Deadlock Analysis in Networks of Communicating Processes

S. D. Brookes
Carnegie-Mellon University
Pittsburgh, Pa.
USA

A. W. Roscoe
Programming Research Group
Oxford University
Oxford
England

To appear in: Proceedings of Advanced NATO Study Institute on Logics and Models for Verification and Specification of Concurrent Systems, La Colle-sur-Loup, October 1984, Springer Verlag Lecture Notes in Computer Science (1985).

The research reported in this paper was supported in part by funds from the Computer Science Department of Carnegie-Mellon University, and by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under Contract F33615-81-K-1539. The views and conclusions contained in it are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

DEADLOCK ANALYSIS IN NETWORKS OF COMMUNICATING PROCESSES

S. D. Brookes
Carnegie-Mellon University
Pittsburgh, Pa.
USA

A. W. Roscoe
Programming Research Group
Oxford University
Oxford
England

0. Abstract.

We use the failures model of Communicating Sequential Processes to describe the behaviour of a simple class of networks of communicating processes. This model is well suited to reasoning about the deadlock behaviour of processes, and we demonstrate this fact by proving some results which help in the analysis of deadlock in networks. In particular, we formulate some simple theorems which characterise the states in which deadlock can occur, and use them to prove some theorems on the absence of global deadlock in certain classes of systems. Some examples are given to show the utility of these results.

1. Introduction.

In [3,4,5] we described the failures model of communicating processes and used it to describe some interesting parallel programming examples. We stated there that the model was well suited, by its very construction, to reasoning about the potential or the absence of deadlock in systems of processes. In this paper we elaborate this point in some detail, developing some ideas which originated in Roscoe's thesis [15]. We provide some simple yet useful theorems which may be used to analyse networks for the potential of deadlock. We demonstrate the utility of these results by examining some examples from the literature. We compare our work briefly with earlier work by several other authors, and make some suggestions for future research.

The simplicity of the mathematical structure of the failures model lends itself to clean formulation of deadlock properties and to formal manipulation of process behaviour. We

assume familiarity with the material of either [3,4] or [5], where details were given of the syntax for processes in a version of CSP and of the mathematical construction of the failures model. We use P and Q to range over processes. A failure is a pair (s, X) consisting of a trace s (a finite sequence of events) and a refusal set X (a set of events). If (s, X) is a possible failure of a process P, we interpret this as saying that the process may refuse all of the events in X after having performed the sequence s; thus, if the process is placed in an environment which only wants to perform events from this set at that stage, deadlock is possible. The improved failures model of [5] also allows a treatment of the phenomenon of *divergence*, which occurs when a process is able to perform an unbounded number of internal actions without communicating with its environment. Processes were described as pairs (F, D), with F being a failure set and D a divergence set. We took in [5] a pessimistic view of divergence, regarding the possibility of divergence as catastrophic. In this view it is useless to try to prove absence of deadlock if there is a possibility of divergence. We will therefore assume in this paper that all processes are divergence-free (have empty divergence set), so that a process is fully described by its failure set. We use the notation $\mathcal{F}[\![P]\!]$ for the failure set of a process P. In [5] we also allowed for the possibility of infinite refusal sets when processes were able to use infinite alphabets. This is important since it allows cleaner statements and easier proofs for several results.

2. Networks of Communicating Processes.

Graphical representations of networks of processes have been used extensively in the literature, notably by Milne and Milner [13]. Almost every paper on deadlock analysis uses a more or less formal notion of network. Our notation will be as follows.

A network is a graph with nodes of the form (P_i, A_i), consisting of a process P_i and alphabet A_i, and with an arc (determined uniquely by the set of nodes) from (P_i, A_i) to (P_j, A_j) iff $A_i \cap A_j \neq \emptyset$ and $i \neq j$. Thus two processes are linked in the graph if and only if their alphabets indicate that there is an event representing a possible communication between them. Of course, this says nothing about whether or not such a communication will ever take place dynamically, and the network structure is static. It may be convenient to think of the arcs in a network as representing communication links. Since CSP treats communication in a more or less symmetric fashion, we do not assign directions to the arcs. Note that in the case of a system in which processes are defined by recursion, the network can be thought of as potentially infinite; a recursive expansion of a process definition can be viewed as replacing a node of the system by a new graph.

For a network $V = \{(P_i, A_i) \mid 1 \leq i \leq n\}$ we define the parallel composition $\mathrm{PAR}(V)$ to be

$$\mathrm{PAR}(V) = \|_{i=1}^{n} (P_i, A_i).$$

This is a *mixed parallel composition* as in [5], with process P_i using alphabet A_i. Note that for a network $V = \{(P_i, A_i) \mid 1 \leq i \leq n\}$ the behaviour after the trace s will be that of the network V after s defined by:

$$V \text{ after } s = \{(P_i \text{ after } s\lceil A_i, A_i) \mid 1 \leq i \leq n\},$$

because at this stage the process at node i has performed the sequence $s\lceil A_i$, obtained by including only the events in s which belong to the set A_i.

Examples.

Example 1. Dining Philosophers.

