APPENDIX D

A TRANSITION TABLE MODEL FOR THE ANALYSIS OF COORDINATION STRUCTURES

D.1. INTRODUCTION

In this appendix we will consider a formal model for the analysis of coordination structures. The model is based on the transition table models used in circuit design theories (see, for instance, **Hartmanis & Stearns '66**).
A full motivation and elaboration of the ideas presented here would be out of place. We must restrict ourselves to a brief outline.
First we will give an informal description of the main aspects of the model. We then present an overview of the formal model, and discuss its application to some programs encountered earlier in this study. We will conclude with a discussion of comparable models which have been treated earlier in the literature on multiprocessing systems.

Informal description
We model the behavior of a system of concurrent processes, in so far as it is considered relevant to the enforcement of the coordination rules, in a set of transition tables. Each table represents the behavior of one specific process. The tables have rows and columns. The rows represent *control states*, and the columns represent *input states*. The state of the system is expressed and abstracted in a finite set of state variables. For each specific state of these variables we include one column in each table. The input states of all tables should always be consistent. We also define, for each combination of a control state and an input state per table, an *output state*. The output state specifies values for a subset of the state variables, which is called the output set of a table. In all *stable* table states this output state must be consistent with the corresponding input state. In a stable table state we can define a *row transition*. The row transition can bring a table from one control state to another. Usually, such a row transition will also change the output state of a table. Observe, however, that the result may be that the table enters an *unstable* table state. In such a state we can use a *column transition* to bring the table back to a (new) stable state, i.e. a state in which the values of the state variables in the input and the output sets are consistent. A single row transition in one table of a coordinated system can thus cause column transitions in all tables. These column transitions, in fact, model the updating of the state variables after state changes caused by one of the tables. A complete state transition then consists of (at most) two phases:

(1) a row transition in one table, and, if the resulting state is unstable,
(2) a column transition in all tables.

To avoid ambiguities we require that all complete transitions be mutually exclusive.As we shall see below, this restriction does not limit the modelling power, but does greatly simplify the modelling itself.

We distinguish between *blocked* states and *free* states in the tables. A state (a combination of an input state and a control state) is blocked if it cannot

be brought to another control state via a direct row transition. A state is free in the other cases. Observe, that the blocked states can model delay-conditions in programs.
For each stable table state we specify a row transition. In blocked stable states this row transition merely specifies the same control state as the current one. For each unstable state we specify a column transition, and/or an output state (the latter specifies the value of the column transition implicitly). With a careful choice of the set of state variables it should be a straightforward task to construct the transition tables for a given system. It is similarly straightforward to construct the corresponding system state diagram from the tables. The latter diagram can then be analyzed on such items as reachability of desirable system states, correct exclusions, no deadlocks, no starvations, in short: the observance of the coordination rules. The only problem can be that the number of system states to be considered becomes too large. To solve that problem we use reduction techniques borrowed from circuit design theories. We can thus reduce the size of the diagrams to a reasonable size, while preserving all the relevant information for the analysis.
The reductions can be obtained by (1) restricting the set of state variables, and/or (2) applying several notions of state equivalence to combine system states. Below we will elaborate a formal model in which we can express these notions more precisely.

D.2. THE FORMAL MODEL

A *coordinated system* is a tuple (T,J), where:

T is a finite, non-empty, ordered set of *section tables*, and
J is a finite, non-empty, ordered set of *state variables*.

We will indicate the complete set of value-vectors for the variables in J with the symbol $\underline{J}^1$. (abbr. nr. 1)

Example:

$$J = \{a,b,c\},$$

where a, b, and c are binary variables. Then:

$$\underline{J} = \{\{0,0,0\}, \{0,0,1\}, \ldots, \{1,1,1\}\}.$$

A *section table* is a tuple $(Q,U,\lambda,\gamma,\delta)$, where:

Q is a finite, non-empty, ordered set of *control states*;
U is a non-empty, ordered subset of J (with the same ordering);
λ is a projection of $Q \times \underline{J}$ in Q named the *row transition relation*;
γ is a projection of $Q \times \underline{J}$ in $\underline{J}$ named the *column transition relation*;
δ is a projection of $Q \times \underline{J}$ in $\underline{U}$ named the *output relation*.

A table is said to be *blocked* in state (q,i), where:

$$q \in Q \text{ and } i \in \underline{J}$$

if: $$\lambda(q,i) = q.$$ (abbr. nr. 2)

In all other cases the table is said to be *free* or *executable* in that state.
A system is said to be blocked in state (q,i), where:

— — — — —

(1) We have used an abbreviated notation here. The meaning of each abbreviation is specified in a list at the end of this appendix. For each new abbreviation we will refer to this list by means of a number.

$q = (q_1, q_2, \ldots, q_n)$ with $n = |T|$, $i \in \underline{J}$, and $q \in Q = (Q_1 \times Q_2 \times \ldots \times Q_n)$

if: $(\forall j)\ (1 \leq j \leq n \rightarrow \lambda_j(q_j,i) = q_j)$.

In all other cases the system is said to be free or executable in that state[1].

A table is said to be *stable* in state (q,i), if:

$$(\forall j)\ \left(1 \leq j \leq |U| \rightarrow {}_j\delta(q,i) = {}_{N({}_jU,J)}i\right). \qquad \text{(abbr. nrs. 4,6,7)}$$

In all other cases the table is said to be *unstable* in that state.
A system is said to be in a stable state if all its tables are in a stable state, and it is said to be in an unstable state in all other cases.

