The Pursuit of Deadlock Freedom

by A.W. Roscoe and Naiem Dathi

**Oxford University Computing Laboratory,
8-11 Keble Road, Oxford OX1 3QD, U.K.**

Abstract *We introduce some combinatorial techniques for establishing the deadlock freedom of concurrent systems which are similar to the variant/invariant method of proving loop termination. Our methods are based on the local analysis of networks, which is combinatorially far easier than analysing all global states. They are illustrated by proving numerous examples to be free of deadlock, some of which are useful classes of network.*

1 Introduction

Deadlock occurs in a concurrent network when no further action can take place. This is usually because, even though each component process is in a state in which it can communicate, its potential communications are blocked by its neighbours. This is a common problem in concurrent systems and is unique to them. A proof of deadlock freedom for such a system is an integral part of a total correctness proof, and is often a desirable first step towards the latter.

Unfortunately, the introduction of concurrency not only introduces the possibility of pathological behaviour such as deadlock, but it also makes systems harder to understand and analyse. Because the components of a concurrent system can often act independently, there is no predetermined sequential order for its various actions. This means that systems can exhibit real *nondeterminism*, or unpredictability: just because a system passes a test once does not mean that it will always do so.

Ignoring the values of its variables, the number of control states in a sequential program increases linearly with the number of lines (the program can be "at" any one line). With concurrent programs this growth becomes exponential: the program can be "at" one line in each of its parallel components.

This observation means that any method of checking for deadlock that involves inspecting the global states of a network is likely to be very unattractive. This paper, which continues the work begun in [BR2,3], describes some methods of deadlock analysis which only involve very local analysis of networks: usually only single processes and pairs of processes which communicate directly. These lead to rules for proving the absence of deadlock which are reasonably easy to apply in practice, and also to ways of devising networks which are deadlock free by construction. The methods we describe are not *complete*, for it is possible to construct examples which are deadlock free for very subtle and non-local reasons; it seems that a complete proof rule must involve fairly exhaustive checking of global states.

The techniques introduced in this paper are closely related to the idea of using a "variant" to prove termination of a loop. The wide applicability of these techniques is illustrated by several examples. Some of these examples are fairly general classes of network and establish some easy to apply design rules for building networks (of certain types) which are deadlock free by design.

In the next section we see how networks of processes are composed and learn how deadlock is represented. Then the simplest version of our variant technique is described and illustrated by examples. In later sections, we see how the results of [BR3] allow us to derive more powerful versions.

We assume a certain familiarity with the version of CSP described in [H], [BHR] and [BR1], and the basic properties of its operators. The mathematics of this paper, like that of [BR2,3], is based on the *failures* model for CSP described in [BR1] (which is an improved version of that of [BHR]). As we shall see, this model has a very simple representation of deadlock. Familiarity with the failures model will be helpful in reading this paper, though its basic structure is described below. The semantics of CSP we use is that of [BR1] and [H], though the definition of the most important operator for the present paper, namely the one for composing networks of processes in parallel, is given below. Even though we have cast our work in this particular framework, we imagine that it will transfer readily to other models of concurrency.

In the failures model a process is modelled as a pair $\langle F, D\rangle$. Here F is a set of failures or pairs (s, X), s being a *trace* (finite sequence of communications) and X being a *refusal set* (set of communications). $(s, X) \in F$ if the process can communicate the elements of s in sequence and then fail to communicate if its environment offers the set X for it to choose from. D is the set of traces on which the process might diverge. A process *diverges* when it performs an unbroken infinite series of internal actions. In a concurrent system where internal communications are hidden for the environment this will often take the form of *livelock*, where the components of the network communicate infinitely among themselves without ever performing an external communication.

From the point of view of the user, a diverging process is deadlocked because it will never communicate with him again. (In fact it is worse because the user can never detect that this is the case.) However, divergence is operationally quite different from deadlock: this means that different techniques are required for analysing a system for potential divergence and deadlocks. We will therefore assume that all the processes we meet are already known to be divergence free. Thus each CSP process P we consider is completely described by its set of failures $\mathcal{F}[\![P]\!]$ (sometimes written $\mathit{failures}(P)$).

A NOTE ON ALPHABETS The CSP parallel operator depends crucially on the idea of an *alphabet*: each process in a network has an alphabet of communications, and no communication can occur until all the processes with it in their alphabets are willing to participate.

Two methods have been used in the literature for introducing these alphabets. In, for example, [BHR] and [BR1,2] these were introduced explicitly into the parallel operator: thus $(P_A\|_B Q)$ was the parallel composition of P and Q, with alphabets A and B respectively. However in [H], *all* processes are defined in such a way that they have an alphabet (usually, though not invariably, the set of communications which they might wish to make). Thus there is no need to introduce explicit alphabets in the parallel operator. In this paper we will follow [H]. The alphabet of the process P will be denoted αP. The traces, refusal sets and divergence traces of P are all composed exclusively of elements of αP.

2 Deadlock and networks

This section summarises the basic facts we will need to know about the representation of deadlock and networks.

A process P can deadlock after the trace s if and only if $(s, \alpha P) \in \mathcal{F}[\![P]\!]$. Thus a process is deadlocked exactly when it must reject everything the environment can offer it. This explains the following definition.

Definition. A process P is said to be *deadlock free* if and only if

$$\forall s \in (\alpha P)^*.\ (s, \alpha P) \notin \mathcal{F}[\![P]\!]\ \ .\ \Box$$

This is a very clear formulation of deadlock freedom and, because of its simplicity, it is the one we shall use. In practice one might wish to complicate it slightly by allowing a process that has already terminated successfully to do nothing. In order to accomodate this more complex form, small modifications would have to be made to most of the definitions and results that follow. But none of the example processes we use can ever terminate, so we will not make these modifications.

The following two laws are often very helpful in establishing that a process is deadlock free. D1 is useful in establishing the deadlock freedom of the individual processes that make up networks. It will allow us to deduce the deadlock freedom of all the component processes of our example networks.

(D1) Suppose the definition of the process P uses only the following syntax:

$$P ::= \mathit{SKIP} \mid a \rightarrow P \mid P;Q \mid P \Box Q \mid P \sqcap Q \mid f(P) \mid p \mid \mu p.P$$

(where p denotes a process variable), but P contains no free process variables. If further P is divergence free, and has every occurrence of SKIP directly or indirectly followed by a ";" (to prevent successful termination), then the process is deadlock free.

(D2) If $P \backslash C$ is divergence free, then it is deadlock free if and only if P is.

D2 observes that, as far as deadlock is concerned, it does not matter whether a process' possible action is external or internal: provided it has *any* action available, it is not deadlocked.

Let us now turn our attention to cases where many processes are working in parallel. If, for each $i \in \{1, 2, \ldots, n\}$, P_i is a process, then we denote their parallel composition by

$$\mathop{\Vert}_{i=1}^{n} P_i \qquad \text{or} \qquad P_1 \Vert P_2 \Vert \ldots \Vert P_n \,.$$

The alphabet of $\Vert_{i=1}^{n} P_i$ is $\bigcup_{i=1}^{n} \alpha P_i$, and a communication a only occurs when every P_i such that $a \in \alpha P_i$ executes it. Thus the communications which lie in more than one αP_i can be regarded as communications between the relevant P_i, while those that are in only one αP_i represent that P_i's communications with the environment. When the P_i are all divergence free this operator is defined:

$$\mathcal{F}[\![\mathop{\Vert}_{i=1}^{n} P_i]\!] = \{(s, \bigcup_{i=1}^{n} X_i) \mid s \in (\bigcup_{i=1}^{n} \alpha P_i)^* \wedge (s \upharpoonright \alpha P_i, X_i) \in \mathcal{F}[\![P_i]\!]\},$$

where $s \upharpoonright \alpha P_i$ denotes the sequence formed from s by removing all elements not in αP_i. If one or more of the P_i can diverge the definition is slightly longer. Note that because P_i must co-operate in every communication in αP_i, if it can refuse something in αP_i, so can the whole process.

The parallel operator is associative in that, when $1 \leq n < m$,

$$(\mathop{\Vert}_{i=1}^{n} P_i) \Vert (\mathop{\Vert}_{i=n+1}^{m} P_i) = \mathop{\Vert}_{i=1}^{m} P_i$$

and symmetric, in that $P \| Q \; = \; Q \| P$.

A *network* V is an indexed set $\{P_i \mid 1 \leq i \leq N\}$ $(N \geq 1)$, where each P_i is a process. The corresponding process $\|_{i=1}^{n} P_i$ is denoted $PAR(V)$. We say that V is deadlock free if $PAR(V)$ is. Note that, in view of the associative and symmetric properties of $\|$, we have

$$PAR(V) \; = \; PAR\{PAR(U_i) \mid 1 \leq i \leq M\}$$

whenever $\{U_1, \ldots, U_M\}$ is a partition of V.

We will generally restrict our attention to networks where no event requires the participation of more than two processes: thus all communication is point to point. Such networks, where $\alpha P_i \cap \alpha P_j \cap \alpha P_k = \emptyset$ whenever i, j and k are all distinct, will be termed *triple-disjoint*.

A *state* of a network $V = \{P_i \mid 1 \leq i \leq N\}$ is a pair $\sigma = \langle s, \langle X_1, \ldots, , X_N \rangle\rangle$ (which we will sometimes abbreviate $\langle s, \underline{X} \rangle$) such that $s \in (\bigcup_{i=1}^{N} \alpha P_i)^*$ and $(s \upharpoonright \alpha P_i, X_i) \in \mathcal{F}[\![P_i]\!]$ for every i. Since the more each individual process refuses, the more likely deadlock becomes, it is sufficient, when considering potential deadlocks, to consider states where each X_i is maximal in *refusals*$(P_i \, \underline{after} (s \upharpoonright \alpha P_i))$. Therefore, throughout this paper, we will assume for convenience that all states have this form. We will denote the maximal failures of a process P, in the sense above, by $\tilde{\mathcal{F}}[\![P]\!]$.

A state of a network shows us what each individual process is refusing. Clearly every maximal failure of a divergence free network corresponds to some state and vice-versa. In particular the network can deadlock on trace s if and only if there is a state $\delta = \langle s, \langle X_1, \ldots, X_N \rangle\rangle$ such that

$$\bigcup_{i=1}^{N} X_i \; = \; \bigcup_{i=1}^{N} \alpha P_i \, .$$

Such states will be termed *deadlock states*.

With every network we may associate a graph, termed the *communication graph*, in the following way. Each process P_i identifies a unique node in the graph, and there is an edge between P_i and P_j $(i \neq j)$ if and only if $\alpha P_i \cap \alpha P_j \neq \emptyset$. Thus two process are joined in the graph if and only if there is the possibility of communication between them.

If W is a not necessarily strict, nonempty subset of the network V, we will term it a *subnetwork*. Notice that some of the communications of the elements of W which were with other elements of V may become external communications of W (i.e., communications of a single process with the environment). We shall say that a property of a network is *hereditary* if it holds of all its subnetworks. For example, a network is hereditarily deadlock free if none of the processes represented by its subnetworks (including itself) can deadlock. Notice, for example, that any network which is triple-disjoint is hereditarily triple-disjoint.

The *vocabulary* of the network $V = \{P_i | 1 \leq i \leq N\}$ is defined to be $\bigcup\{\alpha P_i \cap \alpha P_j | i \neq j\}$: the set of V's "internal communications". This is an important set from the point of view of deadlock analysis because it is the set of events for which agreement is necessary. At any time when V is deadlocked it is clear that no P_i can be willing to communicate outside this set. Note that if W is a subnetwork of V, then the vocabulary of W is a subset of that of V.