In this example due to Dijkstra, well known from the literature [9,12], there are five philosophers, PHIL_i, $(i = 0\ldots4)$, five forks FORK_i $(i = 0\ldots4)$, and a butler process BUTLER. In our version the alphabets of these processes are:

$$\begin{aligned} A_i &= \{\text{i.picks.i, i.puts.i, i.eats, i.enters, i.leaves, i.picks.i+1, i.puts.i+1}\}, \\ B_i &= \{\text{i.picks.i, i}-1\text{.picks.i, i.puts.i, i}-1\text{.puts.i}\}, \qquad (i = 0, \ldots, 4), \\ C &= \{\text{i.enters, i.leaves} \mid 0 \leq i \leq 4\}, \end{aligned}$$

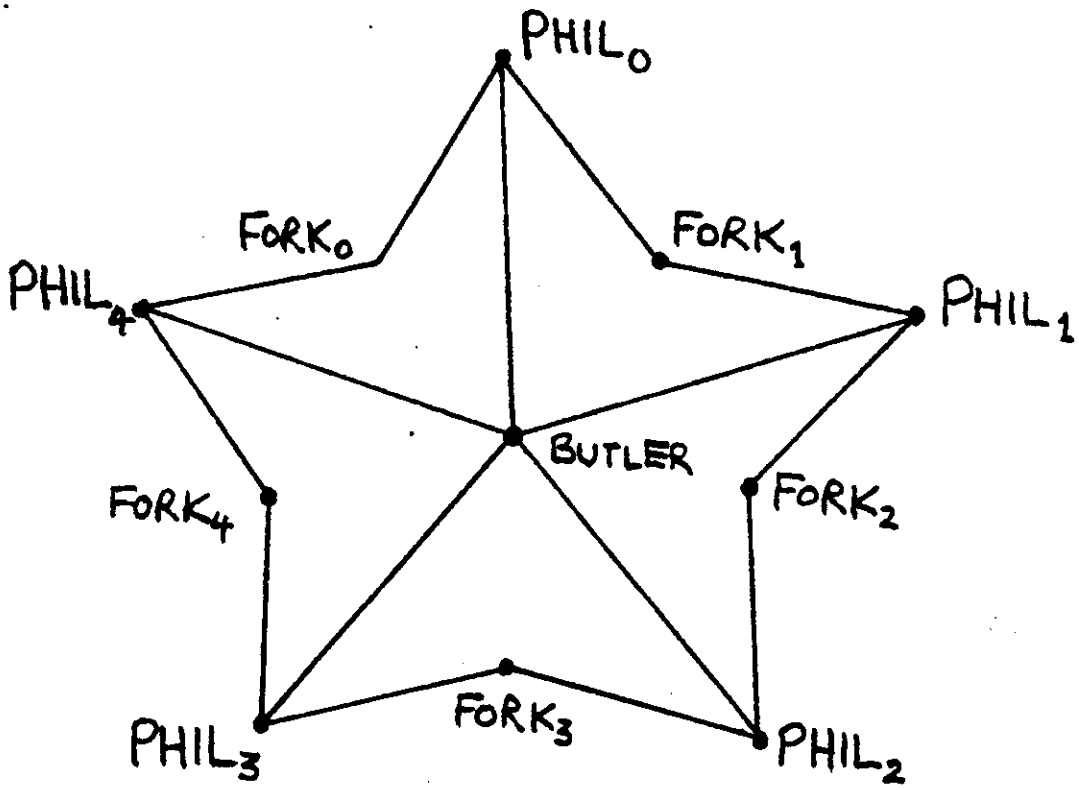
and the process definitions are:

$$\begin{aligned} \text{PHIL}_i &= (\text{i.enters} \rightarrow \text{i.picks.i} \rightarrow \text{i.picks.i+1} \rightarrow \\ &\qquad \text{i.eats} \rightarrow \text{i.puts.i} \rightarrow \text{i.puts.i+1} \rightarrow \text{i.leaves} \rightarrow \text{PHIL}_i), \\ \text{FORK}_i &= (\text{i.picks.i} \rightarrow \text{i.puts.i} \rightarrow \text{FORK}_i) \,\Box\, (\text{i}-1\text{.picks.i} \rightarrow \text{i}-1\text{.puts.i} \rightarrow \text{FORK}_i), \end{aligned}$$

for $i = 0, \ldots, 4$, and

$$\begin{aligned} \text{BUTLER} &= |||_{i=1}^{4} \text{ADMIT}, \\ \text{ADMIT} &= \Box_{i=0}^{4} (\text{i.enters} \rightarrow \text{i.leaves} \rightarrow \text{ADMIT}). \end{aligned}$$

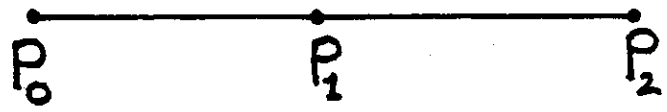
In the definition of the butler process we have used the *interleaving* operation, as described in [3,4,5]. Addition and subtraction of indices is understood to be modulo 5. The system of butler, philosophers and forks corresponds to the following network:



Dining Philosophers

Example 2. A chain of processes.

A simple chain of processes in which each one communicates solely with its immediate predecessor and successor is represented as a particularly simple form of network. The graphical notion of a chain thus corresponds to the use of the CSP chaining operation $\gg$, as defined in [5]. The following diagram illustrates a case when there are 3 processes:



We will be particularly interested in a class of networks whose communication structure is especially simple.

Definition 1. A network V is a *tree* iff each of its connected components has one more vertex than arcs.

We are abusing notation slightly here, because it might be more conventional to term such a network a *forest*. In our terminology, a tree network is a collection of single trees. Trees arise naturally as the communication graphs of networks built with the master-slave operator $[P \parallel \mathrm{m}{:}Q]$. In such a combination P is the master process and it has a slave referred to by the name m. More generally, we can form a combination

$$[P \parallel \mathrm{m}_1{:}Q_1 \parallel \ldots \parallel \mathrm{m}_n{:}Q_n]$$

in which P has n slaves. In the network graph corresponding to this system the root node is the overall master process, and its sons are its immediate slaves. In a tree, each process communicates with its immediate predecessor (master) and with its immediate sons (slaves); the root process also may communicate with its environment. A chain is a special case of a tree. Note also that when recursion is used to define a process the corresponding network may be infinite.

In all of our systems we are considering only two-way communication, so that no single communication can involve more than two processes. This condition corresponds to a simple constraint on alphabets.