Requirements

(1) Unstable table states must be blocked:

$$(\forall j)\ \left(1 \leq j \leq |U| \cap {}_j\delta(q,i) \neq {}_{N({}_jU,J)}i \rightarrow \lambda(q,i) = q\right).$$

(2) For every unstable table state the corresponding column transition relation must specify a stable table state with the same control state *and* the same output state:

$$(\forall q,i)\ \Big(q \in Q \cap i \in \underline{J} \cap (q,i)\ \textit{is unstable} \rightarrow (q,\gamma(q,i))\ \textit{is stable} \cap \delta(q,\gamma(q,i)) = \delta(q,i)\Big) \qquad \text{(abbr. nr. 3)}$$

The construction of a system diagram

(1) Select a stable system state (the initial state of the system diagram)

$$(q,i).$$

Define two ordered sets:

Θ named the set of *reachable states*, and
Φ named the set of *successor states*.

Both sets are initially empty. The successor states of the j-th state in Θ will be listed in random order in the j-th subset of Φ.

(2) Θ +:= (q,i) (a right-addition of one element to ordered set Θ),

Φ +:= $\{s_j:\ 1 \leq j \leq |T|$

$\cap\ (s_j = (q[q_j := \lambda_j(q_j,i)],i) \cap \lambda_j(q_j,i) \neq q_j \cap (\lambda_j(q_j,i),i)\ \textit{is stable})$ (abbr. nr. 8)

$\cup\ s_j = (q[q_j := \lambda_j(q_j,i)]\ ,\ i[U_j := \delta_j(\lambda_j(q_j,i),i)]) \cap (\lambda_j(q_j,i),i)\ \textit{is unstable})\}$ (abbr. nr. 9)

(3) Select an arbitrary state (q',i') from an arbitrary non-empty subset of Φ, but only one which is not yet included in set Θ:

$$(q',i') \notin \Theta.$$

— — — — —

(1) The subscript j refers here to the j-th table in the coordinated system considered.

Make: $(q,i) := (q',i')$

and return to step 2, unless no such (q',i') exists.
(Observe, that this procedure will always terminate within a finite number of steps.)

Equivalences

In order to be able to reduce a coordinated system, and the corresponding system diagram, we must define a notion of state- and system-equivalence.
We define equivalence in terms of reduced sets of state variables.
Define a reduced input set $\bar{J}$ as an ordered subset of J.
The equivalence properties for states are related only to those states which are listed in set Θ (i.e. to reachable *and* stable system states).

(1) State a in coordinated system A is said to be 1-equivalent to state b in system B, with respect to reduced input sets $\bar{J}^A$ and $\bar{J}^B$, if:

$$\bar{J}^A(a) = \bar{J}^B(b). \qquad \text{(abbr. nr. 10)}$$

This is briefly indicated as:

$$(a,b) \in E\,(\bar{J}^A,\bar{J}^B).$$

(2) State a is said to be k-equivalent to state b, for $k > 1$, i.e. $(a,b) \in E_k(\bar{J}^A,\bar{J}^B)$, if:

1. $\bar{J}^A(a) = \bar{J}^B(b)$, *and*

2. $(\forall e,p)\ (e \in (\Phi^A(a))^p \cap \bar{J}^A(e) \neq \bar{J}^A(a) \cap$ (abbr. nrs. 11,12)

$(\forall e',p')\ (e' \in (\bar{\Phi}^A(e'))^{p'} \cap 1 \leq p' < p \rightarrow \bar{J}^A(e') = \bar{J}^A(a)) \cap$ (abbr. nr. 13)

$(\exists f,q)\ (f \in (\Phi^B(b))^q \cap \bar{J}^B(f) \neq \bar{J}^B(b) \cap$

$(\forall f',q')\ (f' \in (\bar{\Phi}^B(f'))^{q'} \cap 1 \leq q' < q \rightarrow \bar{J}^B(f') = \bar{J}^B(b)) \rightarrow$

$(e,f) \in E_{k-1}(\bar{J}^A,\bar{J}^B))$, *and*

3. $(\forall f,q)\ (f \in (\Phi^B(b))^q \cap \bar{J}^B(f) \neq \bar{J}^B(b) \cap$

$(\forall f',q')\ (f' \in (\bar{\Phi}^B(f'))^{q'} \cap 1 \leq q' < q \rightarrow \bar{J}^B(f') = \bar{J}^B(b)) \cap$

$(\exists e,p)\ (e \in (\Phi^A(a))^p \cap \bar{J}^A(e) \neq \bar{J}^A(a) \cap$

$(\forall e',p')\ (e' \in (\bar{\Phi}^A(e'))^{p'} \cap 1 \leq p' < p \rightarrow \bar{J}^A(e') = \bar{J}^A(a)) \rightarrow$

$(e,f) \in E_{k-1}(\bar{J}^A,\bar{J}^A))).$

Less formally, this means that two states are said to be k-equivalent if they have the same reduced input state, *and* if for the first successors of either state with a different input we can find a (k - 1)-equivalent state in the set of successors of the other state, among the first new states which have a different input. (In practice the observance of this property is trivially verified, unlike what one would be tempted to conclude from the complexity of the above sentence.)

The coordinated systems are said to be equivalent, with respect to some reduction sets, if we can find for each state in the one system a k-equivalent state in the other system for all $k \geq 1$, and vice versa.

Reductions: the construction of a minimal system

Consider two equivalent systems; call them A and B.
Consider two states from A which are equivalent to the same state in B; call them a_1 and a_2. We say that a_1 is in relation E' to a_2, or briefly: $(a_1,a_2) \in E'$. (For brevity we delete the specification of the reduced input sets here.) Observe, that relation E' has all the properties of an equivalence relation on the set of states in system A: $(Q \times \underline{J})$ (symmetry, reflexiveness, transitivity). The relation E' therefore defines a partitioning of this set of states in A. Call this partitioning π.
By combining the equivalent states in system A we can construct a minimal system A'. To find A' we can use π. Each block in the partitioning π corresponds to one specific state in A'.

