be reliably predicted. That is, momentary 1 output values may or may not result in 1 input values and if 1 input values appear these values may or may not be recognized. In such cases, it is best to modify the system to eliminate these hazards before producing a graph description of system operation.

Definition 6:

The component state or total component state is defined by the component internal state and component input state.

The initial internal state of the control component in Fig. 1 is 1 and the initial input state is 00. The initial component state is written 1-00.

Because the output state of each component is determined by the component internal state, it is unnecessary to include the output state in the definition of total component state. When a component is in an internal state, it is assumed that the output variables have the values designated for that state. If there is objection to this assumption, we can define being "in an internal state" as the instant when the output variables attain the values specified for the internal state.

Definition 7:

The system state or total system state is defined by the N-tuple consisting of the component states for each of the N components in a parallel system.

The initial system state for the parallel system in Fig. 2 is written (1-0,1-0,1-0). As a consequence of the definition of a parallel system, the initial system state is unique.

A component is stable if the flow table entry for the present total component state is the same as the present internal state; otherwise a component is unstable. A line is stable if the value at every point in the line is the same; otherwise the line is unstable. A sufficient but not necessary condition for a line $\ell_k = (X_k, x_k)$ to be unstable is that X_k and x_k have different values.

Definition 8:

Given two system states A and B (A≠B), B is said to be an immediate successor of A if the following conditions are satisfied:

1. If a component input has a value in B different from the value in A, the line was unstable in A.
2. If a component internal state in B is different from the corresponding internal state in A, the component was unstable in A and the B value is the next state entry in the component flow table as determined by the total component state in A.

The initial system state of the parallel system for the two-process mutual exclusion problem in Fig. 1 is (1-0,1-0,1-00). In this state, all lines are stable but two components, C_1 and C_2, are unstable. There are three immediate successor states to this system state.

(2-0,1-0,1-00)

(1-0,2-0,1-00)

(2-0,2-0,1-00)

In system state (2-0,1-0,1-00), line $\ell_1 = (X_1, x_1)$ and component C_2 are unstable. The immediate successors are

(2-0,2-0,1-00)

(2-0,1-0,1-10)

(2-0,2-0,1-10)

In general, if p lines and components are unstable in a given system state, there are 2^p-1 immediate successor states.

Definition 9:

The system state graph is a directed graph the nodes of which are the system states attained during the operation of the system. Two system states A and B are joined by a directed arc from A to B if and only if B is an immediate successor of A.

Definition 10:

Given two system states A and B, B is said to be a successor of A if

1. B is an immediate successor of A, or
2. B is a successor of an immediate successor of A.

We will use the system state graph to describe the operation of a parallel system. The procedure for obtaining this graph is as follows:

Procedure for Finding System State Graph

1. Given the initial system state, determine all immediate successor states.
2. Consider each immediate successor state as a system state and determine all immediate successors.
3. The procedure terminates when all immediate successors of all immediate successors introduced in steps 1 and 2 have been determined.
4. (Check for output hazards)

   If a system state A has an immediate successor B such that internal state a in A for some component C_i and internal state b in B for

the same component satisfy the conditions in Definition 3 or 4, an output hazard exists.

This procedure considers all possible state transitions for a parallel system; therefore, all output hazards are detected.

An Example: The Buffer Problem

To illustrate the application of the analysis method, consider a parallel system with two components, C_1 and C_2. Component C_1 represents a Central Processing Unit (CPU) which fills a buffer. The buffer is emptied by component C_2, a Channel or Output Controller. Initially the buffer is empty and the Channel and CPU are idle. The problem posed for this system, which we call the buffer problem, is the following:

Problem: (The Buffer Problem)

Design a system with a CPU and Channel, operating as described above, such that at all times during the operation of the system the following restrictions are satisfied:

Restriction 3*:

The CPU and the Channel never access the buffer simultaneously.

Restriction 4:

When the CPU has filled the buffer, the CPU must not access the buffer again until after the buffer has been emptied by the channel.

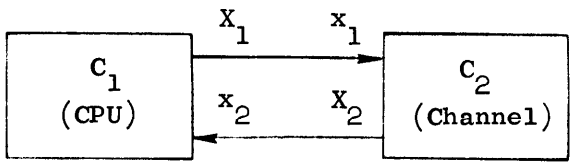
Restriction 5:

When the buffer has been emptied, the Channel must not access the buffer again until the buffer has been filled by the CPU.

* Restrictions 1 and 2 are associated with the mutual exclusion problem.

This problem is similar to the mutual exclusion problem in that the CPU and Channel must not access the buffer simultaneously. It differs in that the order of accessing the buffer is fixed. It is not necessary to specify the buffer as a component of the system in order to study the control interactions of the CPU and the Channel.