Definition 2. An indexed set $\{A_i \mid 1 \leq i \leq n\}$ of alphabets is *triple-disjoint* if $A_i \cap A_j \cap A_k$ is empty whenever i, j, k are distinct.

Definition 3. A network $V = \{(P_i, A_i) \mid 1 \leq i \leq n\}$ is *normal* if its alphabets are triple-disjoint and the traces of each process are generated by the corresponding alphabet:

$$\forall i.\ \mathrm{traces}(P_i) \subseteq A_i^*.$$

Definition 4. A network $V = \{(P_i, A_i) \mid 1 \leq i \leq n\}$ is *unidirectional* if for all i and s there is at most one $j \neq i$ such that

$$\mathrm{initials}(P_i \underline{\ \mathrm{after}\ } s) \cap A_j \neq \emptyset.$$

In a unidirectional system at all times each process is prepared to communicate with at most one other process. Of course, the choice of communication partner may vary during an execution of the system. Many interesting example systems are unidirectional.

Definition 5. A property of networks is *hereditary* if whenever it holds of an entire network it also holds of all non-empty subnetworks.

Note that the properties introduced in Definitions 1, 3 and 4 are obviously hereditary.

Deadlock.

Now we are ready to begin an analysis of the deadlock properties of networks. We say that a process P *can deadlock* after s if $(s, \Sigma) \in \mathcal{F}[\![P]\!]$; this means that the process can refuse all events after performing the sequence s. Conversely, we say that P is *free of deadlock* if for all $s \in \Sigma^*$ we have $(s, \Sigma) \notin \mathcal{F}[\![P]\!]$. Note that this certainly requires that the process be also free of divergence, since in this semantics divergence is regarded as catastrophic [5].

These definitions generalise in the obvious way to a network of processes.

Definition 6. A network V is *free of deadlock* if $\mathrm{PAR}(V)$ is free of deadlock.

Definition 7. A network V is *strongly free of deadlock* if all non-empty subnetworks $U \subseteq V$ are free of deadlock.

Of course, strong freedom from deadlock implies the weaker condition, but the converse is not generally true.

Definition 8. In a network $V = \{(P_i, A_i) \mid 1 \leq i \leq n\}$, a *state* is a trace s of the system together with an indexed set $(X_1, \ldots, X_n)$ of refusal sets X_i such that for each i

$$(s\restriction A_i, X_i) \in \mathcal{F}[\![P_i]\!].$$

The state is *maximal* if each of the refusal sets is maximal. Note that the structure of our model guarantees that each state may be extended to a maximal state. A state is simply a cross-section of the network giving the local information about what each process in the system is refusing on the next step. We will define the *communication graph* of the network in this state as the (directed) graph in which there is an outgoing arc from node i to node j iff $(A_i - X_i) \cap A_j \neq \emptyset$. The following result shows how to characterise the states in which deadlock occurs.

LEMMA 1. *A network $V = \{(P_i, A_i) \mid 1 \leq i \leq n\}$ can deadlock after s iff there is a maximal state $(s, \langle X_1, \ldots, X_n \rangle)$ for which*

$$\bigcup_{i=1}^{n} A_i = \bigcup_{i=1}^{n} (A_i \cap X_i).$$

We will refer to such a (maximal) state as a *deadlock state.*

Definition 9. A *request* in a network $V = \{(P_i, A_i) \mid 1 \leq i \leq n\}$ is a triple of the form

$$\langle u, X, v \rangle, \qquad 1 \leq u, v \leq n, \quad X \subseteq \Sigma,$$

satisfying the condition

$$(A_u - X) \cap A_v \neq \emptyset.$$

This corresponds to a state in which the process P_u at node u wants to participate in an action from the alphabet of the process P_v at node v; X represents a refusal set of P_u, so that $A_u - X$ contains the events P_u may perform next; thus, it may also be possible for the process to perform events requiring the participation of processes other than P_v, depending on the set X. The request is *strong* if

$$\emptyset \neq (A_u - X) \subseteq A_v.$$

In this case, P_u is limited to performing a communication with P_v alone on the next step.

These notions generalise to sequences of requests, having the form

$$\langle u_1, X_1, u_2, X_2, \ldots, X_{n-1}, u_n \rangle.$$

Here we require that each of $\langle u_1, X_1, u_2 \rangle, \langle u_2, X_2, u_3 \rangle, \ldots, \langle u_{n-1}, X_{n-1}, u_n \rangle$ be a request. A sequence is *proper* if all of its nodes are distinct. A sequence is a *cycle* of requests when $u_n = u_1$; a *proper* cycle is one in which (apart from u_1 and u_n) the nodes are distinct. The length of this cycle is $n - 1$, which for a proper cycle is the number of distinct nodes involved.

We may thus speak of a state containing a cycle of requests. A connection between cycles of requests and deadlock is made by the following result. Note that it gives a simple characterisation of the communication graph of a system in a deadlock state: if the system satisfies the conditions of the theorem then deadlock corresponds to a cycle in the communication graph involving at least three distinct nodes.

THEOREM 1. *Let $V = \{(P_i, A_i) \mid 1 \leq i \leq n\}$ be a normal unidirectional network of processes and let $P = PAR(V)$ be their parallel composition. Suppose that the following conditions hold:*

(i) Each P_i is free of deadlock;

(ii) Each pair $[P_i {}_{A_i} \|_{A_j} P_j]$ is free of deadlock;

Then any deadlock state of P contains a proper cycle of strong requests of length at least 3.

Proof.