The problem of constructing a minimal system has now been reduced to the problem of finding the partitioning π. We can do this in steps. First we define a relation E'_k which formalizes k-equivalence of states: $(a_1,a_2) \in E'_k$ implies that a_1 and a_2 are k-equivalent to the same state in some system B. Observe, that E'_k is also an equivalence relation, and thus also defines a partitioning, which we shall call π_k.

It is readily seen that $(a_1,a_2) \in E'_k$ implies $(a_1,a_2) \in E'_{k-1}$, for all $k > 1$. It follows directly that partitioning π_k must be a refinement of partitioning π_{k-1}. As the set of states in each coordinated system is by definition finite this also implies that there is always a finite integer k for which $\pi_k = \pi_{k-1}$. The desired partitioning π can now be found with the aid of the following theorem:

$$(\forall k) \left(k > 1 \cap \pi_k = \pi_{k-1} \rightarrow \pi = \pi_k \right).$$

Proof

It is first proved that the following property holds:

$$(\exists k) \left(k > 1 \cap \pi_k = \pi_{k-1} \rightarrow (\forall h) \ (h \geqslant 0 \cap \pi_{k+h} = \pi_{k-1}) \right)$$

The proof is by induction on h.

(1) The property holds for $h = 0$.

(2) Assume, that the property holds up to some value $m \geqslant 0$. This implies:

$$\pi_{k+h} = \pi_{k-1} \text{ or } (\forall a_1,a_2) \left((a_1,a_2) \in E'_{k+h} \leftrightarrow (a_1,a_2) \in E'_{k-1} \right) \text{ for } 0 \leqslant h \leqslant m.$$

It follows that:

$$(\forall e) \left(e \in \Phi(a_1) \rightarrow (\exists f) \ (f \in \Phi(a_2) \cap (e,f) \in E'_{k+h-1}) \right),$$

and that:

$$(\forall f) \left(f \in \Phi(a_2) \rightarrow (\exists e) \ (e \in \Phi(a_1) \cap (e,f) \in E'_{k+h-1}) \right),$$

again for $0 \leqslant h \leqslant m$.
It follows from the assumption that:

$$\pi_{k+h-1} = \pi_{k+h},$$

for $0 \leqslant h \leqslant m$, and therefore:

$$(e,f) \in E'_{k+h-1} \leftrightarrow (e,f) \in E'_{k+h},$$

which in turn implies that:

$$(a_1,a_2) \in E'_{k+h+1},$$

(according to the definitions of state equivalence).
If the property then holds up to some value $m \geqslant 0$ it will also hold up to the value $h = m + 1$, and so on.

This completes the proof of the property stated above.
With the aid of this property it is easy to prove the theorem.
By definition:

$$(a_1,a_2) \in E' \leftrightarrow (a_1,a_2) \in E'_k, \text{ for all } k \geqslant 1.$$

Above we have shown that there is always a finite value of k for which:

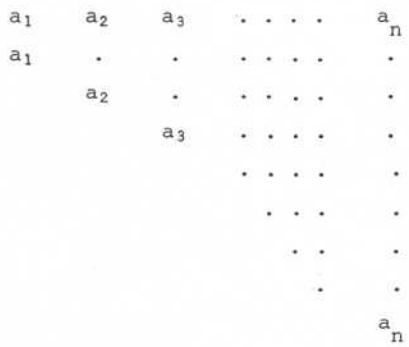
$$(a_1,a_2) \in E'_k \leftrightarrow (a_1,a_2) \in E'_{k+h}, \text{ for all } h \geqslant 0.$$

We can further readily see that:

$$(a_1,a_2) \in E'_k \rightarrow (a_1,a_2) \in E'_{k-h}, \text{ for all } 0 \leqslant h \leqslant k-1,$$

which completes the proof of the theorem.

The minimal system can now be derived quite rapidly with the aid of the so-called *triangle method*. Each possible combination of two attainable states in the coordinated system is represented in a triangle. In a first scan of that triangle we place a cross on all combinations which are *not* in π_1.
In a second scan we consider the remaining places, and we place a cross on all combinations not in π_2. This is repeated until no new crosses can be added in one scan of the still open places in the triangle. The open places in that final triangle represent all combinations in partitioning π.



D.3. APPLICATION

Now that we have the formalism over with, we can study the application of the transition table model. First we consider a simple exclusion problem, for programs with the following structure:

```
repeat A: down A;
          critical program part;
          up A;
          non-critical program part
forever
```

'A' is assumed to be a binary d-semaphore variable, initialized to 1.
We first notice that the value of semaphore variable A cannot be used to abstract the state of the system completely. The value A = 0 merely says that *some* process is in its critical section, but it is not known *which* process this concerns.
To make the modelling possible we must introduce a set of additional (boolean) state variables a_1, ..., a_n, where 'n' is the number of processes, and state variable a_i models a sort of *progress mark* (control-flow point) for the i-th process:

a_i = true means that process i is presently *not* in its critical part;
a_i = false means that process i *is* in its critical program part.