The parallel operator we have defined does not conceal the internal communications of a network: in $PAR(V)$ a communication in the language of V remains visible to the environment. However it is often desirable to conceal the internal communications; this is done, for example, in the chaining operator "$\gg$" of CSP. Indeed it is common practice (as is the case with $\gg$) to hide internal communications as the network is put together, so that networks become interleavings of the parallel and hiding operators. We already know, by (D2) above, that hiding the internal communications would make no difference if done at the outermost syntactic level. The following law shows that it does not matter whether the internal actions of an element of a network are hidden immediately or at the outermost level.

(D3) If $C \cap \alpha Q = \emptyset$, then $(P \backslash C \parallel Q) \ = \ (P \parallel Q) \backslash C$.

Using this law, the associative law for $\parallel$ and perhaps some renaming of internal communications, *any* finite parallel system of processes with internal events hidden at any stage can be proved equivalent to a network with a single hiding operation at the outermost level. Thus, from the point of view of deadlock analysis, all such networks are equivalent to a network without any hiding at all.

We will say that the network $V = \{P_i \mid 1 \leq i \leq N\}$ is *busy* if and only if each component process P_i is deadlock free.

We will concentrate on the deadlock analysis of triple-disjoint, busy networks. In such networks deadlock can only occur when every process is willing to communicate with some neighbour or neighbours, but none of these neighbours is willing to respond. These local situations can easily be detected in the global states of a network.

Definition. Suppose $\sigma = \langle s, \underline{X} \rangle$ is a state of the network $V = \{P_i \mid 1 \leq i \leq N\}$. Then we call $\langle i, j \rangle$ a *request* (respectively a *strong request*) of the state if $i \neq j$ and $(\alpha P_i - X_i) \cap \alpha P_j \neq \emptyset$ (respectively $\emptyset \neq (\alpha P_i - X_i) \subseteq \alpha P_j$). Thus $\langle i, j \rangle$ is a request when P_i is trying to communicate with P_j, and a strong request if P_i can *only* communicate with P_j.

We say this request or strong request is *ungranted* if additionally

$$\alpha P_i \cap \alpha P_j \ \subseteq X_i \cup X_j \ ,$$

i.e., if P_j is unwilling to respond to P_i's request. One can regard ungranted requests as being the building blocks of deadlock.

Sometimes we are only interested in ungranted requests when neither process is able to communicate outside some set Λ. (Often this will be the vocabulary of the network.)

Thus we define $\langle i, j\rangle$ to be an ungranted request or strong request *with respect to* Λ if, in addition to the above,

$$(\alpha P_i - X_i) \cup (\alpha P_j - X_j) \subseteq \Lambda\ .$$

We will write

$$P_i \xrightarrow{\sigma} P_j \quad \text{or} \quad P_i \xRightarrow{\sigma} P_j$$

when $\langle i, j\rangle$ is a request or strong request of σ. Similarly we will write

$$P_i \xrightarrow{\sigma}\bullet\, P_j \quad \text{or} \quad P_i \xRightarrow{\sigma}\bullet\, P_j$$

when $\langle i, j\rangle$ is a ungranted request or strong request of σ, and

$$P_i \xrightarrow{\sigma,\Lambda}\bullet\, P_j \quad \text{or} \quad P_i \xRightarrow{\sigma,\Lambda}\bullet\, P_j$$

when $\langle i, j\rangle$ is an ungranted request with respect to Λ. $\Box$

Notice that if $\Lambda \subseteq \Lambda'$, then $P_i \xrightarrow{\sigma,\Lambda}\bullet\, P_j \;\Rightarrow\; P_j \xrightarrow{\sigma,\Lambda'}\bullet\, P_j$, and that $P_i \xrightarrow{\sigma,\Lambda}\bullet\, P_j \;\Leftrightarrow\; P_i \xrightarrow{\sigma}\bullet\, P_j$ when $\alpha P_i \cup \alpha P_j \subseteq \Lambda$. (Of course, similar observations are true of strong requests.)

Definition. Let $\{P_i \mid 1 \leq i \leq N\}$ be a triple disjoint network with vocabulary Λ. P_i is said to be *blocked* in a state σ if

1. $P_i \xrightarrow{\sigma} P_j$ for some j, and
2. $P_i \xrightarrow{\sigma,\Lambda}\bullet\, P_j$ whenever $P_i \xrightarrow{\sigma} P_j$.

Lemma 1. If σ is a state of the triple disjoint, busy network V, then σ is a deadlock state if, and only if, every process in V is blocked.

Proof. This follows easily from the definitions. $\Box$

Definition. In the network $\{P_i \mid 1 \leq i \leq N\}$ with vocabulary Λ, the sequence of distinct indices $\langle i_0, \ldots, i_{r-1}\rangle$ with $r \geq 3$ is termed a *cycle of ungranted requests* if, for each j, $\langle i_j, i_{j+1}\rangle$ is an ungranted request with respect to Λ (where addition is modulo r). $\Box$

We have stipulated that all cycles of ungranted requests must have length at least three. The length two case, where each of a pair of processes asks the other for a communication, but where they cannot agree on anything, is sufficiently different to deserve special treatment: see section 4.

It will turn out that these cycles are symptomatic of deadlock in large classes of networks. The reason for stipulating that all the ungranted requests are with respect to Λ is that, if one of the P_i can communicate outside Λ, the network cannot be deadlocked.

A *cycle* in a communication graph will be a sequence of at least three distinct processes, each of which is joined to the next by an edge and where the first is joined to the last.

Clearly each cycle of ungranted requests corresponds to a cycle in the communication graph.

It has long been known that trees (networks with no cycles) are especially easy from the point of view of deadlock analysis. For example a number of simple conditions were introduced in [BR2,3] which ensure the deadlock freedom of trees. Perhaps the most useful of these conditions is described in section 4 of the present paper. It is the purpose of this paper to present techniques for proving deadlock freedom even in networks with many cycles.

3 A proof-rule for deadlock freedom.

We are now in a position to present the simplest form of our main result, which establishes the validity of a technique for proving deadlock freedom. The idea behind our method is simple and derives from methods used to prove the termination of loops in standard programming languages. We assign to each process in a network a function which assigns values to the states of that process. If it can be proved that whenever one process is waiting for another, the value of its state must be at greater than that of the one it is waiting for, the network must be deadlock free. This is because in a deadlock state one would, starting from any process, be able to construct an infinite sequence of processes whose states have strictly decreasing values. This means that all the processes in the sequence are different, which is clearly impossible in a finite network. We can think of these functions as being the *variant* of the network.

Theorem 1. Let $V = \{P_i \mid i \in \{1, \ldots, N\}\}$ be a triple-disjoint, busy network with vocabulary Λ. Suppose that there exist functions

$$f_i : \tilde{\mathcal{F}}[\![P_i]\!] \longrightarrow \Pi \qquad (i \in \{1, \ldots, N\})$$

where $(\Pi, >)$ is a partial order, such that if $\sigma = \langle s, \underline{X} \rangle$ is any state of V then

$$P_i \xrightarrow{\sigma,\Lambda}\bullet\, P_j \;\Rightarrow\; f_i(s \restriction \alpha P_i, X_i) > f_j(s \restriction \alpha P_j, X_j) \ .$$

Then V is deadlock free.

Proof. Suppose that V is as in the statement of the theorem and $\delta = \langle s, \underline{X} \rangle$ is a deadlock state.

For each i we may, using Lemma 1, select an index $r(i)$ such that $P_i \xrightarrow{\delta,\Lambda}\bullet\, P_{r(i)}$. Necessarily, then, $f_i(s \restriction \alpha P_i, X_i) > f_{r(i)}(s \restriction \alpha P_{r(i)}, X_{r(i)})$.

Now observe that each of the indices $i, r(i), r^2(i), \ldots, r^m(i), \ldots$ is distinct because, by a trivial induction, if $m < n$, then

$$f_{r^m(i)}(s \restriction \alpha P_{r^m(i)}, X_{r^m(i)}) > f_{r^n(i)}(s \restriction \alpha P_{r^n(i)}, X_{r^n(i)}) \ .$$

This contradicts the fact that our network is finite. □

Unfortunately the preconditions of Theorem 1 are not quite the type of completely local conditions we are seeking for deadlock freedom. The problem is that we are required to check every state of the whole network, rather than the states of small subnetworks. However, it is clear that the condition is essentially one on pairs of processes and it is easy to derive purely local properties which imply it.

Lemma 2. Suppose $V = \{P_i \mid i \in \{1, \ldots, N\}\}$ is a triple-disjoint, busy network with vocabulary Λ, and that $(\Pi, >)$ is a partial order. Then if the functions

$$f_i : \tilde{\mathcal{F}}[\![P_i]\!] \longrightarrow \Pi \qquad (i \in \{1, \ldots, N\})$$

have the property that, whenever $\sigma = \langle s, \langle X_i, X_j \rangle \rangle$ is a state of any two-element subnetwork $\{P_i, P_j\}$ $(i \neq j)$,

$$P_i \xrightarrow{\sigma,\Lambda}\bullet\, P_j \quad \Rightarrow \quad f_i(s \upharpoonright \alpha P_i, X_i) > f_j(s \upharpoonright \alpha P_j, X_j) \; .$$

Then V satisfies the conditions of Theorem 1 (with functions f_i). Furthermore this is also true of every subnetwork of V.

Proof. The main part of this result follows trivially because any ungranted request in V gives rise to an ungranted request with respect to Λ in $\{P_i, P_j\}$. The conditions of this lemma are hereditary for, since the vocabulary Λ' of every subnetwork of V is a subset of Λ, any ungranted request with respect to Λ' is also one with respect to Λ. □

Rule 1. If V is any network satisfying the conditions of Lemma 2, then V is hereditarily deadlock free. □

Rule 1 provides us with a technique for proving deadlock freedom where the checking of preconditions is entirely local: we have to verify that each P_i is deadlock free, that no communication is in the alphabet of three processes, and that each pair of processes which can communicate satisfy the "decreasing variant" condition. The only time when one needs to consider the whole network is while inventing a suitable system of variant functions.

In general, variant functions are associated with the maximal failures of processes, however we will often restrict them to just traces, as these are simpler to work with and almost always sufficient: for deterministic CSP processes, there is no difference.

The guiding principle of our work has been to develop methods which are reasonably easy to apply in practice. (Hence our demand for purely local preconditions.) Therefore an important part of this paper will be a series of worked examples to illustrate their application. The following three examples illustrate the applicability of the Rule 1, which is our most basic technique. The first two are applications to fairly specific systems. The third illustrates how an existing theorem on deadlock freedom can be extended using our techniques, and in turn how this extended result can be used to establish straightforward ways of constructing deadlock free networks.

Example 1: Self-timed version of a systolic array. A common type of systolic array is one where there is a rectangular array of processing elements, each of which repeatedly inputs a pair of elements from its "up" and "left" channels, and then outputs values to its "bottom" and "right" channels and processes its present values. This is sometimes implemented with a distributed clock signal, which ensures that the processing elements proceed in exact step. Usually it will be necessary to use some device such as triangles of delay elements to ensure that the correct pairs of data elements meet.

However, if the processing elements are described as CSP processes (with "handshake" communication) there is no need either for a distributed clock or for the delay elements; the network becomes self-timed. (See [H] for an example of a matrix multiplication algorithm treated in this way.) Which implementation is better will depend on the application: one needs to weigh the overheads of handshake communication against the overheads of distributing a clock, introducing a delay element, and needing to slow the clock to the longest time any element might ever take to complete its computation.