Let $(s, \langle Y_1, \ldots, Y_n \rangle)$ be a deadlock state of P. Then by Lemma 1,

$$\forall i.\ (s \lceil A_i, Y_i) \in \mathcal{F}[\![P_i]\!], \tag{a}$$

$$\bigcup_{i=1}^{n} A_i = \bigcup_{i=1}^{n} (A_i \cap Y_i). \tag{b}$$

By assumption, the Y_i are maximal refusal sets in (a). For each i let $Q_i = P_i \underline{\text{after}}\, s \lceil A_i$, so that $P \underline{\text{after}}\, s = \|_{i=1}^{n} (Q_i, A_i)$. We argue as follows.

• None of the Y_i can include A_i as a subset, as P_i is deadlock-free by assumption (i) and the system is normal. From (b) we see that for each i,

$$A_i - (\bigcup_{j \neq i} A_j) \subseteq Y_i, \qquad \text{FACT 1}$$

so that in this state of the system each process is refusing all events unique to its own alphabet. Hence,

$$A_i - Y_i \subseteq \bigcup_{j \neq i} A_j. \qquad \text{FACT 2}$$

• By maximality of Y_i we know that Y_i contains all of the *impossible* events from the set $A_i - \text{initials}(Q_i)$:

$$A_i - \text{initials}(Q_i) \subseteq Y_i.$$

Hence,

$$A_i - Y_i \subseteq \text{initials}(Q_i).$$

But there is at most one $j \neq i$ with

$$\text{initials}(Q_i) \cap A_j \neq \emptyset,$$

since the system is unidirectional. Hence there is at most one $j \neq i$ for which

$$(A_i - Y_i) \cap A_j \neq \emptyset. \qquad \text{FACT 3}$$

Putting these facts together, we see that there is a unique j (depending on i) such that $i \neq j$ and

$$\emptyset \neq A_i - Y_i \subseteq A_j.$$

Consider j as a function of i, mapping indices to indices. Note that $j(i) \neq i$, and it also happens that $j(j(i)) \neq i$, because if this were to happen we would have a pair of indices $i, j = j(i)$ with

$$A_i - Y_i \subseteq A_j, \qquad A_j - Y_j \subseteq A_i.$$

But by (b) and (3) we would then have

$$A_i \cap A_j \subseteq Y_i \cup Y_j,$$

which would in turn imply that

$$A_i - Y_i \subseteq Y_j, \qquad A_j - Y_j \subseteq Y_i.$$

Hence, we would get

$$(A_i \cap Y_i) \cup (A_j \cap Y_j) = A_i \cup A_j,$$

contradicting the assumption (ii) that the pair $[P_i \,{}_{A_i}\|_{A_j}\, P_j]$ was free of deadlock.

The sequence

$$1, j(1), j^2(1), \ldots$$

must contain a first repetition, say $j^m(1) = j^{m+k}(1)$, since there are only finitely many indices. Define $n_r = j^{m+r}(1)$, for $r = 1 \ldots k$. Then $\langle n_1, Y_{n_1}, n_2, \ldots, Y_{n_{k-1}}, n_k \rangle$ is a cycle of strong requests. ∎

An intuitive interpretation of this theorem is that global deadlock (*i.e.* deadlock of the entire system) can only be caused in a unidirectional normal system by local deadlock or else by a cycle of at least three distinct nodes each demanding to communicate with its successor and refusing to communicate with its predecessor. Some important consequences of the theorem are:

Corollary. If a tree network satisfies the conditions above, then it is strongly free of deadlock.

Proof. A tree has no proper cycles, and all of the hypotheses of the theorem are hereditary properties. ∎

Corollary. In a unidirectional tree network, pairwise freedom of deadlock implies absence of global deadlock.

If a network has only a small number of cycles, it is often possible to prove absence of deadlock by performing a case analysis. For instance, in a *ring* of processes there are only two possible cycles to consider. To satisfy the preconditions of Theorem 3 we still need to prove pairwise freedom from deadlock. This may often be possible by a simple case analysis, and the amount of work involved in the analysis can often be reduced substantially by using the following lemma:

LEMMA 2. *If* $c \notin A$, $c \notin B$, *and* $C = A \cup \{c\}$, *then* $[P_C \,\|_B\, Q] \backslash c = [(P \backslash c)_A \,\|_B\, Q]$.

If c is an event whose hiding in P does not cause any divergence, then $P \backslash c$ can deadlock if and only if P can. Hence, if hiding c does not cause divergence and if $c \in A - B$, we see that $[P_A \,\|_B\, Q]$ is deadlock-free if and only if $[(P \backslash c)_A \,\|_B\, Q]$ is. This concealment of "irrelevant" communication can substantially reduce the complexity of the deadlock analysis. By hiding irrelevant communication we can reduce the amount of detail still further.

Here is an example to illustrate this type of reasoning.

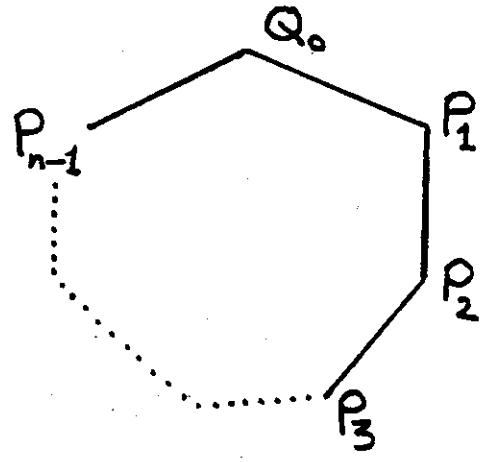
Example 9. Privilege Rings (after Dijkstra [11]).