A concise table results if we make the relations λ, γ, and δ of each table only dependent on the 'a' values of the *other* tables (processes). To this purpose we define a state variable A_i for table i, as follows:

$$A_i = \underset{j \neq i}{\Delta} a_j,$$

where Δ denotes a logical AND-combination of boolean values.
With these simplifications we can represent the table for process i as follows:

A_i	T	F	
α	β	(α)	T
β	α	-	F
			a_i

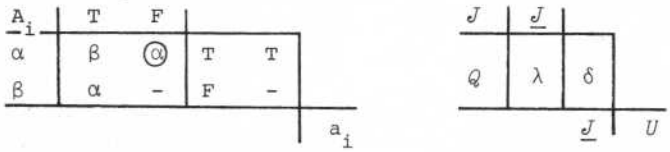
J	$\underline{J}$	
Q	λ	δ
		U

The chosen table lay-out is also shown above. The following points should be noted:

(1) Instead of the full set J, which would contain all state variables a_1, ..., a_n, we have used the concise state variable A_i, with potentially a different value in two different tables. This simplification allows us to construct a table which is independent from the precise number of processes involved in the coordination (i.e. independent from the value of 'n').
As a result of this choice of the variable A_i there are no unstable states in the table presented above.

(2) The value of the row transition relation is circled in the table if the corresponding state is blocked (see state (α,F)).

(3) In the above table the output states are independent from the input states of the table. The output states can therefore be specified per control state (row). A full table, which would contain some redundant information, would look as follows:



(4) Unattainable states are indicated by a dash. Observe, that there are only three attainable table states:

1. state (α,T), which models a state in which the process considered is not in its critical program part, and no other process is,
2. state (α,F), which models a state in which the process considered is prevented from entering its critical program part, and
3. state (β,T), which models the state in which process i is in its critical program part and prevents others from entering theirs.

As a slightly more comprehensive example, let us consider the table for a set of programs with the following structure:

```
repeat A: down A;
       A: down A;
          critical program part;
          up A;
          up A;
          non-critical program part
forever
```

'A' is assumed to be a general d-semaphore variable, with initial value 2. Again we will use a set of additional variables $a_1, \ldots, a_n$. This time each a_i is an *integer* variable with values on the interval (0, 1, 2). The interpretation of these values is as follows:

$a_i = 0$, means that process i is presently *not* in its critical program part, and has not decremented 'A' more often than it has incremented it,

$a_i = 1$, means that process i is presently *not* in its critical program part, but has decremented 'A' *once* more than it has incremented it,

$a_i = 2$, means that process i *is* in its critical program part.

State variable A_i is now defined as follows:

$A_i = 0$ iff (if-and-only-if)

$$(\exists j)\ (1 \leqslant j \leqslant n \cap j \neq i \rightarrow a_j = 0);$$

$A_i = 1$ iff

$$(\forall j)\ (1 \leqslant j \leqslant n \cap j \neq i \rightarrow a_j \geqslant 1) \cap$$

$$(\exists j)\ (1 \leqslant j \leqslant n \cap j \neq i \rightarrow a_j = 1);$$

$A_i = 2$ iff

$$(\forall j)\ (1 \leqslant j \leqslant n \cap j \neq i \rightarrow a_j = 2).$$

The resulting table, for the i-th process, is as follows:

A_i	2	1	0	
α	β	β	(α)	0
β	ψ	(β)	-	1
ψ	ε	-	-	2
ε	α	-	-	1
				a_i

Notice the danger of a deadlock in state (β,1)!
The use of the system diagram and the reduction procedures will be elucidated with the aid of a third example, which is still more comprehensive.
We consider the earlier derived programs for the third readers/writers' problem.

Readers:
repeat

$(WW \cup \bar{P}) \cap AW$: **down** WR;
down AR **up** WR;
critical reading;
set (P,1);
up AR;

forever

Writers:
repeat

$(WR \cup P) \cap AR \cap AW$: **down** WW;
down AW **up** WW;
critical writing;
set (P,0);
up AW;

forever

To be able to construct a concise set of tables we define the additional binary variables: wr_i, ar_i, ww_j, and aw_j, with:

$1 \leqslant i \leqslant r$ (r is the number of reader processes),
$1 \leqslant j \leqslant w$ (w is the number of writer processes), and
$1 \leqslant k \leqslant r + w$.

The interpretation is as follows:

$wr_i = 0$, means that reader-process i has decremented semaphore variable WR more often than it has incremented it,
$wr_i = 1$, means that reader-process i has decremented semaphore variable WR as often as it has decremented it,

and similarly for ar_i, ww_j, and aw_j.

We now define the state variables: WW, AW, WR, P, and A_j:

$WW = 1$ iff $(\forall j)\ (1 \leqslant j \leqslant w \rightarrow ww_j = 1)$

$WW = 0$ otherwise,

and similarly for AW and WR.

P merely represents the value of the semaphore variable P.

$A_j = 1$ iff $(\forall i)\ (1 \leqslant i \leqslant r \rightarrow ar_i = 1) \cap$

$(\forall k)\ (1 \leqslant k \leqslant w \cap j \neq k \rightarrow aw_k = 1)$

$A_j = 0$ otherwise.

To reduce the size of the tables we have combined equal columns, and have indicated by a dash, those state variables in the input set whose precise value is irrelevant.

Readers' table ($1 \leq i \leq r$)

WW AW P	-10	111	-01/011	-00			
α	β	β	β	β	110	111	
β	ψ	ψ	Ⓑ	Ⓑ	010	011	
ψ	ε	ε	α	-	100	101	
ε	○	α	α	-	101	101	
					--0	--1	wr_i ar_i P

Writers' table ($1 \leq j \leq w$)

WR A_j P	-11	110	-00/010	-01			
α	β	β	β	β	110	111	
β	ψ	ψ	Ⓑ	Ⓑ	010	011	
ψ	ε	ε	α	-	100	101	
ε	○	α	α	-	100	100	
					--0	--1	ww_j aw_j P

The construction of the system diagram

We will indicate table states with *place numbers*, for brevity. The state in the i-th control state (row) with the j-th input state (column) is assigned the place number:

$$(R.(j - 1) + i),$$

where R is the number of control states (rows) in set Q: $R = |Q|$.