By moving to the self-timed version of such an algorithm one loses the comforting predictability of the tightly synchronised version: there is no longer any way in which we can predict what state the network will be in at any time. We need to analyse the patterns of communicating behaviour that can arise. Fortunately these are essentially independent of the particular function being computed by the array, and so it is sufficient to examine a paradigm array where all details of the data being processed are omitted, so that only the synchronisations between processes remain. The paradigm processing element is described by the CSP process

$$\begin{aligned} P(up, down, left, right) = {} & (up \to SKIP \parallel left \to SKIP); \\ & (down \to SKIP \parallel right \to SKIP); \\ & P(up, down, left, right) \ . \end{aligned}$$

The array is then formed by setting

$$P_{i,j} = P(v_{i,j}, v_{i+1,j}, h_{i,j}, h_{i,j+1}) \quad \text{and} \quad \alpha P_{i,j} = \{v_{i,j}, v_{i+1,j}, h_{i,j}, h_{i,j+1}\}$$

and forming the triple disjoint, busy network $\{P_{i,j} \mid 1 \le i \le m,\ 1 \le j \le n\}$. Even though the individual elements of this network are quite simple, because they are free to choose their own rates of progress the number of possible behaviour patterns of the whole is very large indeed. It is by no means immediately obvious that none of these can lead to deadlock, but we can prove deadlock freedom using our technique.

Let us consider the operation of the subnetwork $\{P_{i,j}, P_{i,j+1}\}$. Observe that $\alpha P_{i,j} \cap \alpha P_{i,j+1} = \{h_{i,j+1}\}$, and that each process communicates this event exactly once on each cycle: $P_{i,j}$ as either the third or last event, $P_{i,j+1}$ as either the first or second. Therefore the two processes can never be more than one cycle apart. It is easy to see that if $\langle s, \underline{X} \rangle$ is a state of $\{P_{i,j}, P_{i,j+1}\}$ and $P_{i,j} \xrightarrow{\langle s, \underline{X} \rangle}\bullet P_{i,j+1}$ then $P_{i,j}$ must have completed one more cycle than $P_{i,j+1}$. More precisely, we have

$$|s \upharpoonright \alpha P_{i,j}| \ge 4 \times |s \upharpoonright \{h_{i,j+1}\}| + 2 \quad \text{and} \quad |s \upharpoonright \alpha P_{i,j+1}| \ \le\ 4 \times |s \upharpoonright \{h_{i,j+1}\}| - 1$$

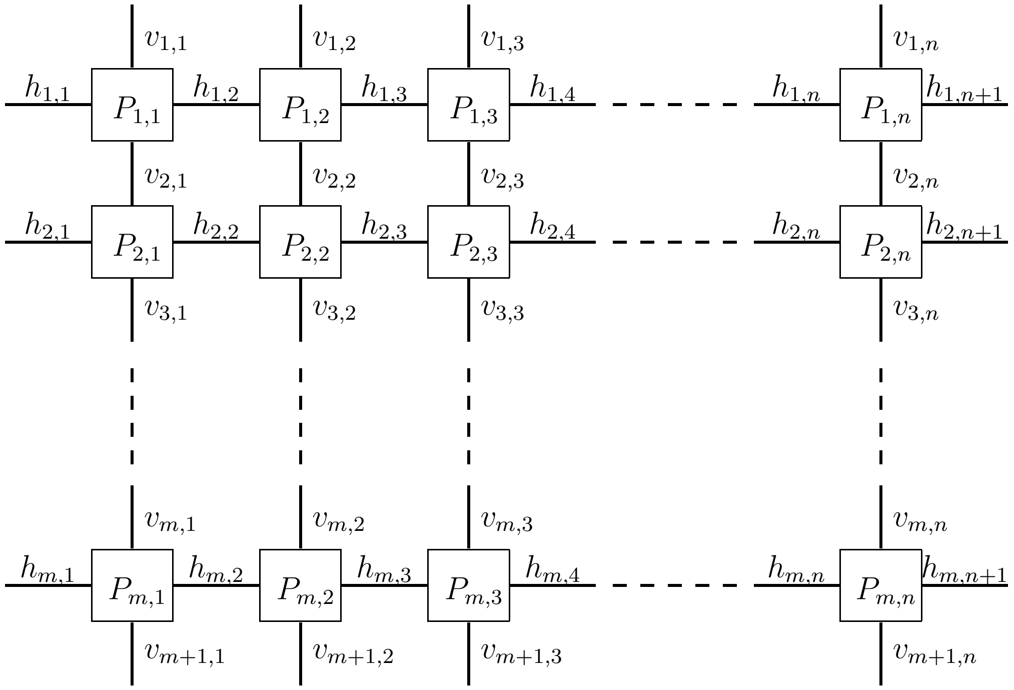

Figure 1: Matrix multiplier array

as $P_{i,j}$ must have executed at least two events of its cycle number $|s\upharpoonright\{h_{i,j+1}\}|+1$ before it can wait for $h_{i,j+1}$ and $P_{i,j+1}$ cannot have completed cycle number $|s\upharpoonright\{h_{i,j+1}\}|$. Thus, combining the above inequalities,

$$3 \;\leq\; |s\upharpoonright\alpha P_{i,j}| \;-\; |s\upharpoonright\alpha P_{i,j+1}|,$$

and hence

$$|s\upharpoonright\alpha P_{i,j}| \;>\; |s\upharpoonright\alpha P_{i,j+1}|+2$$

which in turn implies

$$|s\upharpoonright\alpha P_{i,j}|+2\times(i+j) \;>\; |s\upharpoonright\alpha P_{i,j+1}|+2\times(i+j+1)\;.$$

For similar reasons, if $P_{i,j+1}\xrightarrow{\langle s,\underline{X}\rangle}\bullet P_{i,j}$ then

$$|s\upharpoonright\alpha P_{i,j+1}| \;\geq\; 4\times|s\upharpoonright\{h_{i,j+1}\}| \quad\text{and}\quad |s\upharpoonright\alpha P_{i,j}| \;\leq\; 4\times|s\upharpoonright\{h_{i,j+1}\}|+1$$

and hence

$$1 \;\geq\; |s\upharpoonright\alpha P_{i,j}| \;-\; |s\upharpoonright\alpha P_{i,j+1}|.$$

This is easily seen to imply

$$|s\upharpoonright\alpha P_{i,j+1}|+2\times(i+j+1) \;>\; |s\upharpoonright\alpha P_{i,j}|+2\times(i+j)\;.$$

Clearly the cases dealing with ungranted requests between $P_{i,j}$ and $P_{i+1,j}$ are symmetric, the common event in their alphabet being $v_{i+1,j}$. Therefore in this case if $\langle s, \underline{X} \rangle$ is a state of the subnetwork $\{P_{i,j}, P_{i+1,j}\}$

$$P_{i,j} \xrightarrow{\langle s,\underline{X}\rangle}\!\bullet P_{i+1,j} \;\Rightarrow\; |s \restriction \alpha P_{i,j}| + 2 \times (i+j) \;>\; |s \restriction \alpha P_{i+1,j}| + 2 \times (i+1+j)$$

$$P_{i+1,j} \xrightarrow{\langle s,\underline{X}\rangle}\!\bullet P_{i,j} \;\Rightarrow\; |s \restriction \alpha P_{i+1,j}| + 2 \times (i+1+j) \;>\; |s \restriction \alpha P_{i,j}| + 2 \times (i+j)\ .$$

Thus defining $f_{i,j} : \mathit{traces}(P_{i,j}) \to \mathbf{N}$ where

$$f_{i,j}(s) \;=\; |s| + 2 \times (i+j)$$

we have that, in general, if $P_{i,j} \xrightarrow{\langle s,\underline{X}\rangle}\!\bullet P_{k,l}$ then $f_{i,j}(s \restriction \alpha P_{i,j}) > f_{k,l}(s \restriction \alpha P_{k,l})$. We conclude that the network is deadlock free.

The parallel input, parallel output scheme for relaying information that we used in this example is probably the most efficient in terms of avoiding delays to processing. We will see in Example 3 that it is in itself a powerful tool for ensuring that networks broadly similar to the present one are deadlock free.

It is entertaining to consider variants on the present system where the parallel input/output scheme is replaced by another. In fact *no* rectangular array where, on every cycle, each process always does both its inputs before both its outputs can deadlock: this is shown by considering a network made up of elements with the following form.

$$\begin{aligned} P'(\mathit{up}, \mathit{down}, \mathit{left}, \mathit{right}) = {} & ((\mathit{up} \to \mathit{left} \to \mathit{SKIP}) \sqcap (\mathit{left} \to \mathit{up} \to \mathit{SKIP})); \\ & ((\mathit{down} \to \mathit{right} \to \mathit{SKIP}) \sqcap (\mathit{right} \to \mathit{down} \to \mathit{SKIP})); \\ & P'(\mathit{up}, \mathit{down}, \mathit{left}, \mathit{right})\ \ . \end{aligned}$$

(P' may, on each cycle, choose which input channel it wants to communicate on first, and then which output channel.) Since these are the most nondeterministic processes satisfying the above specification, if a network composed of P's is deadlock free, so must every such network.

The construction of the variant for the network of P's is left as an exercise to the reader. He will find that the variant given above will only guarantee non-strict inequality between all linked pairs of nodes. In fact, because this network is in some sense only marginally deadlock free, and because the nodes are non-deterministic in a crucial way, it is necessary to define the variant functions for P' network in terms of the maximal failures of the nodes, not just their traces.

The fact that this nondeterministic version is finely balanced on the edge of deadlock is emphasised when one considers the corresponding three-dimensional network. This is a three dimensional rectangular array of processors where, on each cycle, each inputs on channels *up, left* and *front* (in nondeterministic order) and then outputs on *down, right* and *back*. When we know that the two-dimensional network is deadlock free it is surprising to discover that network can easily deadlock. (The reader might enjoy constructing this deadlock in a $2 \times 2 \times 2$ array.) $\square$

In the above example there is clearly some sense in which we can regard $f_{i,j}(s \restriction \alpha P_{i,j})$ as measuring the "amount of work" $P_{i,j}$ has done: thus one process can only be waiting for another when it has progressed further in its calculations. Note, for example, that the $f_{i,j}$ are monotonic in the length of trace. This is a useful analogy for this network and can assist in the construction of the variant functions for similar networks. However, as is illustrated by the next example, our technique can be applied in quite different circumstances.

Example 2: The correctness of a resource allocation protocol. Suppose that there are a number of "user" processes $\{P_i \mid 1 \leq i \leq N\}$ which compete for the resources $\{R_j | 1 \leq j \leq M\}$. (No resource may be used by more than one P_i at a time.) A well-known method of avoiding deadlock in this system is to place a linear order on the resources, which we can assume is that on the indices $\{1, \ldots, M\}$, and ensure that no process ever tries to acquire a resource with higher priority than a resource it already holds. Here will show that this result can be proved simply using Rule 1.