We consider a ring of n processes ($n \geq 3$) each of which wants to keep entering a critical section; to maintain mutual exclusion, a process is only allowed to enter its critical region when it has obtained a "privilege" token, which is passed around the ring. When a process wants to begin its critical region it first requests the privilege from its neighbour; when it is granted the privilege, the process performs its critical action and then releases the privilege token. Using mutual recursion, we may define the individual processes P_i ($i < n$) by

$$\begin{aligned} P_i &= (\text{i.get} \rightarrow \text{i+1.find} \rightarrow \text{i.priv} \rightarrow \text{i.crit} \rightarrow \text{i.rel} \rightarrow Q_i) \\ &\quad \Box\ (\text{i.find} \rightarrow \text{i+1.find} \rightarrow \text{i.priv} \rightarrow \text{i–1.priv} \rightarrow P_i) \\ Q_i &= (\text{i.get} \rightarrow \text{i.crit} \rightarrow \text{i.rel} \rightarrow Q_i) \Box (\text{i.find} \rightarrow \text{i–1.priv} \rightarrow P_i). \end{aligned}$$

All arithmetic here is modulo n. P_i represents a node without the privilege token and Q_i represents a node with the privilege. Thus, if P_i wants to get the privilege it must put in a request first to its successor, and wait for that process to find the token and pass it back; if P_i is asked to find the privilege it passes the request on to its neighbour. A Q_i process with the token may either allow the critical action or pass the token on to its predecessor.

Let A_i be the obvious alphabets for these processes. If we begin with the system $V = \{(Q_0, A_0), (P_1, A_1), \ldots, (P_{n-1}, A_{n-1})\}$, so that initially the 0 process has the token, we would like to prove that PAR(V) is free of deadlock. Here is the network graph for the system.



It is easy to see that the system is normal and unidirectional. It remains to prove pairwise freedom from deadlock. We must show that each of the pairs

$$\begin{aligned} &(Q_0, A_0) \parallel (P_1, A_1), \\ &(P_i, A_i) \parallel (P_{i+1}, A_{i+1}) \qquad (0 < i < n), \\ &(P_{n-1}, A_{n-1}) \parallel (Q_0, A_0) \end{aligned}$$

is deadlock-free. These analyses are simplified by judicious use of Lemma 2, as follows. We can hide (without introducing divergence) the set $L_i = \{$ i.get, i.rel, i.find, i−1.priv $\}$ in process P_i or Q_i when the process appears on the left of $\|$ in this list; and the same holds for the set $R_i = \{$ i.crit, i.rel, i+1.find, i.priv $\}$ when on the right. Let $P_i^L = P_i \backslash L_i$ and $Q_i^L = Q_i \backslash L_i$, with similar notation P_i^R and Q_i^R for the other hiding operation. Also, let $A_i^L = A_i - L_i$, with similar notations for the R versions. We have, by definition, and using properties of the hiding operation,

$$\begin{aligned} P_i^L &= \text{i+1.find} \rightarrow \text{i.priv} \rightarrow Q_i^L, \\ Q_i^L &= (\text{i.crit} \rightarrow Q_i^L) \sqcap P_i^L, \\ Q_i^R = P_i^R &= (\text{i.find} \rightarrow \text{i-1.priv} \rightarrow P_i^R) \,\Box\, (\text{i.get} \rightarrow P_i^R). \end{aligned}$$

Using Lemma 2, now we have to prove freedom from deadlock in the pairs:

$$\begin{aligned} &(Q_0^L, A_0^L) \,\|\, (P_1^R, A_1^R), \\ &(P_i^L, A_i^L) \,\|\, (P_{i+1}^R, A_{i+1}^R), \\ &(P_{n-1}^L, A_{n-1}^L) \,\|\, (Q_0^R, A_0^R). \end{aligned}$$

This may be done by a simple case analysis. Therefore we can use the above theorem to deduce the existence, in any deadlock state, of a cycle. Thus, deadlock is possible only if either each process is waiting for its successor or each process is waiting for its predecessor. We can easily formalise and prove the property that there is always exactly one process with the token: either process 0 still has it and no other process has been granted it, or else exactly one process has been given the privilege but has not yet released it. But process i can only be waiting for process $i+1$ if it does not have the token; and, similarly, a process can only be waiting for its predecessor if it does not have the token. It follows that deadlock is impossible.

A modification of this argument goes through whenever the system is started with at least one token in the ring. Of course, in the case where no process initially has the privilege, deadlock must occur.

Example 4. A deadlocked chain.

Define a chain of $n+1$ processes for any $n > 1$ as follows. The processes will be $P_0, \ldots, P_n$, with alphabets A_i given by

$$\begin{aligned} A_0 &= \{\text{1.a}\}, \\ A_i &= \{\text{i.a}, \text{i.b}, \text{i+1.a}, \text{i+1.b}\}, \qquad 1 \leq i < n, \\ A_n &= \{\text{n.b}\}. \end{aligned}$$

The process P_0 wants to keep receiving message a from process P_1, and P_n wants to keep receiving message b from P_{n-1}. Each of the intermediate processes $P_1, \ldots, P_{n-1}$

can repeatedly transmit a from its right to its left, and b from its left to its right. The definitions are:

$$\begin{aligned} P_0 &= (1.a \rightarrow P_0) \\ P_n &= (n.b \rightarrow P_n) \\ P_i &= T_i^a \,|||\, T_i^b, \qquad (1 < i < n), \\ T_i^a &= (i{+}1.a \rightarrow i.a \rightarrow T_i^a), \\ T_i^b &= (i.b \rightarrow i{+}1.b \rightarrow T_i^b). \end{aligned}$$

In the mixed parallel composition corresponding to this system, no event is initially possible. The reason is easy to see: although each of the interior processes in the chain is attempting to perform one of two actions, each of those actions is also in the alphabet of a neighbour who is not willing to cooperate. And although the end processes are trying to receive a message from their neighbours, their neighbouring processes are not initially able to send the message. The chain therefore deadlocks immediately. However, it is not difficult to see that every non-empty proper subnetwork is deadlock-free! This example shows that there exist chains in which deadlock is a global property.