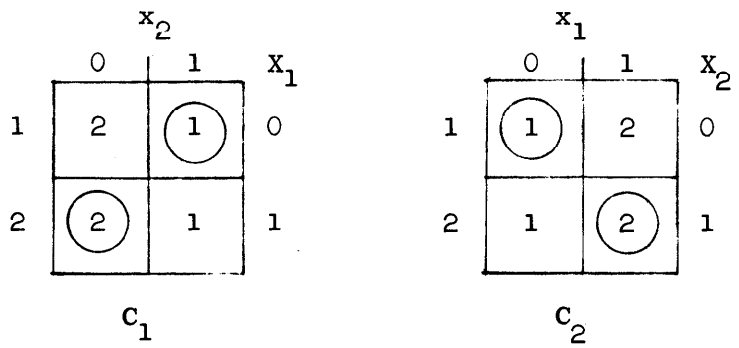
Consider the parallel system specified in Fig. 3. The interpretations of the values on the interconnecting lines and the component states are as follows. Initially, the CPU, C_1, is in total component state 1-0. In this state, the CPU is unstable and the buffer is filled. Eventually the buffer becomes full and C_1 enters internal state 2, a stable state, and sets X_1 to 1. The 1 input to C_2 causes C_2 to become unstable and empty the buffer (component state 1-1). When the buffer is empty, C_2 enters internal state 2, setting X_2 to 1. This output value propagates to C_1 notifying C_1 that the buffer has been emptied and putting C_1 in component state 2-1. In this unstable state, the CPU fills the buffer again. When the buffer is full, the CPU enters internal state 1, setting X_2 to 0. When x_2 becomes 0, the cycle begins again. Notice the significance of the value transitions during the operation of this system. The first time the CPU fills the buffer, it uses a 0 to 1 transition on line (X_1,x_1) to notify the Channel. The second time, the CPU uses a 1 to 0 transition to notify the Channel. The Channel also alternates, first using a 0 to 1 transition on line (X_2,x_2) to indicate that the buffer has been emptied and the next time using a 1 to 0 transition. It is possible to give an interpretation for this system in which a 0 to 1 transition on line



$\mathcal{C} = \{C_1, C_2\}$ $\qquad$ $\mathcal{L} = \{\ell_1, \ell_2\}$

$O_1 = \{X_1\}$ $\qquad$ $I_1 = \{x_2\}$

$O_2 = \{X_2\}$ $\qquad$ $I_2 = \{x_1\}$



(initial system state is (1-0,1-0))

Figure 3. Parallel system for the buffer problem.

(X_1, x_1) is always used to indicate that the buffer is full and a 0 to 1 transition on (X_2,x_2) indicates that the buffer has been emptied.

Programs of the class described in [3] representing the CPU and Channel activity are given in Table 2. An analysis of these programs using the procedures given in [3] will verify that their operation is characterized by the flow tables shown in Fig. 3.

Let us now apply the procedure described earlier to obtain the system state graph for the parallel system of Fig. 3. A table of the system states and their immediate successors as produced by the procedure appears in Table 3. The initial system state is (1-0,1-0) and the only immediate successor is (2-0,1-0). This state has only one immediate successor (2-0,1-1). The procedures continues and terminates when eight system states have been produced. This system may not appear to exhibit much parallelism but remember that it is possible for the CPU to perform other functions after it fills the buffer and while the buffer is being emptied as long as it does not try to add additional information to the buffer. The system graph is shown in Fig. 4. Since there are no output hazards, every output change results in a new input value which must be recognized (Theorem 5). Therefore, line delays cannot affect the operation of this system. If the line delays are set to zero, the system state graph in Fig. 4 can be reduced to the system state graph shown in Fig. 5.

Detection of an Output Hazard

To illustrate the application of the analysis procedure to a system containing an output hazard, we apply the procedure to the parallel system given in Fig. 2. The initial system state is (1-0,1-0,1-0). This state has one immediate successor (1-0,1-0,2-0).

Table 2. CPU and channel Programs for the Parallel System of Fig. 3.

```
    INPUT X2
    OUTPUT X1      (initially X1 is 0)
1:  DUMMY; (fill buffer)
2:  X1: = 1;
3:  WAIT (1, 4);
4:  DUMMY; (fill buffer)
5:  X1: = 0;
6:  WAIT (0, 1).
```

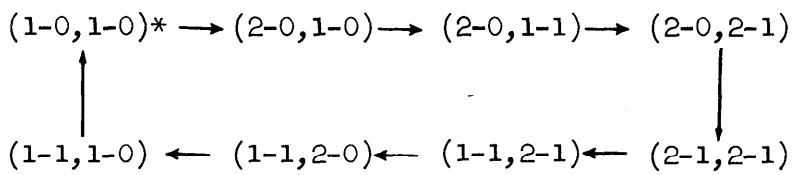
a) CPU

```
    INPUT X1
    OUTPUT X2      (initially X2 is 0)
1:  WAIT (1, 2);
2:  DUMMY; (empty buffer)
3:  X2: = 1;
4:  WAIT (0, 5);
5:  DUMMY; (empty buffer)
6:  X2: = 0;
7:  GO TO 1.
```

b) Channel

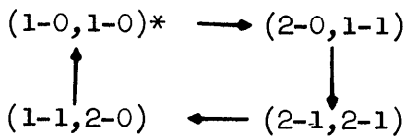
Table 3. System States With Immediate Successors for the Parallel System of Fig. 3.

	system state	immediate sucessors
(initial state)	(1-0,1-0)	(2-0,1-0)
	(2-0,1-0)	(2-0,1-1)
	(2-0,1-1)	(2-0,2-1)
	(2-0,2-1)	(2-1,2-1)
	(2-1,2-1)	(1-1,2-1)
	(1-1,2-1)	(1-1,2-0)
	(1-1,2-0)	(1-1,1-0)
	(1-1,1-0)	(1-0,1-0)



* initial system state

Figure 4. System state graph for the parallel system in Fig. 3.