We first need to define the system a little more precisely. We can define the resource processes

$$R_j \;=\; j.get.i : \{1, \ldots, N\} \rightarrow j.rel.i \rightarrow R_j$$

($\alpha R_j = \{j.get.i, j.rel.i | 1 \leq i \leq N\}$). We will assume that all the P_i are deadlock free, that $\alpha P_i \cap \alpha P_j = \emptyset$ whenever $i \neq j$ and that $\alpha P_i \cap \alpha R_j = \{j.get.i, j.rel.i\}$. Further, whenever $s \in traces(P_i)$ we will assume

$$s \restriction \alpha R_j \leq \langle j.get.i, j.rel.i\rangle^n \qquad \text{for some } n$$

(which means that P_i never tries to release R_j when it doesn't hold it, or to acquire R_j when it does hold it). Thus, when $s \in traces(P_i)$, we can define the set of resources P_i "has" by

$$r_i(s) = \{j \mid |s \restriction \{j.get.i\}| > |s \restriction \{j.rel.i\}|\} \; .$$

We can think of the above conditions as setting up a "reasonable" system of resource-using processes. In this framework we can restate the protocol as follows:

$$s\langle j.get.i\rangle \in traces(P_i) \wedge k < j \;\Rightarrow\; k \notin r_i(s) \; .$$

It is clear that, under our assumptions, the network $V = \{P_i \mid 1 \leq i \leq N\} \cup \{R_j \mid 1 \leq j \leq M\}$ is busy and triple disjoint. As our partial order Π we choose $\{1, 2, \ldots, 2M, 2M + 1, big\}$, where the natural numbers have their usual order and big is maximal in Π. We define functions $f_i : traces(P_i) \rightarrow \Pi$ and $g_j : traces(R_j) \rightarrow \Pi$ as follows:

$$f_i(s) = 2 \times min(\{M + 1\} \cup r_i(s)) - 1$$

$$\begin{aligned} g_j(s) &= big && \text{if } |s| \text{ is even} \\ &= 2 \times j && \text{if } |s| \text{ is odd.} \end{aligned}$$

(Note that, when $s \in traces(R_j)$, $|s|$ is even when the resource is "free" and odd when it is held by some P_i.) It is a trivial matter to verify that these functions satisfy the required condition for Rule 1; we may therefore conclude that the network is deadlock free.

Example 3: Networks of cyclic elements. In [D], Dijkstra and Scholten state and prove a theorem concerning the deadlock freedom of networks in which every element communicates with its neighbours in a strict cyclic order. We show here that an extension of this result can be proved via Rule 1. The following is the theorem of [D] re-phrased to reflect the terminology of the present paper.

Theorem [D]. Suppose each of the processes in the triple-disjoint, conflict free[1] network $\{P_i \mid 1 \le i \le n\}$ communicates with all its neighbours in strict cyclic order: P_i communicates with the processes with index $\langle c_{i,1}, \ldots, c_{i,m_i}\rangle$ (where $c_{i,1}, \ldots, c_{i,m_i}$ are all distinct) unendingly in cyclic order starting with $c_{i,1}$. Then the network is hereditarily deadlock free if and only if there is no cycle in the communication graph

$$\langle a_0, \ldots, a_{m-1}\rangle \quad (a_i \text{ all distinct, } m > 2)$$

where, for each j, process a_j wants to talk to process a_{j+1} before it is prepared to talk to a_{j-1} (arithmetic being modulo m). (In other words, if s_j and p_j are defined to be such that $c_{j,s_j} = a_{j+1}$ and $c_{j,p_j} = a_{j-1}$ then $p_j > s_j$ for all $j \in \{0, 1, \ldots, m-1\}$.) □

We strengthen this result by replacing the cyclic communication with single processes by cyclic parallel communications with groups of processes in the style of Example 1. Formally we say that a process communicates in parallel with the nonempty subset C of its neighbours when, before it can proceed, it must communicate with each element of C exactly once and furthermore, until this is finished, always requests a communication with all elements of C with which it has not yet communicated. Typically this would be implemented by the CSP construct

$$\mathop{\Big\|}_{P \in C} a_P \to SKIP \quad ,$$

Where $a_P \in \alpha P$ for each $P \in C$.

Theorem. Suppose each of the processes in the triple-disjoint, conflict free network $\{P_i \mid 1 \le i \le n\}$ communicates with each of its neighbours once on every cycle by communicating in parallel with subsets of them in cyclic order. Thus there is some partition $\{C_{i,1}, \ldots, C_{i,m_i}\}$ of the indices of each P_i's neighbours such that, in strict and unending cyclic order, P_i communicates with the processes with indices in each $C_{i,j}$ in parallel. Then the network is hereditarily deadlock free if and only if there is no cycle in the communication graph

$$\langle a_0, \ldots, a_{m-1}\rangle \quad (a_i \text{ all distinct, } m > 2)$$

[1] *Conflict free* networks are defined formally at the begining of the next section. Essentially they are networks where no pair of processes can request a communication of each other without being able to agree on one.

where, for each j, process a_j wants to talk to process a_{j-1} before it is prepared to talk to a_{j+1} (arithmetic being modulo m). (In other words, if s_j and p_j are defined to be such that $a_{j+1} \in C_{j,s_j}$ and $a_{j-1} \in C_{j,p_j}$ then $p_j < s_j$ for all $j \in \{0, 1, \ldots, m-1\}$.) $\square$

Proof. The "only if" part of the proof is essentially the same as that in [D]: one simply observes that any cycle with the property above is a subnetwork that can (and does) deadlock. None of the members a_j of the cycle can agree to communicate with its successor a_{j+1} until it has communicated with its predecessor a_{j-1}: therefore no element of the cycle can be the first to communicate with its successor. This means that after a few communications the subnetwork is bound to find itself in the state where each process is waiting (exclusively) for its predecessor. Indeed (as is observed in [D]), if the network is connected a cycle of this type must eventually deadlock the whole system. The number of cycles completed by a pair of neighbouring processes differs by at most one, as they share one event on each cycle. Hence in a connected network with a cycle of the type described, no process can complete more cycles than the diameter of the graph.

To prove the "if" part we will define a relation on the (undirected) edges of the communication graph. We will denote the edge between P_i and P_j by the two element set $\{i, j\}$. For all i and $k, j \leq m_i$ we define

$$\{i, a\} \prec \{i, b\} \quad \text{whenever } a \in C_{i,j}, b \in C_{i,k} \text{ and } j < k.$$

Thus the edges joining each process to its neighbours are "ordered" according to the order in which P_i first addresses them, with no order between edges along which P_i communicates in parallel. Note that, because no two processes have more than one edge in common, $\prec$ induces a partial order on the edges surrounding each individual P_i.

Now let $\lhd$ be the transitive closure of $\prec$. Claim that $\lhd$ is a partial order (on the set of all edges of the graph). This could only fail if there were a sequence (necessarily with length > 2) of distinct edges

$$\{i_0, j_0\} \prec \{i_1, j_1\} \prec \cdots \prec \{i_m, j_m\}$$

where $i_0 = i_m$ and $j_0 = j_m$.

Since each pair of edges ordered by $\prec$ have a node in common, and since $\prec$ is transitive on edges all of which have a particular node in common, we may assume that

$$0 \leq r < m \;\Rightarrow\; j_r = i_{r+1} \, .$$

(Any edge which has the same node in common with each of its neighbours may be removed from the sequence.)

Consider the cycle of processes given by $\langle i_0, i_1, \ldots, i_{m-1} \rangle$. For each k we have $\{i_k, i_{k-1}\} \prec \{i_k, i_{k+1}\}$ (arithmetic modulo m). Thus, if we define s_k and p_k as in the statement of the theorem, we have $s_k > p_k$ by definition of $\prec$. This contradicts the assumption that no such cycle exists. Hence $\lhd$ is a partial order.

Since every partial order can be extended to a linear order it follows that there is a function g from the edges of the graph to the open interval $(0, 1)$ such that

$$\{i, j\} \lhd \{k, l\} \;\Rightarrow\; g\{i, j\} < g\{k, l\} \;.$$

Now define functions $f_i : \mathit{traces}(P_i) \to \mathbf{R}$ by

$$f_i(s) \;=\; \left[\frac{|s|}{N_i}\right] + \mathit{max}\{g\{i, j\} \mid \exists a \in \alpha P_j . s\langle a\rangle \in \mathit{traces}(P_i)\}$$

where N_i is the number of neighbours of P_i and $[x]$ denotes the greatest integer less than or equal to x.

In other words, $f_i(s)$ is the number of complete cycles that P_i has executed plus the greatest value of any edge over which it can communicate next. Note that if P_i has completed n cycles then $n < f_i(s) < n + 1$.

Consider the interaction of any pair of neighbouring processes $\{P_i, P_j\}$. By our assumptions they share exactly one communication on every cycle. Hence if $P_i \xrightarrow{\sigma}\bullet P_j$ it is clear that either P_i and P_j have completed the same number of cycles but that P_j has not yet reached the point in its present cycle where it can communicate with P_i, or P_j has completed one less cycle than P_i. In the latter case it is clear that, if P_i is on cycle n, then

$$f_i(s \upharpoonright \alpha P_i) > n > f_j(s \upharpoonright \alpha P_j)$$

which establishes the desired inequality.

So suppose that both processes have completed n cycles. Since P_i can communicate with P_j on its next step we have from the definition of f_i that $f_i(s \upharpoonright \alpha P_i) \geq n + g\{i, j\}$. On the other hand we know that P_j has not yet reached the point where it can communicate with the set of its neighbours that includes i, so that every event which it can communicate next belongs to some αP_k with $\{j, k\} \prec \{j, i\}$. The definition of f_j then tells us that $f_j(s \upharpoonright \alpha P_j) < n + g\{i, j\}$. Thus the inequality of Rule 1 is satisfied in this case as well. We may thus conclude that the network is deadlock free. □

This result has a number of useful corollaries, three of which are listed below. They all illustrate the power of parallel communication as a means of avoiding deadlock.

Corollary 1. If a network of the type described in the statement of the above theorem is modified by replacing some of the P_i by processes which are identical except that all the communications in some consecutive groups of communications now occur in parallel, deadlock cannot be introduced.

Proof. Modifying a network in this way can only serve to make the relation $\prec$, and hence the partial order $\lhd$, smaller so that no cycle of the type described in the statement of the theorem can be introduced. □

Corollary 2. If, in a network of the type described in the statement of the theorem, each process does all its communication on each cycle in parallel, then that network is deadlock free.

Proof. In such a network the relations $\prec$ and $\lhd$ are empty. $\Box$

Corollary 3. Suppose V is a network of the type described in the theorem on which there is a partial order on the elements of the network such that every pair of neighbours is comparable. (In a network laid out geometrically this might be the partial order induced by the x-coordinate of the processes' positions; the comparability condition meaning that there is no direct link between two processes with the same x-coordinate.) Suppose further that each process is designed so that on each cycle it communicates with all its neighbours less than itself in parallel. (The order of its communications with those greater than itself is not specified, and neither is the relative order of the parallel block within these communications.) Then the network is deadlock free.

Proof. Any cycle of the type described in the statement of the theorem would necessarily have one or more maximal elements. So suppose P_i is a maximal element, and its predecessor and successor in the cycle are P_j and P_k. On the one hand, since by assumption both P_j and P_k are less than P_i, the latter communicates (on each cycle) with P_j and P_k in parallel. On the other hand (by the properties of the cycle) P_i needs to communicate with P_j before it can first communicate with P_k. This gives a contradiction, proving the Corollary. $\Box$

One case of this corollary is where each process cycles between communicating with all its left-hand neighbours in parallel, and all its right-hand neighbours in parallel. Even though the right-hand parallel communication is not strictly necessary to achieve deadlock freedom, we can expect it to improve efficiency. Note that the network of Example 1 falls exactly into this category. (The partial order is that induced by $i + j$.) Often, as in Example 1, the communications to the left will be inputs and those to the right outputs. In this case the outputs at the extreme right of the network will correlate exactly with the inputs at the extreme left: the first output on each channel is produced by the first group of inputs, and so on.

It is at first sight surprising that some networks proved deadlock free by Corollary 3 are so (even networks of the very restricted type described in the last paragraph). For example the one in Figure 2, where there is no clear division of the processes into "levels", meaning that some parts of the network apparently want to go faster than others.

Later, when the necessary machinery is available, we will be able to generalise Corollary 3 a little.