3. A more general result.

The above theorem is only applicable when the system is unidirectional. More general results are needed to tackle some other interesting examples such as the Dining Philosophers. A useful technique is based on the concept of *competition* ("cliques" in the terminology of [15]).

Define a competition relation COMP on triples of nodes of V:

$$\begin{aligned} (i,j,k) \in \mathrm{COMP} \quad \Leftrightarrow \quad & i \neq j \;\&\; j \neq k \;\&\; k \neq i \\ & \&\; \exists s.(\mathrm{initials}(P_i \underline{\text{ after }} s) \cap A_j \neq \emptyset \;\&\; \mathrm{initials}(P_i \underline{\text{ after }} s) \cap A_k \neq \emptyset) \end{aligned}$$

Intuitively, $(i,j,k) \in \mathrm{COMP}$ iff at some stage P_j and P_k can find themselves in competition for the attention of P_i. We use the notation $\mathrm{COMP}(i,j) = \{\, k \mid (i,j,k) \in \mathrm{COMP} \,\}$. Thus, $k \in \mathrm{COMP}(i,j)$ means that P_k may compete at some time with P_j for the attention of P_i.

When $i \neq j$ we define $\mathrm{comp}_V(i,j)$ to be the smallest subset C of V containing i and j, and closed under COMP:

$$k, l \in C \;\&\; (k,l,m) \in \mathrm{COMP} \quad \Rightarrow \quad m \in C.$$

Where V is obvious from the context we will omit the subscript. The set $\mathrm{comp}(i,j)$ is an "upper bound" on the set of processes which might interfere or compete directly with communications between P_i and P_j, in that this set contains (at least) all of the processes which may dynamically find themselves competing with a communication between P_i and P_j. We will refer to $\mathrm{comp}(i,j)$ as the *competition set* of processes i and j, and to the

members of this set other than i and j as *competitors*. Note that the definition depends on the processes as well as on the communication structure of the network, since it takes into account the dynamic behaviour of the processes and not just the static communication pattern of the alphabets. Of course, if $k \in \mathrm{comp}(i,j)$ then there must be an arc between P_i and P_k and between P_j and P_k. The converse is not true in general.

Examples.

1. In the dining philosophers system (*Example 1*), two adjacent philosophers can find themselves competing for the same fork, and any two philosophers can compete for entry into the room via the butler process. The only interesting competition sets are

$$\mathrm{comp}(\mathrm{PHIL}_i, \mathrm{FORK}_i) = \mathrm{comp}(\mathrm{PHIL}_{i+1}, \mathrm{FORK}_i) = \{\mathrm{PHIL}_i, \mathrm{FORK}_i, \mathrm{PHIL}_{i+1}\},$$
$$\mathrm{comp}(\mathrm{PHIL}_i, \mathrm{BUTLER}) = \{\mathrm{PHIL}_0, \ldots, \mathrm{PHIL}_4, \mathrm{BUTLER}\},$$

for $i = 0 \ldots 4$. This is as expected: neighbouring philosophers can compete for the fork situated between them, and all philosophers compete for the attention of the butler.

2. In the privilege ring example (*Example 3*), there are no non-trivial competition sets.

3. In the deadlocked chain example (*Example 4*), the only whole system is a competition set, since any adjacent pair P_{i-1} and P_{i+1} can compete for the process P_i in between them.

4. In general, the competition sets of a tree network are particularly simple in structure. There are no nontrivial competitors for any two non-adjacent nodes. If nodes i and j are adjacent then $\mathrm{comp}(i,j)$ forms a connected subtree of the network. Moreover, any two of these nontrivial competition sets either coincide or else have at most a single node in their intersection.

Of course, these properties of trees do not hold of general networks. The following simple property of competition sets, true in all networks, will be useful.

LEMMA 3. *If $U \subseteq V$ and $i, j \in U$, then*

$$comp_U(i,j) \subseteq comp_V(i,j).$$

Other properties of competition sets may be established in a fairly simple manner, and will be used without proof. Further details appear in [15]. Note in particular that

$$\mathrm{comp}(i,j) = \mathrm{comp}(j,i),$$
$$k \in \mathrm{comp}(i,j) \Rightarrow \mathrm{comp}(i,k) \subseteq \mathrm{comp}(i,j),$$
$$A_i \cap A_j = \emptyset \Rightarrow \mathrm{comp}(i,j) = \{i,j\}.$$

The use of competition sets in deadlock analysis is illustrated in the following result.

THEOREM 2. *For any normal network* $V = \{(P_i, A_i) \mid 1 \leq i \leq n\}$, *if every competition set* $U \subseteq V$ *is strongly free of deadlock, then any deadlock state* $(s, (X_1, \ldots, X_n))$ *of* V *must contain a cycle of requests not entirely contained in any single competition set, and satisfies the additional condition:*

$$\forall i, j.[i \neq j \Rightarrow \exists k \in comp(i,j).\ (A_k - X_k) \subseteq \bigcup\{A_r \mid r \notin comp(i,j)\}].$$

Proof. Similar to that of Theorem 1. ∎

This theorem places a further constraint on the structure of a deadlocking cycle of requests: that it must contain nodes from at least two distinct competition sets. This makes it easier to analyse the network for possible deadlocks, since there are usually fewer cycles satisfying this condition and consequently fewer cases to check. The additional condition also helps to reduce the number of cycles to consider; it says that every pair of processes has a competitor who wants to communicate only outside of the competition set.