Example:

In the readers' table we have:

- place number $((\alpha,-10)) = 1$, and
- place number $((\beta,-00)) = 14$.

With the aid of the generating procedure discussed earlier we can construct the sets Θ and Φ. In a manual analysis this is only practical for small numbers of readers and writers. (We discuss a method to circumvent this problem below.) For one reader and one writer we arrive at the following results:
Each element in set Θ, and each element in each subset of Φ is specified by a pair of place numbers. The first number in the pair refers to the readers' table; the second number refers to the writers' table. The initial state is (1,5).

Θ = {(1,5), (2,9), (3,9), (8,13), (5,1), (9,2), (9,3), (13,8), (1,6), (13,7), (6,1), (7,13), (2,10), (10,2), (3,10), (10,3), (12,14), (14,12), (14,11), (11,14)}.

There are 20 reachable states. Again for brevity we will specify these states below by refering to their order number in set Θ, as given above.

State *8* thus means state (13,8), which means: ((α,-00), (ε,110)). (In this specific state the writer is about to leave its critical section.)
Set Φ will be specified with the abbreviated state numbers:

$$\Phi = \{\{2,9\}, \{3,13\}, \{4,15\}, \{5,17\}, \{6,11\}, \{7,14\}, \{8,16\}, \{1,18\}, \{10,13\}, \{8,19\}, \{12,14\}, \{4,20\}, \{15\}, \{16\}, \{17\}, \{18\}, \{6\}, \{2\}, \{2\}, \{6\}\}.$$

More conveniently still we can list the contents of these two sets in a table; specifying for each reachable state the corresponding immediate successors:

1 → 2,9	11 → 12,14
2 → 3,13	12 → 4,20
3 → 4,15	13 → 15
4 → 5,17	14 → 16
5 → 6,11	15 → 17
6 → 7,14	16 → 18
7 → 8,16	17 → 6
8 → 1,18	18 → 2
9 → 10,13	19 → 18
10 → 8,19	20 → 17

The complete system diagram is given in figure D.1. It contains 20 states and 32 transition links. Note the duality between respectively states:

(1,2,3,13,15,9,10,8,18,19), and
(5,6,7,14,16,11,12,4,17,20).

The first subset relates to reader preference (P = 0); the second subset relates to writer preference (P = 1).

Reducing the system with respect to an input set which contains only the state variables AR and AW we arrive at a diagram of 12 states and 20 transition links, as shown in figure D.2.
Observe, that the system diagram thus reduced differs fundamentally from the diagram which results if one just combines all system states with an equivalent reduced input state, as shown in figure D.3. The latter diagram contains less information, and it does contain spurious paths which may frustrate a correctness analysis[1]. Note, for instance, that the latter diagram cannot be used to verify whether starvation or deadlock is possible. It can only be used to establish the mutual exclusion property. The diagram derived with the reduction procedure described in this appendix does, however, contain all the relevant information.
Reducing the system diagram with respect to an input set which contains also WR and WW yields the same result as in figure D.2. Reducing the diagram with respect to an input set which contains only state variable P would yield a trivial diagram of two states (one for P = 0 and one for P = 1) and two transition links.

For larger numbers of readers and/or writers the size of even the reduced system diagram will be unmanageably large if we do not apply still other notions of state equivalence. For instance, we need not distinguish between readers as such and writers as such; we only want to distinguish between reader-actions and writer-actions. A state in which reader "i" is requesting access to its critical section may therefore well be considered equivalent to a state in which some reader "j", with $i \neq j$, requests access to its section.