* initial system state

Figure 5. System state graph for the parallel system in Fig. 3 when line delays are zero.

A partial table of system states with immediate successors is shown in Table 4. The hazard is detected in determining the immediate successors of state (2-1,1-0,2-1). It is possible for C_3 to change from component state 2-1 to 1-1 which changes X_3 from 1 to 0. But x_3 has the value 0 and therefore an output 1 hazard exists.

System State Graph for the Two-Process Mutual Exclusion Problem

Table 5 contains the 64 system states with immediate successors which are attained during the operation of the parallel system for the two-process mutual exclusion problem defined in Fig. 1. The system state graph is given in Fig. 6. If line delays are zero, the graph in Fig. 6 can be reduced to the graph shown in Fig. 7.

CORRECT OPERATION FOR PARALLEL SYSTEMS

We now consider what it means to say that a parallel system operates correctly. The systems we have discussed have been designed to solve particular problems such as the mutual exclusion problem or the buffer problem. A problem is a word statement with a number of restrictions which must be met if the problem is to be solved. We say that a parallel system is correct with respect to a given problem if the system operates so that the restrictions are always satisfied. This notion differs from the idea of correctness for a computation where correctness usually means that the computation halts and gives the desired answer [1, 1a, 8, 12, 18, 19, 20]. Since the systems we deal with do not halt under ordinary circumstances, we have found it necessary to formulate a different interpretation of correct operation.

Table 4. Partial Table of System States With Immediate Successors for the Parallel System in Fig. 2.

system state	immediate successors	
(initial state) (1-0,1-0,1-0)	(1-0,1-0,2-0)	
(1-0,1-0,2-0)	(1-1,1-0,2-0)	
	(1-0,1-1,2-0)	
	(1-1,1-1,2-0)	
(1-1,1-0,2-0)	(2-1,1-0,2-0)	
	(1-1,1-1,2-0)	
	(2-1,1-1,2-0)	
(2-1,1-0,2-0)	(2-1,1-0,2-1)	
	(2-1,1-1,2-0)	
	(2-1,1-1,2-1)	
(2-1,1-0,2-1)	(2-1,1-0,1-1)	(output hazard detected)

Table 5. System States With Immediate Successors for the Parallel System of Fig. 1

system states	immediate successors
(1-0,1-0,1-00)	(2-1,1-0,1-00)
	(1-0,2-0,1-00)
	(2-0,2-0,1-00)
(2-0,1-0,1-00)	(2-0,2-0,1-00)
	(2-0,1-0,1-10)
	(2-0,2-0,1-10)
(1-0,2-0,1-00)	(2-0,2-0,1-00)
	(1-0,2-0,1-01)
	(2-0,2-0,1-01)
(2-0,2-0,1-00)	(2-0,2-0,1-10)
	(2-0,2-0,1-01)
	(2-0,2-0,1-11)
(2-0,1-0,1-10)	(2-0,2-0,1-10)
	(2-0,1-0,3-10)
	(2-0,2-0,3-10)
(2-0,2-0,1-10)	(2-0,2-0,3-10)
	(2-0,2-0,1-11)
	(2-0,2-0,3-11)
(1-0,2-0,1-01)	(2-0,2-0,1-01)
	(1-0,2-0,2-01)
	(2-0,2-0,2-01)

Table 5 (cont)

(2-0,2-0,1-01)	(2-0,2-0,1-11)
	(2-0,2-0,2-01)
	(2-0,2-0,2-11)
(2-0,2-0,1-11)	(2-0,2-0,3-11)
(2-0,1-0,3-10)	(2-0,2-0,3-10)
	(2-1,1-0,3-10)
	(2-1,2-0,3-10)
(2-0,2-0,3-10)	(2-0,2-0,3-11)
	(2-1,2-0,3-10)
	(2-1,2-0,3-11)
(2-0,2-0,3-11)	(2-1,2-0,3-11)
(1-0,2-0,2-01)	(2-0,2-0,2-01)
	(1-0,2-1,2-01)
	(2-0,2-1,2-01)
(2-0,2-0,2-01)	(2-0,2-0,2-11)
	(2-0,2-1,2-01)
	(2-0,2-1,2-11)
(2-0,2-0,2-11)	(2-0,2-1,2-11)
(2-1,1-0,2-10)	(1-1,1-0,3-10)
	(2-1,2-0,3-10)
	(1-1,2-0,3-10)
(2-1,2-0,3-10)	(1-1,2-0,3-10)
	(2-1,2-0,3-11)
	(1-1,2-0,3-11)

Table 5 (cont)