It should be possible to discover further extensions to the Theorem above, and also others in a similar style. Hopefully these will lead to more results like the three corollaries above, which have the great virtue of being simple to apply in practice. $\Box$ **(End of Example 3)**

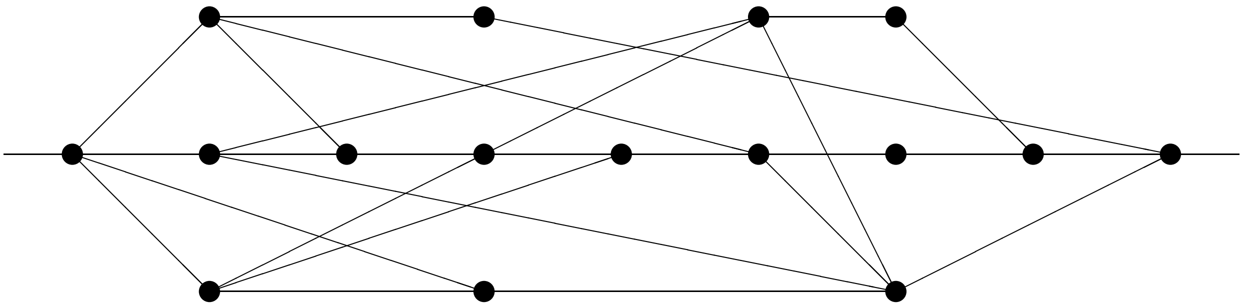

Figure 2: A typical left-then-right system which Corollary 3 proves deadlock free

Rule 1 is strong enough to prove the deadlock freedom of many systems in addition to those shown above. It can be used to prove the deadlock freedom of the "token passing ring" system of [BR2]. (Here the fact that ungranted requests need only be in the vocabulary of the network is crucial.) Also it can be applied to various versions of the "dining philosophers", including the one treated using Rule 2 in the next section.

However there are many networks to which it cannot be applied. There are networks, like the dining philosophers (all of whom are, say, right-handed) with a "butler" process who regulates the number of philosophers who may sit down, which though deadlock free are not *hereditarily* deadlock free. (After the removal of the butler, this system may deadlock.) Secondly there are networks where a pair of processes can each simultaneously be willing to communicate with each other without there being any joint communication possible. (Under the conditions of Rule 1, each process would need a variant smaller than the other.) The following theorem identifies precisely those networks to which Rule 1 can be applied, its proof is in the same vein as that of the main theorem in Example 3.

Theorem 2. Suppose $V = \{P_i \mid i \in \{1, \ldots, N\}\}$ is a triple-disjoint, busy network with vocabulary Λ. Let $\Pi = \bigcup_{i=1}^{N}(\tilde{\mathcal{F}}[\![P_i]\!] \times \{i\})$. Define the relation $\succ$ on Π as follows: $(s, X, i) \succ (t, Y, j)$ holds whenever there exists a state $\sigma = \langle u, \langle X, Y\rangle\rangle$ of $\{P_i, P_j\}$ such that $u{\restriction}\alpha P_i = s$, $u{\restriction}\alpha P_j = t$ and $P_i \xrightarrow{\sigma,\Lambda}\bullet\, P_j$. Then V can be proved deadlock free by Rule 1 if, and only if, the transitive closure of $\succ$ on Π is a partial order. $\Box$

Proof. Let $\rhd$ be the transitive closure of the relation $\succ$ on Π. Firstly, let us assume $(\Pi, \rhd)$ is not a partial order. This can only be if there is a sequence

$$(s_0, X_0, i_0) \succ (s_1, X_1, i_1) \succ \cdots \succ (s_m, X_m, i_m)$$

where $s_0 = s_m$, $X_0 = X_m$ and $i_0 = i_m$. Now, if Rule 1 can be used on V, by the properties of the variant functions and the definition of $\succ$, we would have

$$f_{i_0}(s_0, X_0) > f_{i_1}(s_1, X_1) > \cdots > f_{i_m}(s_m, X_m)$$

and hence that $f_{i_0}(s_0, X_0) > f_{i_0}(s_0, X_0)$, which is a contradiction. Thus, if $(\Pi, \triangleright)$ is not a partial order, Rule 1 cannot be applied to V.

Now, let us assume that $(\Pi, \triangleright)$ is a partial order. Then define functions

$$f_i : \tilde{\mathcal{F}}[\![P_i]\!] \to \tilde{\mathcal{F}}[\![P_i]\!] \times \{i\} \quad \text{by } f_i(s, X) = (s, X, i).$$

It is easy to verify that, using $(\Pi, \triangleright)$ as the partial order required by Rule 1, the functions f_i are in fact variant functions. Hence, if $(\Pi, \triangleright)$ is a partial order, it is possible to prove V deadlock free by Rule 1.

So we conclude that V may be proved deadlock free by Rule 1 exactly when $(\Pi, \triangleright)$ is a partial order. $\Box$

Note that when Rule 1 fails, from the theorem above, there exists a "local cycle of ungranted requests". However, this does not necessarily imply the existence of a global state in which there is a cycle of ungranted requests (although the converse is true). An example of such a network is the message passing ring in [BR3] where each element is a restricted double buffer. This network is deadlock free, has no global state in which there is a cycle of ungranted requests but does have local cycles that preclude the use of Rule 1.

Having established the type of networks which Rule 1 encompasses, the rest of this paper is devoted to extending our methods to wider classes of network, and to developing an understanding of which networks are liable to be susceptible to a given technique.

4 Networks with conflict

The notions of *conflicts* and *strong conflicts* between pairs of processes were introduced in [BR3], and used to prove results about deadlock. In this section we summarise the main ideas from the earlier paper and show how, by incorporating these ideas into our present theory, we can make our proof rule both easier to apply and more widely applicable.

Definition. Suppose Λ is a set of communications. The processes P, Q are said to be *in conflict with respect to* Λ in the state $\sigma = \langle s, \langle X, Y \rangle \rangle$ of $\{P, Q\}$ if and only if

$$P \xrightarrow{\sigma,\Lambda}\bullet\, Q \quad \text{and} \quad Q \xrightarrow{\sigma,\Lambda}\bullet\, P\ .$$

They are in *strong conflict* if, additionally,

$$P \overset{\sigma,\Lambda}{\Longrightarrow}\bullet\, Q \quad \text{or} \quad Q \overset{\sigma,\Lambda}{\Longrightarrow}\bullet\, P\ .$$

P and Q are said to be *conflict free* (respectively *strong conflict free*) with respect to Λ if they have no conflicts (resp. strong conflicts) with respect to Λ.

We will say that a network V is conflict free (resp. strong conflict free) if each of its pairs of processes is conflict free (resp. strong conflict free) with respect to Λ, the vocabulary of V. □

A conflict is precisely the "cycle of ungranted requests of length two" described at the end of Section 2. A strong conflict is one where one of the pair of processes involved is completely blocked by the other. Once again we restrict our attention to the case where neither process can communicate outside the vocabulary of the network because no network is deadlocked when any of its components is able to communicate independently. It is important to note that conflict freedom and strong conflict freedom are hereditary properties of a network and can be checked by purely local analysis.

The communication patterns of most practical parallel systems are fundamentally conflict free in that, even though a given version of a program is not, it can be trivially be redesigned to be so (see [BR3] for an example). For example, any pair of processes connected by a single occam-like channel are conflict free because the inputting process cannot be willing to talk to the other without being willing to accept anything the other might offer. Some systems, often ones with particularly symmetric communication patterns, cannot be made conflict free. However it is hard to think of a sensible system that cannot be made free of strong conflict, since strong conflict arises when one process is willing to talk to another, even though it is itself preventing that process from proceeding. As we shall see, strong conflict freedom is the more important of the two conditions. See [BR3] for a more detailed discussion of these conditions, and a number of examples.

The following theorem gives us a sharp and usable characterisation of deadlock states.

Theorem 3 [BR3] Every deadlock state of a triple disjoint, busy, strong conflict free network has a cycle of ungranted requests. □

This theorem says that when deadlock occurs in a "reasonable" network, there is a chain of processes in which each process is waiting for the one ahead of it in the chain, and where the last process is waiting for the first one. The following useful result is an immediate corollary.

Theorem 4 [BR3] If V is a busy, triple disjoint, strong conflict free network whose communication graph is a tree, then V is deadlock free. □

In practice, many parallel networks are trees (for example pipeline systems and binary trees). Theorem 4 is usually all that one needs to prove their deadlock freedom.

If V is a network, we define the *disconnecting edges* of V to be the edges of the communication graph whose removal would increase the number of components of the graph. The *essential components* of V are the components of the graph after all disconnecting edges are removed. (In graph theoretic terms, the essential components are the maximal edge bi-connected subgraphs.) Note that the disconnecting edges are precisely the edges which cannot be part of any cycle in the graph. Observe also that the essential components of a tree are its individual processes, and that the essential components themselves always form a tree when an edge is drawn between a pair if and only if there can be communication between any of their elements. This fact, and analysis of conflict freedom, establishes the following result.

Theorem 5 [BR3] Suppose V is a triple disjoint network with essential components $V_1, \ldots, V_k$ where the pair of processes joined by each disconnecting edge are conflict free with respect to Λ, the vocabulary of V. Then if each of the V_i is deadlock free, so is V. $\Box$

This result identifies parts of networks which can, from the point of view of deadlock analysis, be regarded as independent. This is very useful since we can reasonably expect a small network to be much easier to analyse than a big one.

Theorem 3 can be combined with the idea of variants to give a number of results on deadlock which are sharper than either Theorem 1 or Theorem 3 by themselves. In the following result we relax the conditions on variants, so that the variant never increases on an ungranted request, rather than strictly decreases.

Theorem 6 Let $V = \{P_i \mid i \in \{1, \ldots, N\}\}$ be a triple disjoint, busy, strong conflict free network with vocabulary Λ. Furthermore, suppose there exist functions

$$f_i : \tilde{\mathcal{F}}[\![P_i]\!] \longrightarrow \Pi \qquad (i \in \{1, \ldots, N\})$$

where $(\Pi, >)$ is a partial order, such that whenever $\{i, j\}$ is a non-disconnecting edge and $\sigma = \langle s, \langle X_i, X_j \rangle \rangle$ is a state of $\{P_i, P_j\}$ then

$$P_i \overset{\sigma,\Lambda}{\longrightarrow}\bullet\, P_j \;\;\Rightarrow\;\; f_i(s \upharpoonright \alpha P_i, X_i) \geq f_j(s \upharpoonright \alpha P_j, X_j)\,.$$

Then any deadlock state $\langle s, \underline{X} \rangle$ of V contains a cycle of ungranted requests $\langle i_0, \ldots, i_{m-1} \rangle$ such that all the $f_{i_j}(s \upharpoonright \alpha P_{i_j})$ are equal.

Proof. Suppose V satisfies the prerequisites of the theorem and $\langle s, \underline{X} \rangle$ is a deadlock state. By Theorem 3, $\langle s, \underline{X} \rangle$ has a cycle of ungranted requests, say $\langle i_0, \ldots, i_{m-1} \rangle$. Now define $g : \{0, \ldots, m-1\} :\longrightarrow \Pi$ by setting $g(k) = f_{i_k}(s \upharpoonright \alpha P_{i_k}, X_{i_k})$. Since none of the edges making up the cycle can be a disconnecting edge, it follows from the properties of the functions f_i that

$$g(0) \geq g(1) \geq \cdots \geq g(m-1) \geq g(0)$$

and hence that $g(i) = g(0)$ for all i. $\Box$

Observe that when we define all the variant functions to be the same constant, Theorem 6 reduces to Theorem 3. Shortly this result will allow us to sharpen the technique introduced in Rule 1. By itself it provides a useful tool for analysing "difficult" networks for potential deadlocks by placing bounds on the places deadlock might appear. One tries to produce a system of variants which is as "refined" as possible (i.e., yields as many strict inequalities as possible). The search for deadlocks is then restricted to cycles with equal variant. The power of this idea is illustrated by returning to two of our earlier examples.