— — — — —

(1) Cf. **Howard & Alexander '73.**

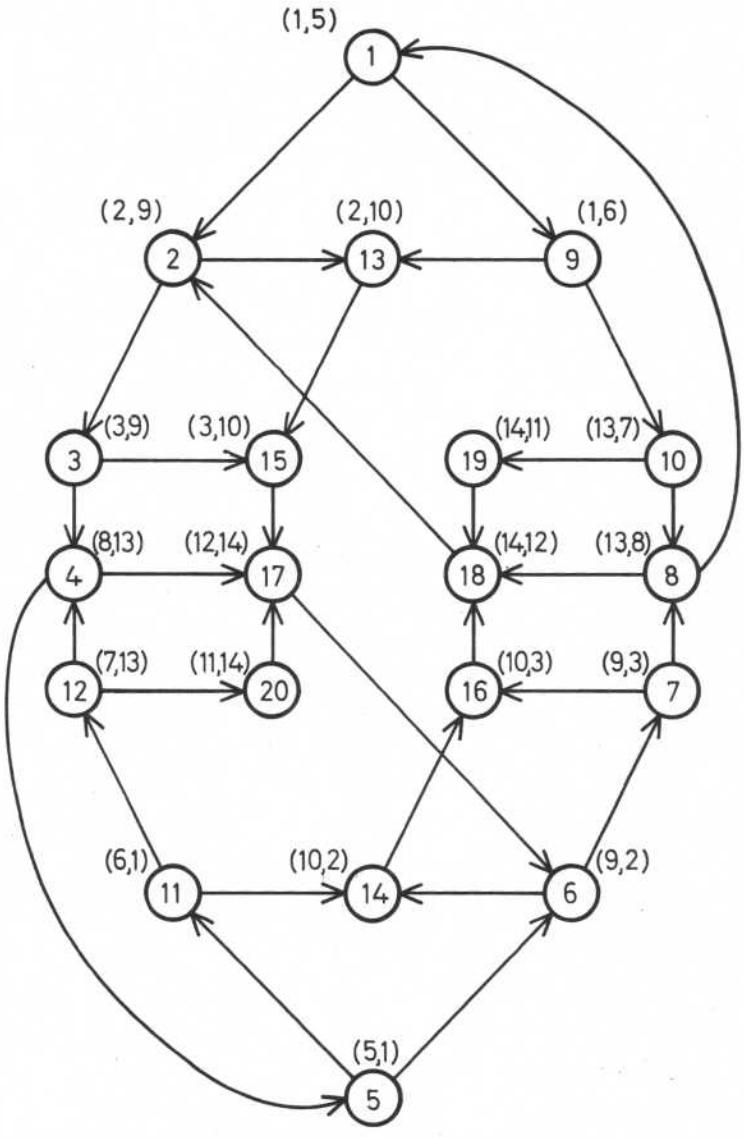


Figure D.1.
Non-Reduced System Diagram
for Readers & Writers' Problem
(1 reader and 1 writer)

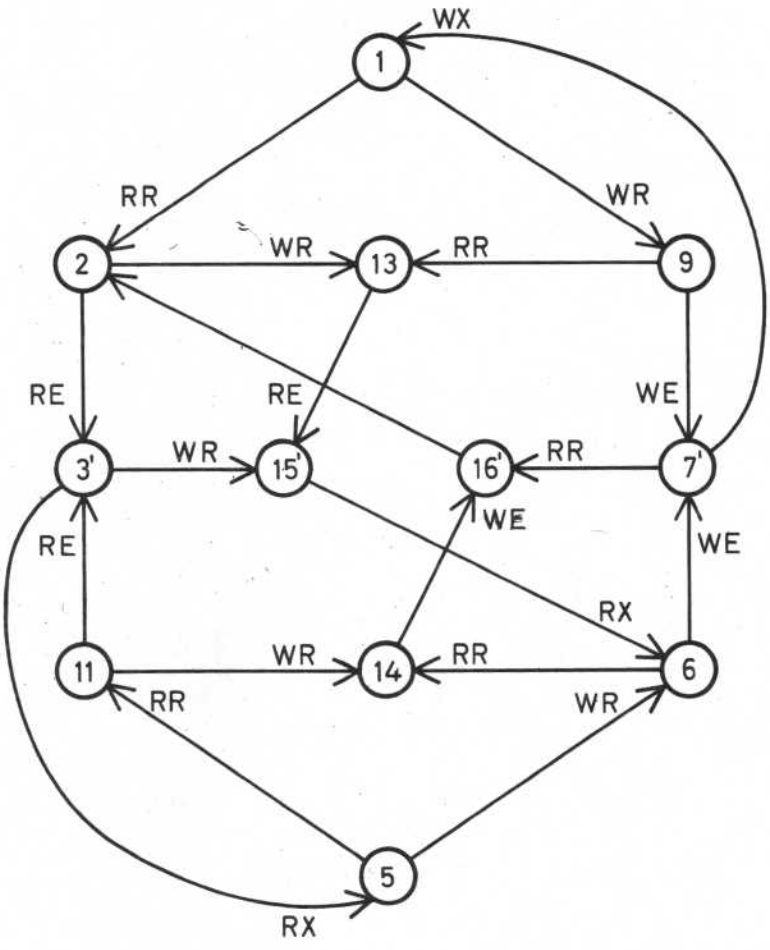


Figure D.2.
Reduced System Diagram
3d Readers & Writers' Problem

RR = Reader Request
RE = Reader Entry
RX = Reader Exit
WR = Writer Request
WE = Writer Entry
WX = Writer Exit

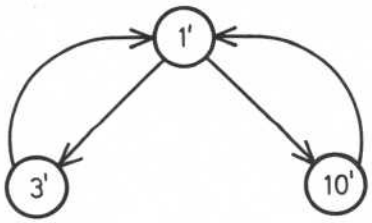


Figure D.3.
Folded System Diagram
3d Readers' & Writers' Problem
1' = 1, 2, 5, 6, 9, 11, 13, 14.
3' = 3, 4, 12, 15, 17, 20.
10' = 7, 8, 10, 16, 18, 19.