(2-1,2-0,3-11)	(1-1,2-0,3-11)
(2-0,2-1,2-01)	(2-0,2-1,2-11)
	(2-0,1-1,2-01)
	(2-0,1-1,2-11)
(1-0,2-1,2-01)	(2-0,2-1,2-01)
	(1-0,1-1,2-01)
	(2-0,1-1,2-01)
(2-0,2-1,2-11)	(2-0,1-1,2-11)
(1-1,1-0,3-10)	(1-1,2-0,3-10)
	(1-1,1-0,3-00)
	(1-1,2-0,3-00)
(1-1,2-0,3-10)	(1-1,2-0,3-11)
	(1-1,2-0,3-00)
	(1-1,2-0,3-01)
(1-1,2-0,3-11)	(1-1,2-0,3-01)
(2-0,1-1,2-01)	(2-0,1-1,2-11)
	(2-0,1-1,2-00)
	(2-0,1-1,2-10)
(2-0,1-1,2-11)	(2-0,1-1,2-10)
(1-0,1-1,2-01)	(2-0,1-1,2-01)
	(1-0,1-1,2-00)
	(2-0,1-1,2-00)
(1-1,1-0,3-00)	(1-1,1-0,4-00)
	(1-1,2-0,3-00)
	(1-1,2-0,4-00)

Table 5 (cont)

(1-1,2-0,3-00)	(1-1,2-0,4-00)
	(1-1,2-0,3-01)
	(1-1,2-0,4-01)
(1-1,2-0,3-01)	(1-1,2-0,2-01)
(2-0,1-1,2-00)	(2-0,1-1,1-00)
	(2-0,1-1,2-10)
	(2-0,1-1,1-10)
(2-0,1-1,2-10)	(2-0,1-1,3-10)
(1-0,1-1,2-00)	(2-0,1-1,2-00)
	(1-0,1-1,1-00)
	(2-0,1-1,1-00)
(1-1,1-0,4-00)	(1-0,1-0,4-00)
	(1-1,2-0,4-00)
	(1-0,2-0,4-00)
(1-1,2-0,4-00)	(1-0,2-0,4-00)
	(1-1,2-0,4-01)
	(1-0,2-0,4-01)
(1-1,2-0,4-01)	(1-0,2-0,4-01)
	(1-1,2-0,2-01)
	(1-0,2-0,2-01)
(1-1,2-0,2-01)	(1-0,2-0,2-01)
	(1-1,2-1,2-01)
	(1-0,2-1,2-01)

Table 5 (cont)

(2-0,1-1,1-00)	(2-0,1-1,1-10)
	(2-0,1-0,1-00)
	(2-0,1-0,1-10)
(2-0,1-1,1-10)	(2-0,1-0,1-10)
	(2-0,1-1,3-10)
	(2-0,1-0,3-10)
(1-0,1-1,1-00)	(2-0,1-1,1-00)
	(1-0,1-0,1-00)
	(2-0,1-0,1-00)
(1-0,1-0,4-00)	(2-0,1-0,4-00)
	(1-0,2-0,4-00)
	(2-0,2-0,4-00)
(1-0,2-0,4-00)	(2-0,2-0,4-00)
	(1-0,2-0,4-01)
	(2-0,2-0,4-01)
(1-0,2-0,4-01)	(2-0,2-0,4-01)
	(1-0,2-0,2-01)
	(2-0,2-0,2-01)
(1-1,2-1,2-01)	(1-0,2-1,2-01)
	(1-1,1-1,2-01)
	(1-0,1-1,2-01)
(2-0,1-1,3-10)	(2-1,1-1,3-10)
	(2-0,1-0,3-10)
	(2-1,1-0,3-10)

Table 5 (cont)

(2-0,1-0,4-00)	(2-0,2-0,4-00)
	(2-0,1-0,4-10)
	(2-0,2-0,4-10)
(2-0,2-0,4-00)	(2-0,2-0,4-10)
	(2-0,2-0,4-01)
	(2-0,2-0,4-11)
(2-0,2-0,4-01)	(2-0,2-0,4-11)
	(2-0,2-0,2-01)
	(2-0,2-0,2-11)
(1-1,1-1,2-01)	(1-0,1-1,2-01)
	(1-1,1-1,2-00)
	(1-0,1-1,2-00)
(2-1,1-1,3-10)	(1-1,1-1,3-10)
	(2-1,1-0,3-10)
	(1-1,1-0,3-10)
(2-0,1-0,4-10)	(2-0,2-0,4-10)
	(2-0,1-0,3-10)
	(2-0,2-0,3-10)
(2-0,2-0,4-10)	(2-0,2-0,3-10)
(2-0,2-0,4-11)	(2-0,2-0,2-11)
(1-1,1-1,2-00)	(1-0,1-1,2-00)
	(1-1,1-1,1-00)
	(1-0,1-1,1-00)

Table 5 (cont)