Example 1 revisited. Recall the nondeterministic verisions of the array in two and three dimensions. We observed there that the two dimensional case was deadlock free, but that its variant was hard to construct, but that the three dimensional case deadlocks. The reasons for this apparently paradoxical situation become far clearer when we examine the networks using Theorem 6. The variants we take are very simple: in the two dimensional case

$$f_{i,j}(s) \;=\; [|s|/2] + i + j$$

and in the three dimensional case

$$f_{i,j,k}(s) \;=\; [|s|/3] + i + j + k \;.$$

In each case the variant is the number of half cycles the process has completed plus its "depth" into the network. In each case it is simple to establish that the conditions of Theorem 6 are satisfied. It is also simple to establish that if $P_{i_1,j_1(,k_1)} \stackrel{\sigma}{\longrightarrow\bullet} P_{i_2,j_2(,k_2)}$ and $P_{i_2,j_2(,k_2)} \stackrel{\sigma}{\longrightarrow\bullet} P_{i_3,j_3(,k_3)}$ and all three processes have the same variant, then $i_1 + j_1(+k_1) = i_3 + j_3(+k_3)$. From this we can deduce that, in the two-dimensional case, any cycle of ungranted requests is a subset of $\{(i,j) \mid i+j \in \{r, r+1\}\}$ for some r, while in the three dimensional case it is a subset of some $\{(i,j,k) \mid i+j+k \in \{r, r+1\}\}$. The former is essentially a straight line, and contains no cycles (proving deadlock freedom), while the second is essentially a plane and contains many cycles (which is the reason why this network deadlocks). $\Box$

Example 3 revisited. We are now in a position to fulfil the promise made earlier and generalise Corollary 3. On careful analysis it turns out that the crucial features which make this result true are that, on each "cycle", every process communicates exactly once with each neighbour and that its communications with its left-hand neighbours are in parallel. The hypothesis (inherited from our extension of the Dijkstra theorem) that there is a pre-determined cyclic order to the communications turns out not to be necessary. Note that the processes in the networks on the Theorem below are free to be nondeterministic provided they satisfy the basic specification laid out.

Theorem Suppose $V = \{P_i \mid 1 \leq i \leq n\}$ is a triple disjoint, busy, conflict free network where each process communicates with each of its neighbours once on each "cycle". (In other words, if P_i has m neighbours, then for each k its communications number $k \times m + 1$ to $(k+1) \times m$ consist of one with each neighbour.) Suppose further that there is a partial order on the elements of the network such that every pair of neighbours is comparable, and that each process is designed so that on every cycle it communicates with all neighbours less than itself in parallel. Then the network is deadlock free.

Proof. The variant of P_i is the number of cycles it has completed. Since each pair of neighbours share exactly one event on each cycle, no process can wait for a process that has completed more cycles than it has. Thus these functions satisfy the conditions of Theorem 6. The form of these functions means that the cycles of ungranted requests described in the statement of Theorem 6 necessarily consist of processes on the same cycle. Suppose P_i is a maximal element of such a cycle, and its predecessor and successor in the cycle are P_j and P_k. Because P_i and P_j are on the same cycle we can deduce that P_i has not communicated with P_j on its present cycle. On the other hand, since by assumption both P_j and P_k are less than P_i, the latter communicates (on each cycle) with P_j and P_k in parallel. Thus, since P_i is willing to communicate with P_k it must also be willing to communicate with P_j. This contradicts the facts that the pair $\{P_i, P_j\}$ is conflict free and that P_j has an ungranted request to P_i. $\Box$

We can in fact show that Rule 1 is applicable to this network. It might interest the reader to prove this using Theorem 2 and much the same argument as above. $\Box$

In both of the above examples where we have been able to prove a network deadlock free, the crucial feature has been that no cycle of ungranted requests with equal variants can exist. In each case this followed only after examination of the processes. A rather more straightforward application of this theorem is where no such cycle exists because the network contains enough edges where strict variant inequality is maintained.

Note that cycles in communication graphs can be thought of as consisting of directed edges: an edge of the graph together with a direction. We can represent the directed edge from P_i to P_j as the ordered pair $\langle i, j \rangle$ just as we could represent the undirected edge as the set $\{i, j\}$.

Rule 2. If the network V satisfies the conditions of Theorem 6 and additionally E a set of directed edges from the communication graph of V such that every cycle in the graph contains at least one edge from E and such that whenever $\langle i, j \rangle$ in E and $\sigma = \langle s, \langle X_i, X_j \rangle \rangle$ is any state of $\{P_i, P_j\}$ then

$$P_i \xrightarrow{\sigma,\Lambda}\bullet P_j \quad \Rightarrow \quad f_i(s|\!\backslash \alpha P_i, X_i) > f_j(s|\!\backslash \alpha P_j, X_j) \quad \text{if } \langle i, j \rangle \in E$$

Then V is hereditarily deadlock free. $\Box$

The validity of this rule follows easily from Theorem 6.

Since the preconditions of Rule 1 trivially imply conflict freedom, the preconditions of this theorem are easily seen to be weaker than those of Rule 1. Thus the second rule we give is in some sense strictly stronger than Rule 1. Notice that, since there is no condition relating the values of the variants of elements of different essential components, it is possible to develop the variant of each of these components independently.

We give three examples of the use of Rule 2. The first is to a network where the possibility of having non-strict inequalities leads to much simpler variant functions than could have been used under Rule 1. The second and third examples are not conflict free, and so could not have been treated at all using Rule 1.

Example 4: The n dining philosophers. This problem is sufficiently well-known to need little introduction. A number of philosophers ($PHIL_0 \cdots PHIL_{n-1}$) sit at a circular table, and between each pair $PHIL_i$ and $PHIL_{i+1}$ lies $FORK_i$. (All arithmetic in this example will be modulo n.) In order to eat, a philosopher requires both neighbouring forks (left and right). Deadlock can occur when all philosophers pick up one fork simultaneously: none can acquire the second fork he needs until another philosopher releases it; but no philosopher will release a fork until he has eaten.

As stated earlier, one way to prevent deadlock occurring is to ensure that the network contains at least one left-handed philosopher (i.e., a $PHIL_i$ who will always seek to pick up $FORK_i$ before $FORK_{i-1}$) and one right-handed one. The rest may nondeterministically choose, on each visit to the table, which fork to seek first. The following processes describe the actions of $PHIL_i$ making a single visit to the table, when respectively he opts for the left or right fork first.

$$\begin{aligned} LEFTVISIT_i &= i.sits \rightarrow i.picksup.i \rightarrow i.picksup.i-1 \rightarrow i.eats \\ &\quad \rightarrow i.putsdown.i-1 \rightarrow i.putsdown.i \rightarrow i.getsup \rightarrow SKIP \\ RIGHTVISIT_i &= i.sits \rightarrow i.picksup.i-1 \rightarrow i.picksup.i \rightarrow i.eats \\ &\quad \rightarrow i.putsdown.i \rightarrow i.putsdown.i-1 \rightarrow i.getsup \rightarrow SKIP \end{aligned}$$

The left-handed philosopher $PHIL_l$ is thus defined

$$PHIL_l = LEFTVISIT_l; PHIL_l$$

and the right-handed philosopher $PHIL_r$ is

$$PHIL_r = RIGHTVISIT_r; PHIL_r.$$

When $i \notin \{r, l\}$, we have

$$PHIL_i = (LEFTVISIT_i \sqcap RIGHTVISIT_i); PHIL_i.$$

Fork processes are described

$$\begin{aligned} FORK_i &= (i.picksup.i \rightarrow i.putsdown.i \rightarrow FORK_i) \\ &\quad \Box(i+1.picksup.i \rightarrow i+1.putsdown.i \rightarrow FORK_i)\ . \end{aligned}$$

The alphabet of each process is just the set of all events used in its definition.

The component processes are trivially deadlock free and the network is conflict free because the communications between each pair follow a strict cyclic pattern (see [BR3], Lemma ..).

We choose the set $\{0,1\}$ (with its usual order) as our partial order, and the directed edges from $PHIL_l$ to $FORK_{l-1}$ and from $PHIL_r$ to $FORK_r$ as the set E over which strict inequality is required. (Clearly every cycle includes one of these.) The variant functions are, as we will see, extremely simple.

The variant function f_i of $FORK_i$ is defined

$$\begin{aligned} f_i(s) &= 0 && \text{if } i \in \{r+1, r+2, \ldots, l-2\} \\ f_i(s) &= 1 && \text{if } i \in \{l, l+1, \ldots, r-1\} \\ f_i(s) &= \begin{cases} 1 & \text{if } |s| \text{ is even} \\ 0 & \text{if } |s| \text{ is odd} \end{cases} && \text{if } i \in \{l-1, r\}. \end{aligned}$$

The variant of $FORK_i$ is thus either the constant 0 or the constant 1 unless it is one of the forks at the end of an edge in E, in which case it is 1 or 0 depending on whether it is "free" or not.

The variant function g_i of $PHIL_i$ is defined

$$\begin{aligned} g_i(s) &= 0 \text{ if } i \in \{r+1, \ldots, l-1\} \\ g_i(s) &= 1 \text{ if } i \in \{l, \ldots, r\} \end{aligned}$$

Thus all the philosophers have constant variant functions.

It is clear that the variant conditions are met on all edges other than those including $FORK_{l-1}$ and $FORK_r$, since on all such edges the processes at either end have equal constant variants. It is clear that a philosopher can only be waiting for a fork when he wants to pick the fork up, but the fork is in the possession of its other user. Since both of the two forks in question have value 0 when possessed by a user, the non-strict inequality is clear in this case. Indeed, when $PHIL_l$ or $PHIL_r$ is waiting for one of these forks the inequality is strict (as required), since these philosophers always have value 1.

If $FORK_{l-1}$ or $FORK_r$ is waiting for a philosopher other than $PHIL_l$ or $PHIL_r$, there can be no problem since the philosopher's variant is the constant 0. Similarly there is no problem when one of these forks are waiting for one of these two philosophers to pick them up, for the fork's variant is then 1. Finally, observe that since the only action that $PHIL_l$ performs between picking up $FORK_{l-1}$ and putting it down is the external action $l.eats$, there can never be an ungranted request from $FORK_{l-1}$ to $PHIL_l$ while the former's variant is 0. (A symmetric argument applies to $FORK_r$ and $PHIL_r$.)

Thus the preconditions of Rule 2 are met, so we can conclude that this network is deadlock free. The reader might like to verify that this example can be proved deadlock free by Rule 1, but will inevitably find that this proof is rather harder than the above.

As observed earlier, we cannot hope to prove the deadlock freedom of the well-known solution to the dining philosophers problem that involves a "butler" or "footman" process which prevents more than $n-1$ philosophers sitting down using any rule which establishes hereditary deadlock freedom, for the simple reason that this system is not. We will show in a later paper how this network can be dealt with by adding "invariants" to our armoury. The best that can be managed for this system using our present techniques is to use Theorem 6: variants will show that the only possible cycles are the well-known ones where each philosopher has one fork, and this cannot happen because of the structure of the processes used. □

As we have already observed, the preconditions of Rule 1 imply conflict freedom, which means that there is no hope of using that rule to prove deadlock freedom in networks which have conflict. The reason for conflict appearing in a correct network is almost invariably that a pair of processes which are fairly symmetric with respect to each other have two channels linking them, one for each to initiate some interaction with the other.