THEOREM 3. *For any normal tree* V, *if every competition set* $U \subseteq V$ *is strongly free of deadlock then* V *is strongly free of deadlock.*

Proof. By the previous theorem, deadlock is possible for V after a trace s only if there are (maximal refusal) sets $X_1, \ldots, X_n$ satisfying the conditions:

$$(s\lceil A_i, X_i) \in \mathcal{F}[\![P_i]\!], \tag{1}$$

$$\bigcup_{i=1}^{n} A_i = \bigcup_{i=1}^{n} (A_i \cap X_i), \tag{2}$$

$$\forall i, j.[i \neq j \Rightarrow \exists k \in \mathrm{comp}(i,j).\ (A_k - X_k) \subseteq \bigcup\{A_r \mid r \notin \mathrm{comp}(i,j)\}], \tag{3}$$

and such that this state $(s, (X_1, \ldots, X_n))$ includes a cycle of requests which does not lie wholly inside a single competition set. The indices of this cycle, say $i_1, \ldots, i_k$, satisfy the conditions:

$$(A_{i_r} - X_{i_r}) \cap A_{i_{r+1}} \neq \emptyset$$

for each r (counting modulo k). We claim that any such sequence must lie entirely within a single competition set; in particular, we will show that

$$\{i_1, \ldots, i_k\} \subseteq \mathrm{comp}(i_1, i_2). \tag{4}$$

This will provide us with a contradiction, and therefore allow us to conclude that deadlock is impossible, given the preconditions of the theorem.

The result (4) is proved by induction on the size of the cycle, k. If $k = 2$ there is nothing to prove. Assume true for all cycles of requests of length less than k. There are two cases: either i_1 is repeated later in the sequence $i_2, \ldots, i_k$, or else there is no repetition.

• In the latter case, when i_1 is not repeated, we can see from the tree structure that $i_2 = i_k$, so that $\langle i_2, X_{i_2}, \ldots, i_{k-1}, X_{i_{k-1}}, i_k \rangle$ is a cycle of requests for the processes $\langle i_2, \ldots, i_k \rangle$. Thus, by inductive hypothesis we have

$$\{ i_2, \ldots, i_k \} \subseteq \mathrm{comp}(i_2, i_3).$$

Let $Q_i = P_i$ after $s \lceil A_i$. By construction we have

$$\begin{aligned} \mathrm{initials}(Q_{i_2}) \cap A_{i_3} &\neq \emptyset, \\ \mathrm{initials}(Q_{i_2}) \cap A_{i_1} &\neq \emptyset, \end{aligned}$$

because $i_2 = i_k$. We also know that

$$(A_{i_2} - X_{i_2}) \cap A_{i_3} \neq \emptyset, \quad (A_{i_k} - X_{i_k}) \cap A_{i_1} \neq \emptyset,$$

and

$$(A_{i_2} - X_{i_2}) \subseteq \mathrm{initials}(Q_{i_2}), \quad (A_{i_k} - X_{i_k}) \subseteq \mathrm{initials}(Q_{i_k}).$$

This latter is by maximality of the X_i as in the proof of Theorem 1. Hence, $(i_2, i_1, i_3) \in r$, and $i_3 \in \mathrm{comp}(i_1, i_2)$. It follows that

$$\{ i_1, \ldots, i_k \} \subseteq \mathrm{comp}(i_1, i_2).$$

• In the other case, when i_1 is repeated, we have for some $1 < r < k$, $i_r = i_1$. Clearly there are two cycles of requests:

$$\begin{aligned} &\langle i_1, X_{i_1}, \ldots, i_{r-1}, X_{i_{r-1}}, i_r \rangle \\ &\langle i_r, X_{i_r}, \ldots, i_k, X_{i_k}, i_1 \rangle. \end{aligned}$$

By the inductive hypothesis,

$$\begin{aligned} \{ i_1, \ldots, i_r \} &\subseteq \mathrm{comp}(i_1, i_2) \\ \{ i_r, \ldots, i_k \} &\subseteq \mathrm{comp}(i_r, i_{r+1}) = \mathrm{comp}(i_1, i_{r+1}). \end{aligned}$$

However, by construction,

$$\mathrm{initials}(Q_{i_1}) \cap A_{i_2} \neq \emptyset \quad \text{and} \quad \mathrm{initials}(Q_{i_1}) \cap A_{i_{r+1}} \neq \emptyset,$$

as before, so that $i_{r+1} \in \mathrm{comp}(i_1, i_2)$. Thus, $\mathrm{comp}(i_1, i_{r+1}) \subseteq \mathrm{comp}(i_1, i_2)$, from which the result follows. Having established the truth of (4), that completes the proof of this theorem. ∎

Example: Absence of deadlock in the dining philosophers system.

We argue along the following lines, although we will omit details. Suppose that deadlock is possible, and consider the communication graph of the system in a deadlock state. We will reach a contradiction.

- If there is an edge from PHIL_i to FORK_i then FORK_i must have a unique outgoing edge, to PHIL_{i-1}; if PHIL_{i-1} has an edge leading to FORK_i then there is a unique outgoing edge, to PHIL_i.

- By looking at the parallel composition of PHIL_i and FORK_i, we see that: whenever FORK_i has a unique outgoing edge leading to PHIL_i, then PHIL_i has a unique outgoing edge leading to FORK_{i+1}. Similarly, whenever FORK_{i+1} has a unique outgoing edge to PHIL_i, then PHIL_i has a unique outgoing edge to FORK_i.

- By Theorem 2, there must be an edge leading out from the competition set $\{\mathrm{PHIL}_0, \ldots, \mathrm{PHIL}_4, \mathrm{BUTLER}\}$. By the previous two properties this means that there must be a cycle of requests linking all of the philosophers and forks. Moreover, the edges of this cycle must be the only edges leading from these processes, so that the butler process is not involved.

- Use the definition of the butler process to demonstrate that this situation can never arise, by a simple counting argument: the butler is designed to ensure that no more than four philosophers can ever be seated simultaneously. In any trace of the system the number of *enter* events cannot exceed the number of *leave* events by more than 4.