(1-1,1-1,3-10)	(1-1,1-0,3-10)
	(1-1,1-1,3-00)
	(1-1,1-0,3-00)
(1-1,1-1,1-00)	(1-0,1-1,1-00)
	(1-1,1-0,1-00)
	(1-0,1-0,1-00)
(1-1,1-1,3-00)	(1-1,1-0,3-00)
	(1-1,1-1,4-00)
	(1-1,1-0,4-00)
(1-1,1-0,1-00)	(1-0,1-0,1-00)
	(1-1,2-0,1-00)
	(1-0,2-0,1-00)
(1-1,1-1,4-00)	(1-0,1-1,4-00)
	(1-1,1-0,4-00)
	(1-0,1-0,4-00)
(1-1,2-0,1-00)	(1-0,2-0,1-00)
	(1-1,2-0,1-01)
	(1-0,2-0,1-01)
(1-0,1-1,4-00)	(2-0,1-1,4-00)
	(1-0,1-0,4-00)
	(2-0,1-0,4-00)
(1-1,2-0,1-01)	(1-0,2-0,1-01)
	(1-1,2-0,2-01)
	(1-0,2-0,2-01)

Table 5 (cont)

(2-0,1-1,4-00)	(2-0,1-0,4-00)
	(2-0,1-1,4-10)
	(2-0,1-0,4-10)
(2-0,1-1,4-10)	(2-0,1-0,4-10)
	(2-0,1-1,3-10)
	(2-0,1-0,3-10)

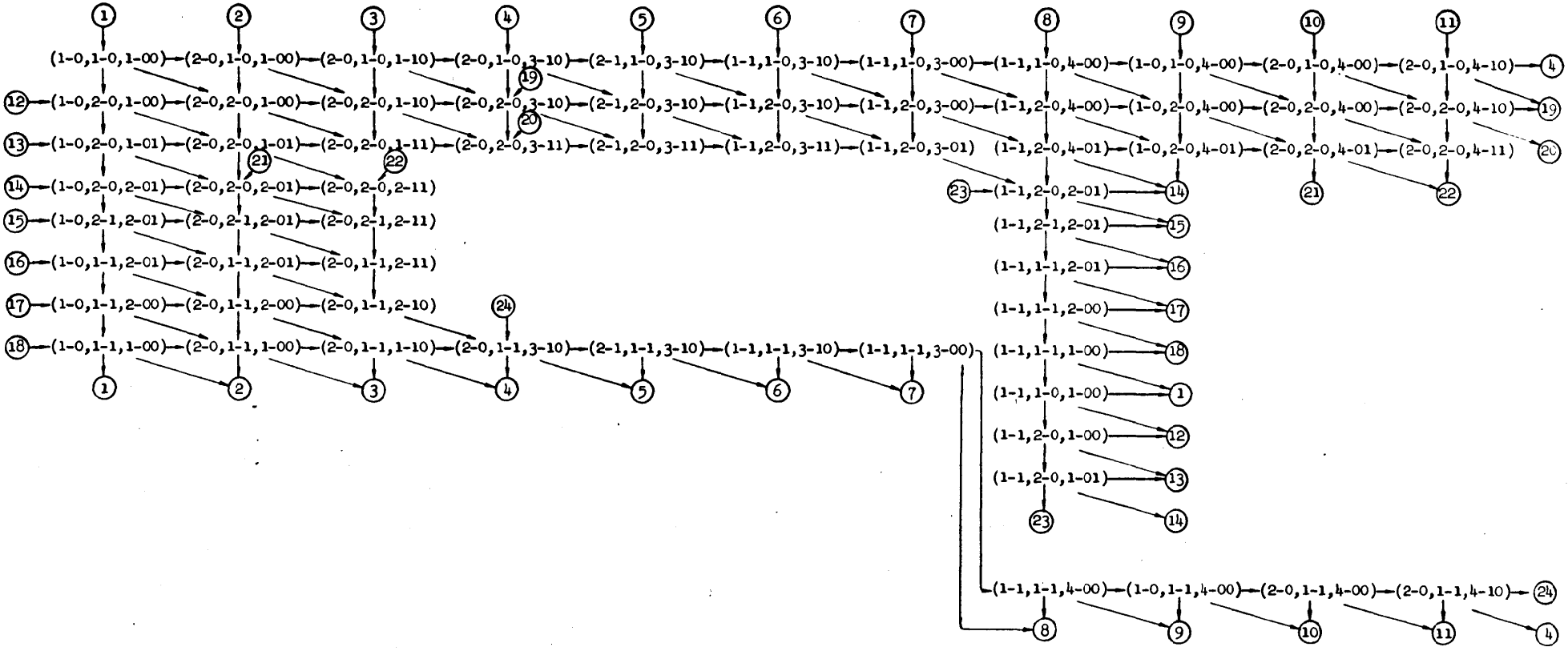
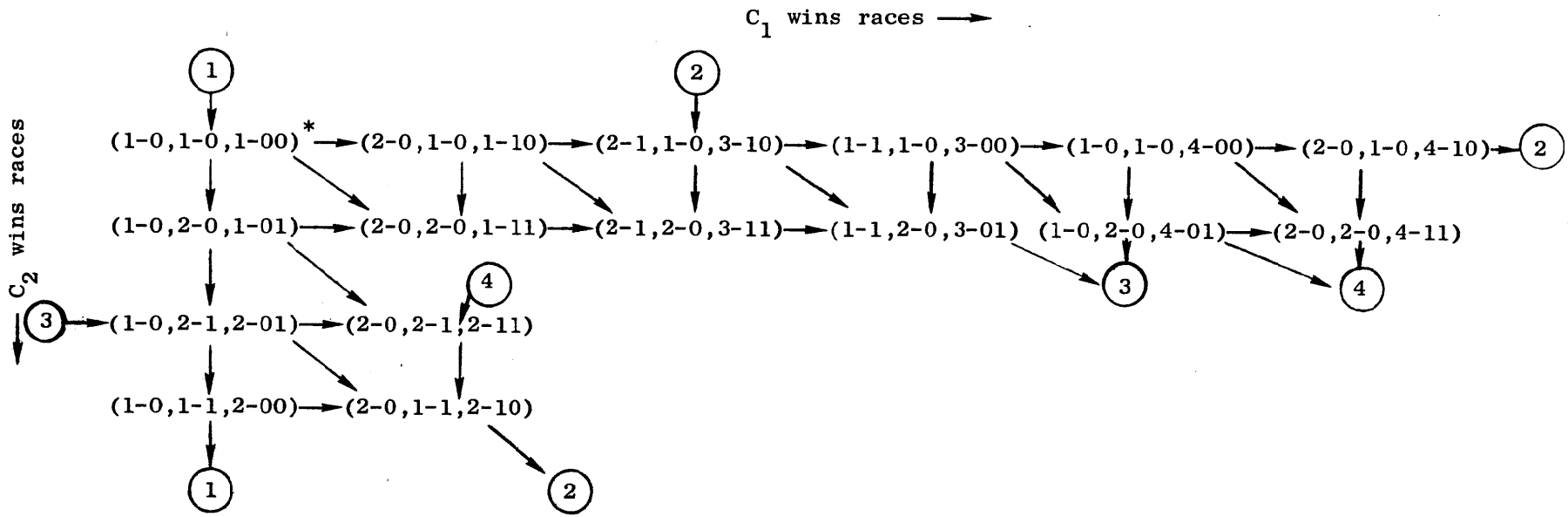