For example, if they are two nodes in a mail network, each might be in a position to send a message to the other. It is the authors' experience that most interesting small examples with conflict are trees and therefore best dealt with by Theorem 4. Perhaps this is because Rule 2 only allows us to treat networks where the edges over which conflict can appear contain no tree: thus any non-tree example must contain at least two different "modes" of communication. This is clearly illustrated in the following two examples.

Example 5: A message switching network. Suppose there are a number of potential senders of messages, each of whom might wish to send a message to any one of a class of receivers. It is possible to construct a network which implements the mail service they require out of binary switching nodes connected in a "butterfly" pattern. For simplicity we will assume that there are exactly 2^n senders and 2^n receivers. (A few simple generalisations of what follows extend this to arbitrary, and possibly different, non-zero numbers at each end.)

Between the ith sender and the ith receiver ($i \in \{0, \ldots, 2^n - 1\}$) there is a chain of n switch processes. In addition to an *in* and *out* channel (connected in the chain), a switch process has a *swin* and *swout* channel connected to another switch process (input to output). The rth process on chain number i is linked (via these extra channels) to the rth process in the (unique) chain whose index j differs in only the rth binary digit from i. On receiving a message on *in*, the rth switch process on chain i either passes it on down the same chain (*out*) or passes it over (via *swout*) to its linked chain depending on whether the rth binary digit in the destination process agrees with that of i. On receiving a message on *swin*, the process passes it to *out*. Figure 3 illustrates the connections in this network when $n = 3$.

From the point of view of proving deadlock freedom, we need not concern ourselves about the contents of the messages passing though the network. Indeed, we need not worry either about the routing algorithm described above, so long as we accept that a message entering a node on channel *in* may (so far as we are concerned) nondeterministically be sent either along *out* or along *swout*. (Of course, a system of processes with this behaviour is more nondeterministic than our actual network, so proving it deadlock free is certainly enough.) Thus, much as in the case of the systolic array, we will omit all details of actual messages from the process definition we give here, so retaining only details of synchronisations.

$$
\begin{aligned}
SWITCH(in, out, swin, swout) \;&=\; S, \text{where} \\
S \quad &= \quad (in \to ((out \to S) \sqcap S')) \\
&\qquad \Box(swin \to out \to S) \\
S' \quad &= \quad (swout \to S) \\
&\qquad \Box(swin \to out \to S')
\end{aligned}
$$

Notice that, when this process contains a message it wants to pass across its link, it retains the ability to accept a message from the linked process. This is to avoid the linked processes becoming deadlocked when each wants to relay a message to the other.

The result of combining $n \times 2^n$ of these switches together as described above is a strong conflict free network. It is not conflict free, for the pairs of linked switches are in conflict

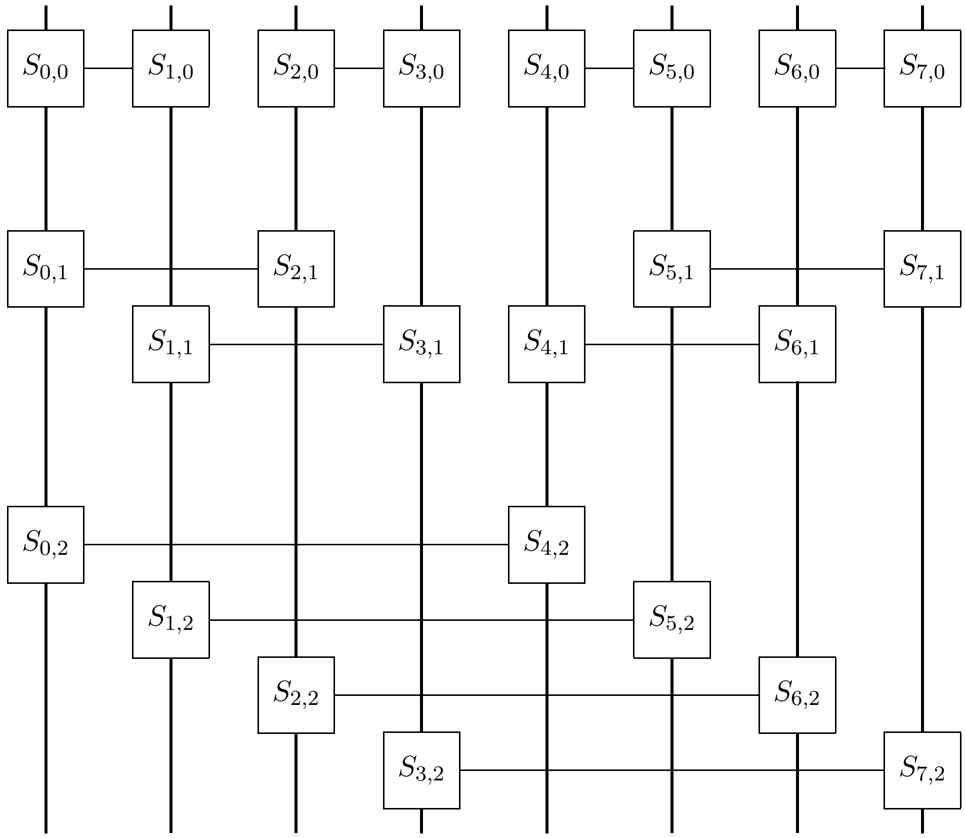

Figure 3: Message switching network for $n = 3$

when they are empty; if we had not made them able to accept input from the link when able to output to it, there would have been strong conflict.

Let E be the set of all edges linking nodes in the same chain: clearly every cycle includes an element of E. If $S_{i,r}$ is the rth switch on the ith chain, we define the variant functions:

$$f_{i,r}(s) \;=\; \begin{cases} r & \text{when the node contains no messages,} \\ -r & \text{when it contains one or more messages.} \end{cases}$$

That these functions work is clear when we observe that no process that contains a message can, in this network, have an ungranted request to one that is empty. Note that when a pair of linked switches on different chains are either both empty or both full, they have the same variant. □

Example 6: Adding a mail service to Example 2. Suppose we have designed a parallel network and proved it free of deadlock, but we would like to add some further communication links for some purpose. A good example would be the network of Example 2: where, in addition to the resource management we might like to implement a mail service between the user processes. In general it is extremely likely that such additions will lead to deadlock, even when the communications introduced are extremely simple. For example, suppose we introduce a mail channel between two processes, each of which is only prepared to communicate on this channel when it holds more than half of the resources: it is easy to see that deadlock can ensue.

Fortunately it is possible to add a mail service in such a way that it has a system of variant functions satisfying the conditions of Rule 2. Choose some tree interconnection pattern which spans all the user processes, either by linking them directly, by adding one or more mail server processes or by a mixture of the two. The idea is that the new edges thus introduced will be included in the set of edges over which equality of variant is allowed (i.e., the complement of E). Now implement a mail service over these edges in such a way that the following hold.

1. The network remains busy and triple disjoint.

2. The network remains strong conflict free. (In this context this means that there cannot be a pair of processes each of which is committed either to send a message to the other, or to receive a message from the other.)

3. No user process can execute any mail event while it holds a resource. (In other words, it can only use the new service while its variant, as defined in Example 2, is maximal.)

Then the augmented network is deadlock free.

To prove this we define the set E of edges to be all directed edges from user processes to resources. (Clearly this set includes at least one element of every cycle.) We then define variant functions as in Example 2 (thinking of any mail server processes as users which never use any resource), except that the value *big* is identified with $2 \times M + 1$ (rather than being greater than it as before). These functions satisfy the preconditions of Rule 2.

For suppose one process, P, has an ungranted request to another, Q. If neither is a resource then the edge between them is not in E and, by (3) above, P's variant is $2 \times M + 1$, which is the maximum of all possible variant values in the network (and hence certainly greater than or equal to that of Q).

If P is a resource and Q is a user process then the edge between them is not in E. If P is unused then its variant is $2 \times M + 1$ and exactly the same argument as used above applies; if P is in use then the properties of the variant inherited from Example 2 apply.

Finally, if P is a user process and Q is a resource, the arguments of Example 2 still apply.

One interesting feature of this example is the way in which we used variants as a way of discovering what the correct way of extending the existing network was, rather than just as a tool for proving an already constructed system. $\Box$

Finding useful general conditions under which similar manipulation of networks can be done safely is an interesting subject for future research. What seem to be the essential features that make the above example work were the existence of a proof of deadlock freedom of the original network (that of Example 2) with a set E of cycle-cutting edges that continued to work in the augmented network, and the fact that new communications were only possible when the original variants were maximal.

5 Postscript: Using invariants

The methods described so far can only prove a network deadlock free when it is *hereditarily* deadlock free. In a network which is deadlock free but not hereditarily so, there is some subnetwork which is restrained from getting into a deadlock state by one or more processes not in the subnetwork. This is clearly the case in the best-known such network, the dining philosophers with "butler". There is a clear sense in which the deadlock freedom of such a network can be rather less a locally checkable property than in the networks we have examined up to now: while examining the interactions of a pair of processes one might well have to consider the ways in which they are influenced by other elements of the network.

This having been said, it is often not too hard to identify the properties of a network that make it deadlock free, even though its subnetworks are not. (Of course, in the example quoted above, it is the behaviour of the butler.) It is clearly desirable that we should be able to incorporate such information into our methods, thereby enabling ourselves to prove these networks deadlock free.

Recall that when one uses the decreasing variant technique to prove that a WHILE loop terminates, one is only expected to prove that the variant decreases when certain conditions hold (namely, that the boolean condition of the loop is false, and that the loop invariant holds). For example, in the program

```
IF n >= 0 THEN
  WHILE n <> 0 DO
  BEGIN
    n := n - 1;
    m := m + 1;
  END
```

one is only expected to prove that the variant ($|\mathtt{n}|$) decreases when the invariant ($(\mathtt{n} \geq 0 \wedge (\mathtt{n} + \mathtt{m} = \mathtt{n}_0 + \mathtt{m}_0)$) holds and $\mathtt{n} \neq 0$). This suggests that it might be possible to use similar ideas to limit the cases where we have to prove a network variant decreases round a cycle.

There is no obvious analogue of the WHILE boolean in our world, except perhaps the ability of a process to communicate externally, which we have treated fully already. We will thus concentrate on the idea of using *invariants*. There are at least two distinct levels at which one can to use these: either to establish reasonable local behaviour or reasonable global behaviour. In this paper we will consider mainly the local case, where each process in the network has its own separate invariant.

The invariant of a network element will be a condition on those traces which the process can actually execute while the whole network is running. If one could prove that, in the context of the network, a process satisfied some invariant, one could restrict the domain of a process' variant to those traces (or corresponding maximal failures) satisfying its invariant. Furthermore, it would only be necessary to prove the deadlock-freedom of the elements of the network and the strong conflict freedom of the pairs while they satisfied their invariant.

This is done in a slightly different sense than before. One proves that in every state satisfying its invariant, each process is able to perform some action after which the invariant will still hold. This stronger form of "busy-ness", when proving strong conflict freedom, allows us to ignore any request whose satisfaction would mean that the requesting process' invariant no longer holds. (In proving the validity of this technique one replaces each process P_i in the network by the process $P_i \| d(I_i)$, where $d(I_i)$ is the deterministic process with the same alphabet as P_i which is at any time be prepared to communicate precisely those actions which keep its invariant I_i true.)

Of course one is obliged to prove, by considering the behaviour of the network as a whole, that each process' invariant actually is satisfied at all times. Notice that this proof technique can only be useful where the global behaviour of the network restricts the traces in which its individual components can engage.