4. Comparison with other work.

Many authors have worked on deadlock analysis in networks of processes, notably [6,7,8]. Our model has the advantage of providing a succinct and mathematically tractable representation of deadlock. We have been able to use the model in proofs of some interesting results on the analysis of deadlock in networks, and then to prove absence of deadlock in some well known examples such as the Dining Philosophers. The theorems of this paper are only a sample of a large class of general results which we will be able to derive for analysing the deadlock properties of networks. We have focussed here mainly on results pertaining to unidirectional systems and trees of processes. Dijkstra [8] proved some similar theorems on the absence of deadlock in unidirectional networks for the special case in which the patterns of communication were cyclic: each process rotated its communication requests in cyclic order through its immediate neighbours. Dijkstra stated that his results were applicable in a more general setting, and we have demonstrated that this is indeed the case. We also hope that we may be able to represent some of the results obtained by Chandy and Misra [6] in our setting. Other results related to ours are contained in [14],

where Reisig considers a Petri net model and gives a characterization of deadlock states. The work of Apt et al. [1,2] on reasoning about partial correctness of CSP programs also contains some methods for analysing deadlock, using global invariants. Essentially, this work is based on a rather different approach from ours: typically, a CSP program is first transformed syntactically into a program in a *guarded command* language [10] which no longer involves communication, and matching pairs of communications from the original program are transformed (synchronized) into assignment statements. Then one reasons about the absence of deadlock by finding a global invariant which guarantees that no deadlock state can be reached, because it is false in deadlock states.

5. Future work.

As we stated earlier, the deadlock analysis theorems of this paper exemplify a class of general results on deadlock in networks of parallel processes. Theorem 1 concerns unidirectional networks, and Theorem 3 is applicable to tree networks. We hope to prove more results along these lines in the future and to use them to analyse the properties of a larger number of classes of networks. For instance, our definition of competition set provided a fairly crude bound on the set of processes which could be involved in causing a local deadlock; a more careful analysis may lead to sharper results. It would be especially useful to be able to refine the results on competition sets so as to minimise the amount of local checking that is required in order to establish freedom from deadlock. All the same, the competition set analysis described in this paper does seem to be fairly useful. Sharper results, although less generally applicable, may be obtained by focussing instead on a notion of *conflict* with respect to a set of events. Roughly speaking, two processes can conflict over a set C of events if they can reach a state in which they can only perform events in C next, each is requesting the other process to do something, but they cannot agree on any communication. Although we do not elaborate here, this notion can be used to modify the statement of Theorem 3 and its corollaries, so that slightly different hypotheses are used in deadlock analysis. This material will be developed further in future work and will appear in an extended version of this paper.

Another important topic for future work is the proof of absence of proper cycles of requests in networks. This will be vital if we are to apply these techniques to networks with large numbers of possible cycles. It seems likely that in many cases this will involve the discovery of global invariants and associated properties of the underlying graphs, and again we see the possibility of some connections with the ideas of [1,2].

Acknowledgements.

The authors would like to thank C. A. R. Hoare for his many helpful suggestions and discussions, and for his encouragement and guidance during the development of this work. Discussions with Krzysztof Apt, Jay Misra, Ernst-Rudiger Olderog, David Reed and Wolfgang Reisig have been very useful.

6. References.

[1] Apt, K. R., A Static Analysis of CSP Programs, in: Logics of Programs, Proceedings, Springer Verlag LNCS vol. 164, pp. 1-17 (1983).

[2] Apt, K. R., Francez, N., and de Roever, W. P, A proof system for communicating sequential processes, TOPLAS vol. 2 no. 3, pp. 359-385 (1980).

[3] Brookes, S. D., Hoare, C. A. R., and Roscoe, A. W., A Theory of Communicating Sequential Processes, Oxford University Computing Laboratory, Programming Research Group, Technical Report PRG-16.

[4] Brookes, S. D., Hoare, C. A. R., and Roscoe, A. W., A Theory of Communicating Sequential Processes, JACM July 1984.

[5] Brookes, S. D., and Roscoe, A. W., An Improved Failures Model for Communicating Processes, Proc. NSF–SERC Seminar on Concurrency, Springer Verlag LNCS (to appear, 1985).

[6] Chandy, K. M., and Misra, J., Deadlock Absence Proofs for Networks of Communicating Processes, Information Processing Letters, Vol. 9, no. 4, Nov. 1979.

[7] Chandy, K. M., Misra, J., and Haas, L. M., Distributed Deadlock Detection, ACM TOPLAS Vol. 1 no. 2, pp 144-156 (1983).

[8] Dijkstra, E. W., A Class of Simple Communication Patterns, EWD643, in: Selected Writings on Computing, Springer Verlag (1982).

[9] Dijkstra, E. W., Hierarchical Ordering of Sequential Processes, Acta Informatica 1, pp 115-138 (1971).

[10] Dijkstra, E. W., Guarded Commands, Nondeterminacy and Formal Derivation of Programs, CACM, Vol. 18 No. 8, August 1975.

[11] Dijkstra, E. W., Invariance and non-determinacy, in: Mathematical Logic and Programming Languages, C. A. R. Hoare and J. C. Shepherdson, eds., Prentice-Hall International Series in Computer Science (1985).

[12] Hoare, C. A. R., Communicating Sequential Processes, CACM 1978.

[13] Milne, G., and Milner, R., Concurrent Processes and their Syntax, JACM vol 26 no. 2, pp 302-321 (1979).

[14] Reisig, W., Deterministic Buffer Synchronization of Sequential Processes, Acta Informatica 18, 117-134 (1982).

[15] Roscoe, A. W., A Mathematical Theory of Communicating Processes, Ph. D. thesis, Oxford University (1982).