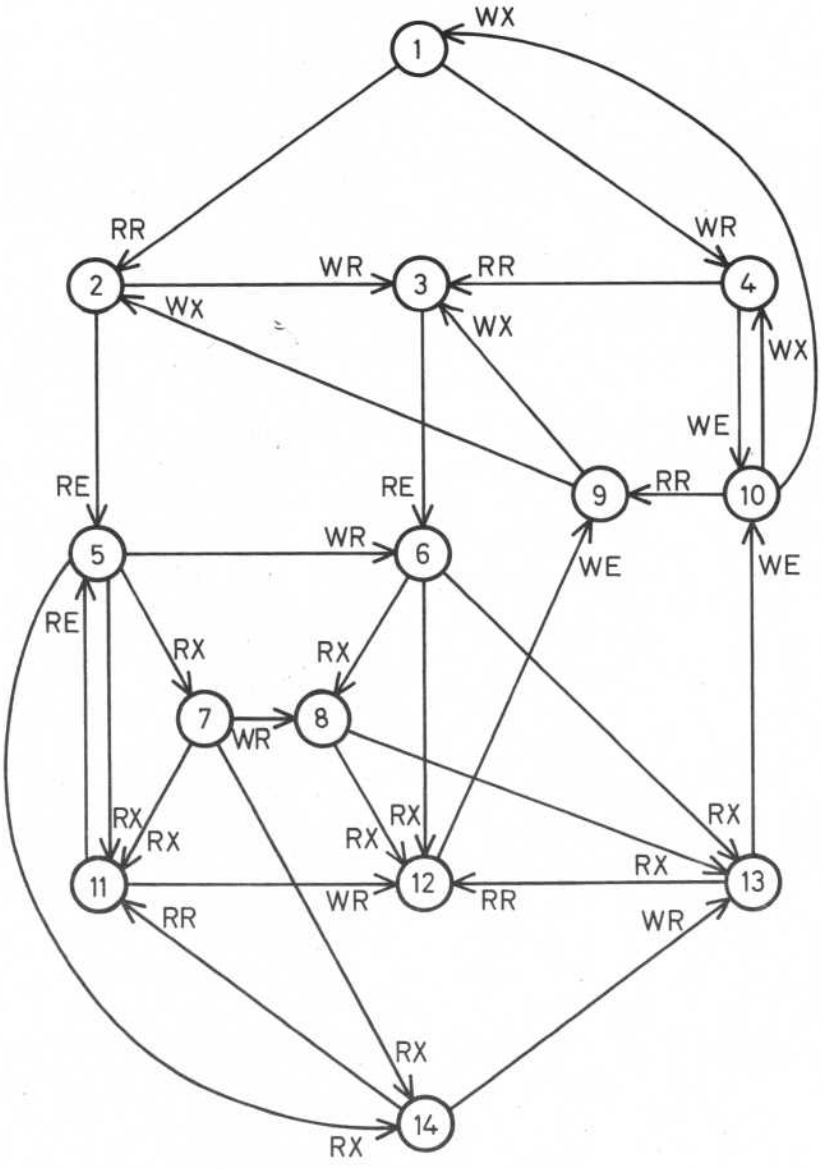


Figure D.4.
Reduced System Diagram
3d Readers & Writers' Problem
(arbitrary number of readers and writers)

The generating procedure for the construction of the system diagram must be adapted such that one can combine these equivalent[1] states in an early stage. In this manner one can derive a concise system diagram, like the one presented in figure D.4, which holds for arbitrary numbers of readers and writers. Below we give an interpretation of the diagram, while refering to the state numbers:

(1) Empty state. All processes are in the initial state (i.e. 'idle'). Reader preference.
(2) One or more readers have requested access. All others are idle. Reader preference.
(3) One or more readers and one or more writers have requested access. All others are idle. Waiting writers are blocked; waiting readers are free. Reader preference.
(4) One or more writers have requested access. All others are idle. Reader preference.
(5) One or more readers are executing; zero or more readers have requested access. All others are idle. No reader has completed since state 5 was entered.
(6) One or more readers are executing; zero or more readers have requested access, and one or more writers have requested access. All others are idle. No reader has completed since state 5 or 6 was entered. Waiting writers are blocked.
(7) See state 5. At least one reader has completed.
(8) See state 6. At least one reader has completed. Waiting readers and waiting writers are blocked.
(9) One writer is executing. One or more readers, and zero or more writers have requested access. Waiting readers and waiting writers are blocked.
(10) One writer is executing. Zero or more writers are waiting, and blocked. All others are idle.
(11) See state 2. Writer preference.
(12) See state 3. Waiting readers are blocked; waiting writers are free. Writer preference.
(13) See state 4. Writer preference.
(14) See state 1. Writer preference.

Observe, that the solution analyzed here guarantees that writers cannot be starved, which is a result of the inclusion of states 7 and 8. (Waiting readers are blocked in state 8.)

D.4. DISCUSSION

The model developed in this appendix is inspired by the Mealy and Moore table models from sequential machine theory. There are, however, important differences from these models. The conventional models do not allow for the column-transitions, which play an essential role in the model developed here. The conventional models do not allow for the overlapping of input and output sets, and there the output relations are defined differently.

> (Cf. "... the outputs of a Mealy machine are thought of as occurring while the machine is going through a transition between states." **(Hartmanis & Stearns '66, pg. 16).**)

The reasons for our diverting from the conventional models are that a straightforward application of these models, which were explicitly designed for *sequential* machines, would create severe modelling problems, like:

— — — — —

(1) Observe, that we give a new interpretation to the term 'equivalence' here.

- timing problems;
- an insuperable growth of the system diagrams, and
- a loss of modelling clarity.

Earlier, **Bredt '70** has described the application of a transition table model to the analysis of arbitration algorithms in a multiprocessing system. Bredt assumed the availability of a special system component (a circuit or a process) to which all arbitration concerns could be delegated. The competing processes are assumed to send request signals to this 'arbiter' upon their entry to, and exit from, their critical program sections. The processes are forced to wait until the arbiter acknowledges their request.
A similar approach was briefly outlined by **Boute '78** ('the multiple finite state machine').
Bredt-did allow for finite but *unbounded* 'signal propagation times' between system components (e.g. processes). In the present model this would imply that the time between a row transition and the consecutive column transition(s) is unbounded, and that still other row transitions may be triggered in the meantime. Bredt was only able to handle the resulting increase in complexity by restricting himself to one specific type of coordination (i.e. with a dedicated system component: the arbiter) and a specific signalling strategy.
The access of state variables in a coordinated system (e.g. via semaphore operations) is, however, mutually excluded anyway, so there is no need to allow for unbounded signal propagation times in the present case at all.
Gilbert & Chandler '72 described a formal model for the analysis of concurrent processes with the aid of transition rules. Their method could, however, lead to an infinite set of transition rules, which would preclude analysis. Some improvements were presented by **Lister '74**. The method described by Lister can, however, still lead to an unmanageably large set if transition rules and system states, though not infinitely large. Neither Gilbert & Chandler nor Lister applied or mentioned the transition table methods, though the underlying ideas of their methods are certainly comparable. The transition table method described in this appendix improves both Gilbert & Chandler's and Lister's method significantly.
As an example: the first (simplest) version of the readers/writers' problem with reader-priority, was analyzed by Lister. The system which solved this problem was modelled in a state diagram of 50 states and 88 transitions.
With the transition table model described here we could model the third (most complex) version of the readers/writers' problem in a state diagram of only 14 states and 31 transitions. Modelling the same simple problem as analyzed by Lister would lead to a reduced diagram of only 4 states and 6 transitions.