Figure 6. System state graph for the two-process mutual exclusion problem.



*initial system state

Figure 7. System state graph for the parallel system in Fig. 1 when line delays are zero.

Correctness of the Solution to the Buffer Problem

Consider the parallel system shown in Fig. 3 proposed as a solution to the buffer problem. Restriction 3 states that the CPU and the Channel must never access the buffer simultaneously. The system states which represent simultaneous access by the CPU and Channel are (1-0,1-1), (1-0,2-0), (2-1,1-1), (2-1,2-0). The system state graph in Fig. 4, or the list of system states in Table 3, specifies all system states attained during the operation of the system. Since none of the above system states appear, it is impossible for the system to violate Restriction 3. Consider Restriction 4. When the buffer has been filled, the CPU enters either component state 2-0 or 1-1. To determine if the restriction is always satisfied, we must examine paths in the system state graph. A path is a sequence of system states, $s_0, s_1, \ldots, s_i, s_{i+1}, \ldots$, such that state s_{i+1} is an immediate successor of state s_i for $i = 0,1,\ldots$. To verify that Restriction 4 is satisfied, we examine the system state graph and determine that each time the CPU enters component state 2-0 (in system state (2-0,1-0)) or component state 1-1 (in system state (1-1,2-1)) that in every path from these system states, the Channel enters component state 1-1 or 2-0 before the CPU enters component state 1-0 or 2-1. The state graph shows that this is the case and consequently Restriction 4 is satisfied. The verification of Restriction 5 follows in a similar way. Each time the Channel enters component state 1-0 (in system state (1-1, 1-0)) or 2-1 (in system state (2-0,2-1)), the CPU enters states 1-0 or 2-1 before the Channel enters 1-1 or 2-0.

Since Restrictions 3, 4, and 5 are satisfied, we conclude that the parallel system of Fig. 3 is correct with respect to the buffer problem.

In this analysis of the buffer problem solution, two types of tests were applied. The first required the examination of the system state graph to determine if certain "forbidden states" were entered. The second type of test consisted of an examination of the system state graph for the possibility of "forbidden paths". These two types of tests are also used in the analysis of the parallel system for the two-process mutual exclusion problem.

Correctness of the Solution to the Two-Process Mutual Exclusion Problem

The system state graph for this system was given in Fig. 6. The restrictions for this problem are Restrictions 1 and 2. The first restriction states that at most one component may be in a critical section at a time. In terms of our parallel system, this means that system states of the form (2-1, 2-1,*) must never occur. The asterisk indicates that any component state is acceptable for component C_3. An examination of the system state graph, or the table of states in Table 5, shows that no states with this form are ever entered; therefore, Restriction 1 is satisfied. Restriction 2 states that whenever a component, say C_1, wants to enter its critical section (C_1 is in component state 2-0) that C_1 must eventually enter (the critical section) component state 2-1. Thus whenever the system enters a system state of the form (2-0,*,*-1*), that is, component C_1 has requested access to its critical section and the request has

propagated to the control input, every path with such a state as its initial state must contain a state of the form (2-1,*,*). There are 16 system states which have the form (2-0,*,*-1*). Every path from these states does contain a state of the form (2-1,*,*). Therefore, when C_1 requests access to its critical section, it must gain access. The argument that C_2 must gain access to its critical section when it desires follows in a similar manner. On the basis of these arguments, we say that the parallel system of Fig. 1 is correct with respect to the mutual exclusion problem.

Correctness, Determinacy, and Output Functionality

In this section, we contrast our notion of correct operation with the notions of determinate, completely functional, and output functional systems which have been proposed by Adams, Karp and Miller, Luconi, Rogriquez, and others [1,9,10,11,15,16,17,23,24,25]. In all of these systems or models, the operation of a component is specified by a function rather than by a flow table. Our flow tables can be represented as functions if the component internal state is added as an input and an output. The operation of the component is then described by a function which maps the original input state and internal state into the output state and next internal state. The definition of such functions for the mutual exclusion problem flow tables of Fig. 1 is given in Table 6. These functions are partial functions which are defined only when the original flow tables are unstable. Components become ready for execution when their functions become defined. In the Karp and Miller model [10, 11] there is an additional

Table 6. Functional Description of Component Operation for the Parallel System of Fig. 1

f_1: $z_1 I_1 \longrightarrow X_1 I_1$

0 1	1 2
1 2	0 1

a) C_1

f_2: $z_2 I_2 \longrightarrow X_2 I_2$

0 1	1 2
1 2	0 1

b) C_2

f_3: $x_1 x_2 I_3 \longrightarrow X_3 X_4 I_3$

0 1 1	0 1 2
1 1 1	1 0 3
1 0 1	1 0 3
0 0 2	0 0 1
1 0 2	1 0 3
0 0 3	0 0 4
0 1 3	0 1 2
0 1 4	0 1 2
1 1 4	0 1 2
1 0 4	1 0 3

c) C_3 (control)

portion of the system called the control which is used to regulate when functions may be executed. The effect would be essentially the same in this example however. A system is said to be determinate or completely functional if the history or sequence of values associated with each variable in the system is unique. We now show that the parallel system we have given is not determinate in this sense. To simplify the discussion, assume that all line delays are zero. That is, when a component changes its output value, the value immediately propagates to the input at the other end of the line. The initial component state is 1-0 for both C_1 and C_2. Thus f_1 and f_2 are both defined. The initial component state for C_3 is 1-00 and f_3 is undefined. Suppose f_2 is executed. The input to f_3 becomes 011 and now f_2 and f_3 are defined. If f_3 is executed, the value of the internal state for C_3 will be 2. However, if f_2 is executed and then f_3 is executed, the value of the internal state of C_3 will be 3. This means that the sequence of values associated with the internal state of C_3 is dependent on the speed of operation of the components in the system and therefore the system is not determinate or completely functional. This system violates one of the conditions which have been shown to be sufficient to guarantee determinate (completely functional) operation. Therefore, the lack of determinacy is not too surprising. The condition states that if two functions are simultaneously defined, the execution of one of the functions cannot affect the values to be produced by the other function. Clearly, the execution of f_2 affects

the value of the internal state of C_3 produced when f_3 is executed. This situation is another example of a transformation-loss [15] or a violation of semi-modularity [23]. The model of Karp and Miller deals with parallel program schemata which are parallel systems in which the interconnections are known but the functional behavior of each component is not. Determinacy in this model means that for every possible assignment of functions for the components in the system (every possible interpretation) the sequence of values associated with each variable must be unique. In such a model, sufficient conditions for determinacy require that if two functions can be executed in parallel, the intersection of the input set of each component with the output set of the other must be empty. This condition is also violated by f_2 and f_3 since the intersection of the output set of f_2 $\{X_2, I_2\}$ with the input set of f_3 $\{x_1, x_2, I_3\}$ contains the common line (X_2, x_2).

Luconi and Van Horn [25] were aware of the possibility that certain variables, perhaps not essential as far as the correct operation of the system was concerned, might not have unique value sequences associated with them. To allow for this possibility, they introduced the notion of output functional systems or systems in which unique value sequences were required only for some designated subset of the set of variables in the system. Let us require that only the inputs (and outputs) of each component have unique value sequences and allow the sequences of values associated with internal states to be non-unique. All input and output values are initially 0. An examination of the system state graph in Fig. 6 shows that the sequence

of values for all inputs and outputs are the same and have the form (0,1,0,1,0,1,...). That is, the values are alternately 0 and 1 for as long as the system exists.

We now consider whether there is any relation between our notion of a correct system and an output functional system. We show, by presenting two examples, that the notions are not related. In the first example, we give a system which has the same configuration as the parallel system for the mutual exclusion problem and which has the same value sequences associated with the input and output variables (0,1,0,1,0,1,0,1,...) as the correct system but which is not correct in the sense that it fails to solve the mutual exclusion problem. Second, we give a system which correctly solves the mutual exclusion problem but which is not output functional.

To obtain the first system use the same parallel system as before but replace the control (C_3) flow table shown in Fig. 1 by the flow table shown in Table 7. With this control mechanism, if one process is enabled and the other process asks to be enabled, the control enables both processes. This allows both processes to be in critical sections simultaneously, violating Restriction 1. If the system state graph were constructed for this system, it would show that the input and output variables still have the same value sequences as before (0,1,0,1,0,1,...). The difficulty with output functionality in this example is the fact that the state of the system has been neglected.