Control-flow predicates
It is fairly simple to model processes with arbitrary nestings of selection structures and/or iteration structures. Observe, that one can simply include the set of relevant control-flow predicates (and the operations on these) in sets or private 'auxiliary variables'.

Automated analysis and design of coordination structures
Each coordinated system of processes can be modelled in a set of transition tables and thus be analyzed with the aid of a system diagram. We may however conjecture that the reverse is also true: from a system diagram one can derive transition tables which can be translated into a set of concurrent processes coordinated with (extended d-) semaphores. (Assuming that the number of processes is known, and that the interpretation of the (given) system diagram is known, i.e. which transitions correspond to which processes.) The problem, for the moment, is that a single state diagram can yield a great number of transition tables of which only few correspond to practical programs. The adequacy of the tables depends heavily on the precise coding of states in state

variables selected, and there are many ways to select such codings. We will not elaborate these points any further here, but leave them as an interesting subject for further research.

D.5. REFERENCES

Bredt, T.H. (1970), *The mutual exclusion problem*, Techn. Report Nr. 9, Stanford Electronics Lab., STAN-CS-70-173, Aug. 1970, 62 pgs.

Boute, R.T. (1978), *Logical models for computer control of telephone exchanges*, Proc. Third Int. Conf. on Softw. Eng., for Telec. Switching Systems, Helsinki, June 1978, pp. 18-24.

Gilbert, P. & Chandler, W.J. (1972), *Interference between communicating parallel processes*, Comm. ACM, Vol. 15, No. 6, June 1972, pp. 427-437.

Hartmanis, J. & Stearns, R.E. (1966), *Algebraic structure theory of sequential machines*, Prentice Hall, Int. Series in Appl. Math., 1966.

Howard, J.H. & Alexander, W.P. (1973), *Analyzing sequences of operations performed by programs*, In: Program Test Methods, W.C. Hetzel (ed.), Prentice Hall, 1973, pp. 239-254.

Lister, A.M. (1974), *Validation of systems of parallel processes*, Computer Journal, Vol. 17, No. 2, 1974, pp. 148-151.

Symbols

T set of section tables,
J set of state variables; input set,
Q set of control states,
U set of output variables; output set,
λ row transition relation,
γ column transition relation,
δ output relation,
Θ set of reachable system states,
Φ set of successor states.

For the above symbols we use (1) right-hand superscripts to denote a specific coordinated system, (2) right-hand subscripts to denote a specific table in a system, and (3) left-hand subscripts to denote a specific set-element. Thus the symbol ${}_{j}S^{A}_{t}$ denotes the j-th element in set S of the t-th table in system A.

E_k state equivalence relation between systems,
E'_k state equivalence relation within systems,
π_k the partitioning of $(Q \times \underline{J})$ defined by E'_k.

Abbreviations

1. $\underline{S}$, where S is a set of variables: the complete set of value-vectors,
2. $\lambda(q,i) = q'$ is short for: $((q,i),q') \in \lambda$,
3. $\gamma(q,i) = i'$ is short for: $((q,i),i') \in \gamma$,
4. $\delta(q,i) = u$ is short for: $((q,i),u) \in \delta$,
5. $|S|$, where S is a set: the cardinal number of set S,
6. $\mathbf{N}(s,S)$, with $s \in S$: the order number of element s in ordered set S,
7. ${}_{j}S$, where S is an ordered set: the j-th element in S,
8. $q\,[q_j := q']$, where $q = (q_1, q_2, \ldots, q_n)$: the result of the replacement of the j-th element in q with (the value of) q',
9. $i\,[U_j := u']$, where $i \in \underline{J}$, and $u' \in \underline{U_j}$: the result of the replacement of the current values of the variables specified in U_j (i.e. the output set of the j-th table) by those in u', that is:

$$(\forall m)\ \left(1 \leqslant m \leqslant |U_j| \rightarrow {}_{\mathbf{N}({}_{m}U_j, J)}{}^{i} := {}_{m}\delta_j(\lambda_j(q_j,i),i)\right),$$

10. $S(a)$, where S is a set of state variables, and a is a stable system state: the value-vector which specifies the current values of the input variables in state a.
11. $\Phi(a)$, the set of successors of state a: ${}_{N(a,\Theta)}\Phi$.
12. $(\Phi(a))^p$, the p-th generation of successors of state a:

$$(\Phi(a))^p = \bigcup_{e \in \Phi(a)} (\Phi(e))^{p-1} \text{ and } (\Phi(e))^1 = \Phi(e),$$

13. $(\bar{\Phi}(a))^p$, the p-th generation of predecessors of state a:

$$(\bar{\Phi}(a))^p = \bigcup_{e \in \bar{\Phi}(a)} (\bar{\Phi}(e))^{p-1} \text{ and } (\bar{\Phi}(e))^1 = \bar{\Phi}(e) = \{s: e \in \Phi(s)\}.$$