The second example is also obtained by starting with the parallel system for the mutual exclusion problem. This time we modify the flow

Table 7. Modified Control Flow Table For the Mutual Exclusion Problem

	x_1x_2 00	01	11	10	X_3X_4
1	(1)	2	4	3	00
2	1	(2)	4	3	01
3	1	2	4	(3)	10
4	1	2	(4)	3	11

a) Flow Table (C_3)

f_3: $x_1x_2I_3$	$\longrightarrow$ $X_3X_4I_3$
0 1 1	0 1 2
1 1 1	1 1 4
1 0 1	1 0 3
0 0 2	0 0 1
1 1 2	1 1 4
1 0 2	1 0 3
0 0 3	0 0 1
0 1 3	0 1 2
1 1 3	1 1 4
0 0 4	0 0 1
0 1 4	0 1 2
1 0 4	1 0 3

b) Function

table for C_2 so that the component asks to enter its critical section only once and when it leaves the critical section it halts. A possible flow table for C_2 is shown in Table 8. Initially, the component is in component state 1-0. This modification alone leaves a system which is correct with respect to the mutual exclusion problem and which is still output functional. That is X_4, x_4, X_2 and x_2 have the finite value sequence (0,1,0) and X_3, x_3, X_1, and x_1 have the value sequence (0,1,0,1,0,1,...). Suppose we add a connection from component C_2 to component C_1 so that when C_2 exits its critical section it notifies C_1 which then also halts. The resulting system is still correct with respect to the mutual exclusion problem (we omit the details but the control is still the same). Now, however, there are many different possible value sequences for X_1, x_1, X_3, and x_3 and the exact sequence depends on the speed of operation of C_2.

These examples show that parallel systems which are correct with respect to the mutual exclusion problem need not be output functional. While the examples may be somewhat contrived, we believe they cast doubt on the use of output functional or determinate operation as a design criterion for parallel systems. A model in which the speed of component operation has no affect is not always desirable. It is possible for a system to take alternative actions precisely as a result of the time necessary to perform certain tasks. For example, a system might perform a computation two different ways in parallel and use the result that is available first.

Table 8. Modified Flow Table for C_2 (Fig. 1) to Allow Entering Critical Section Only Once

	z_2 = 0	z_2 = 1	x_2
1	2	(1)	0
2	(2)	3	1
3	(3)	(3)	0

CONCLUSIONS

We have described an analysis procedure for parallel systems. A block diagram summarizing the analysis phases is given in Fig. 8. The analysis procedure can be applied to any system defined as in Definition 1 if appropriate restrictions on system operation can be stated. More efficient analysis procedures are necessary if total system analysis is to be practical. In a future paper, we consider a more problem-oriented approach to system analysis.

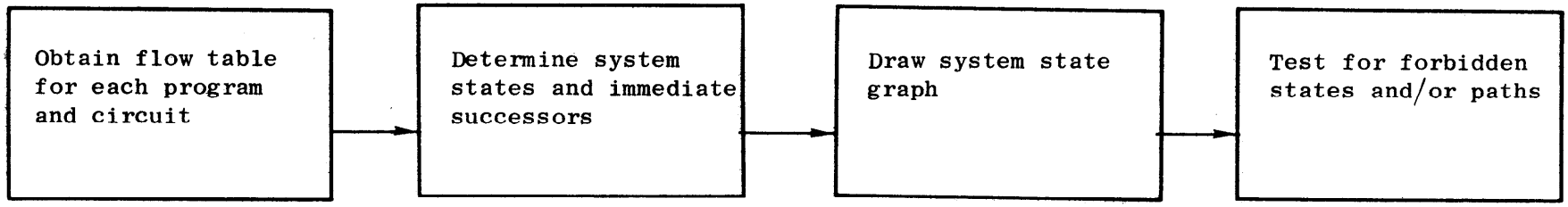


Figure 8. Analysis procedure for parallel systems.

REFERENCES

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

[1a] Ashcroft, E. and Manna, Z. Formalization of properties of parallel programs. Memo No. AIM-110, Stanford Artifical Intelligence Project, Stanford University, Stanford, California (Feb 1970).

[2] 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).

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

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

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

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

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

[8] Floyd, R.W. Assigning meanings to programs. Proc. of Symposium on Applied Mathematics, Vol. 19, American Mathematical Society (1967), 19-32.

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

[10] 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.

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

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

[13] Knuth, D.E. The Art of Computer Programming. Vol. 1, Addison-Wesley Publishing Co., Reading, Mass. (1968)

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

[15] 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.

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

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

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

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

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

[21] McCarthy, J. A basis for a mathematical theory of computation. in Computer Programming and Formal Systems, P. Baffort and D. Hirshberg (eds.), North-Holland, Amsterdam (1963), 33-70.

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

[23] 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 1, Harvard University Press (1959), 204-243.

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

[25] Van Horn, E.C. Computer design for asynchronously reproducible multiprocessing. MAC-TR-34 (Thesis), MIT, Cambridge, Massachusetts (Nov 1966).

[26] Wood, P.E. Jr. Switching Theory. McGraw-Hill, New York (1968).