The following example, in the style of [H], illustrates this type of reasoning.

Example 7: Vending machine. We consider three processes: a customer, a messenger and a vending machine arranged in a line. The vending machine gives chocolates in return for 5p coins but, due to a design flaw, will break (deadlock) if two coins are inserted without the first chocolate being removed.

$$VM \quad = \quad \mathit{in5p} \rightarrow (\mathit{outchoc} \rightarrow VM \Box \mathit{in5p} \rightarrow STOP)$$

The messenger knows nothing of this flaw, and faithfully carries any coin from the customer to the machine, and any chocolates from the machine to the customer.

$$MSGR \quad = \quad (\mathit{5p} \rightarrow \mathit{in5p} \rightarrow MSGR) \Box (\mathit{outchoc} \rightarrow \mathit{choc} \rightarrow MSGR)$$

Notice that the combination $MSGR \| VM$ can still deadlock, for, as defined, the messenger might take a second coin to the machine before bringing back the first chocolate. However the customer is mindful of this, and so carefully makes sure he has received the last chocolate before parting with any more money.

$$CUST \quad = \quad \mathit{5p} \rightarrow \mathit{choc} \rightarrow CUST$$

It is not too hard to see that the network $CUST \| MSGR \| VM$ is deadlock free. However, notice that the subnetwork $CUST \| MSGR$ can also deadlock, for if the first thing the messenger does is to take a 'free' chocolate to the customer he is unable to accept it and the messenger is unable to accept the coin he offers.

To prove this network deadlock free we need invariants. Because our network is a tree it will be sufficient, after establishing invariants, to check that the network is busy and strong conflict free in the sense outlined above.

The invariant of the vending machine process is

$$I_{VM} \;=\; \#outchoc \leq \#in5p \leq \#outchoc + 1$$

where, for example $\#in5p$ denotes the number of times that event has occurred up to any given time. That of the messenger is

$$I_{MSGR} \;=\; \#choc \leq \#outchoc \leq \#in5p \leq \#5p \leq \#choc + 1.$$

The customer's invariant is *TRUE* (i.e., it imposes no constraint). Since at every time the messenger and machine have communicated exactly the same number of *outchoc* and *in5p* events, it is clear that if the messenger satisfies its invariant then the other two satisfy theirs.

It is simple to verify that, in the sense above, the network is busy and conflict free with respect to these invariants. That the invariants do in fact hold follows easily from the inequalities below, which are derived from the definitions of the individual processes.

$$\begin{array}{cl}
\#choc \leq \#5p \leq \#choc + 1 & \text{from } CUST \\
\#choc \leq \#outchoc \leq \#choc + 1 & \text{from } MSGR \\
\#in5p \leq \#5p \leq \#in5p + 1 & \text{from } MSGR \\
\#outchoc \leq \#in5p & \text{from } VM
\end{array}$$

An interesting extension of this network is obtained by replacing the single messenger by a row of messengers which pass the coins and chocolates backward and forward in the manner of a chain of people conveying buckets of water to a fire and the empty buckets back. If all the messengers have the same definition as the one above (except for renaming of communications) the resultant network is deadlock free (the proof being a straightforward extension of the above), but every proper connected subnetwork of size greater than one can deadlock. There is an interesting contrast between this and the "deadlocked chain" example of [BR2,3], which has almost exactly the opposite properties. □

Even though the above technique works for proving deadlock freedom, it would be a grave mistake to rely on it regularly. On the one hand it will probably be easier in practice to design the elements of a network so that their local behaviour is good than to prove this from the global properties of the network. After all, the behaviour of one or a pair of processes is very much easier to understand and control than that of the whole network, so it would be strange to expect to use the latter to control the former. Also it must be wrong to get round a bug in one part of a program by designing a second part of the program to avoid it, rather than eliminating the bug at source. Such an approach would, for example, make re-using parts of programs far more dangerous. For example, in network above the vending machine should be replaced by one that cannot deadlock, and the messenger(s) should be replaced by one(s) that know that every chocolate is preceded by a coin.

$$\begin{array}{rcl}
VM' & = & in5p \rightarrow (outchoc \rightarrow VM') \\
MSGR' & = & 5p \rightarrow in5p \rightarrow outchoc \rightarrow choc \rightarrow MSGR'
\end{array}$$

The astute reader will have noticed that, in each case, we have replaced each process P_i by one that is equivalent to its parallel composition with the deterministic CSP process $d(I_i)$ described above, that will communicate any action which preserves its invariant. Note that the revised network is now *hereditarily* deadlock free. Given a set of invariants which ensure the local good behaviour of a network, this provides a general technique for converting it into a "well-constructed" network.

Of course it would be far preferable to include suitable invariants in the initial specifications from which the elements of the network are developed, rather than having to modify their definitions as above.

Thus there is a sense in which we prefer to regard this form of invariant as a guide to the correct (re-)construction of a network, rather than as something closely linked to our ideas about variants. We argue that the style of proof above will not be necessary for a properly constructed system.

The possible uses of *global* invariants for networks where each state of individual processes is reachable but which fail to be hereditarily deadlock free is briefly discussed in the Conclusions section below.

6 Conclusions and Prospects

The aim of this paper has been to show how a few simple ideas can make deadlock analysis both clear and tractable. We believe that the methods we have described are applicable to a large proportion of real systems, perhaps after a certain amount of redesign along the lines set out in the postscript. We also believe that these methods and the insight into deadlock produced by our investigations should be useful for producing new networks which are deadlock free by design.

The methods we have presented are those which we believe will be most commonly useful in practice. However, as we observed earlier, they are not complete in that some deadlock free networks cannot be so proved using them. There are a few distinct situations under which they fail; in the following paragraphs we identify these and indicate what modifications will be required to get round them.

Sometimes a network can be deadlock free even though it contains a cycle of ungranted requests, usually because one or more processes in the cycle have alternative requests whose eventual satisfaction will break the cycle. Our existing methods can deal with this situation when this alternative request is outside the vocabulary of the network (for the process, formally, cannot then be making an ungranted request) or when the alternative request is along a conflict free disconnecting edge (by Theorem 5). However, in other cases we would need to extend our methods. A typical example would be where a number of user processes share some resource using a "token ring" (see [BR2]) and have some other mode of interconnection which stops the edges from the ring elements to the users being disconnecting. This network now initially has a cycle of ungranted requests (the ring elements listening for a request from their anticlockwise neighbour for the token), but each element is also waiting for its user process. It is fairly easy to extend our method to deal with this type of network: under certain circumstances, where a process has requests to several of its neighbours, we may select which one must have a lesser variant. It is possible to find several extensions of Rule 2 which take advantage of this fact: they either use functions to do the selecting or examine larger "localities", typically the set of a process and all its neighbours.

We have already observed that our existing rules can never prove the deadlock freedom of non-hereditarily deadlock free networks. Some of these can be improved to being hereditarily deadlock free along the lines described in the postscript, but others such as the "butler" version of the dining philosophers are more subtle. These are busy, strong conflict free and have every state of each process reachable within the context of the whole network (so that no individual process invariants can help us by excluding unwanted ones).

Such a network can only work because, exactly as happens in this version of the dining philosophers, one part of the network keeps another from entering a deadlocked state. It seems inevitable that, in general, arguments for this type of behaviour must be non-local in nature (it is easy to imagine, for example, a system of dining philosophers interacting via a number of intermediaries with a butler). In such examples perhaps the best we can hope for is that we can express the essential property which is preventing deadlock as a global invariant of the network (for the dining philosophers this would be "not all philosophers are sat down"). We can then take advantage of this property when constructing the systems of variants: one way of doing this is to construct several different systems where the variant functions are in fact partial functions, but proving that in every global state satisfying the invariant, at least one of the systems of variants is totally defined. (For the dining philosophers we could define one system of variants for each philosopher, for use when he is the one prevented from sitting down last.) The extent to which one can usefully adopt this version of the invariant/variant approach must depend on the difficulties of individual examples.

In addition to the types of network mentioned above where it is obvious that Rules 1 and 2 cannot apply, one occasionally comes across a network which, though hereditarily deadlock free and without cycles of ungranted requests, still seems to be too much for these methods. The authors have only come across one such example: the message passing ring in [BR3] (mentioned in the present paper after Theorem 2) where each element is a restricted double buffer (though Theorem 6 may still be deployed to good effect).

In this message passing ring, it turns out if, whenever a pair of processes communicate they tell each other the current value of their variant (i.e., that prior to the current communication), variants for Rule 1 can easily be constructed because the traces of individual processes then contain sufficient information about neighbours. It seems that this is because the relation described in Theorem 2 is in some sense "refined" by this transformation.

We intend to continue our investigations into deadlock by developing some of the above ideas further and trying to achieve as good an understanding as possible of the relative capabilities and difficulties of the various techniques. We also intend to investigate ways of proving other properties of networks by local methods, particularly related ones such as absence of livelock and starvation. We also intend to investigate the ways in which, as described in the last paragraph, trivial transformations of processes that do not change the basic communication pattern can be used in aiding proofs of network properties.

When other authors have addressed the problem of proving deadlock freedom in a general way they have tended to describe methods of proof which are essentially global. Given that they were generally looking for complete methods, this is understandable. Usually these have involved proving some invariant of the global state that precludes the blocking of all processes [AFR, OG, S]. It should often be possible to integrate our methods with fairly general techniques such as these, in that the invariants to be proved could be just the preconditions of our rules.

Perhaps the most similar approach to ours has been that of Chandy and Misra [CM], who have described different measures termed "priorities" for use in proving deadlock freedom. In the terminology of the present paper a set of priorities for a network, given a linear order, is a collection of maps from the vocabulary of the network to the linear order (one map for every global state) such that, whenever a process is blocked it must be willing to communicate the event of minimum priority in the intersection of its alphabet and the vocabulary of the network. A triple-disjoint, busy network permitting a set of priorities is deadlock free. This method is easily seen to be complete. In general, obtaining a set of priorities may be difficult, as it entails global checking; however, for some networks within the framework of this paper, simplifications to this end can be made. For a conflict-free network, it suffices to have priorities defined only on edges of the communication graph for which there exists an ungranted request. Moreover, for a network amenable to Rule 1, we can construct a set of priorities locally in the following way. Firstly, as any partial order may be extended to a linear order, we will assume, without loss of generality, that the range of the variants yielded by Rule 1 is a linear order. Then, if a process in the network is blocked, assign a priority to each edge incident on it on which there is an ungranted request, the value of the variant of the process being waited upon. The case of Rule 2 and non conflict free networks (which are, however, strong conflict free) is a little harder because we would have to assign priorities to individual actions rather than edges, and to take account of the large scale topology of a network. Using the concepts of conflict freedom and strong conflict freedom has enabled us to abstract away from individual events and assign priorities to *processes*.

Acknowledgements

We would like to thank C.A.R. Hoare for a large number of helpful comments and suggestions on drafts of this paper. Michael Goldsmith discovered that an earlier version of Corollary 3 in Example 3 could be strengthened; this led directly to the discovery of its present form.

References

[AFR] K.R. Apt, N. Francez and W.P. De Roever, *A Proof System For Communicating Sequential Processes.* ACM TOPLAS Vol. 2, No. 3 (July 1980) 359-385

[BHR] S.D. Brookes, C.A.R. Hoare and A.W. Roscoe, *A Theory of Communicating Sequential Processes.* Journal of the ACM Vol. 31, No. 3 (July 1984) 560-599.

[BR1] S.D. Brookes and A.W. Roscoe, *An Improved Failures Model for Communicating Processes.* Proceedings of the Pittsburgh Seminar on Concurrency, Springer LNCS 197 (1985) 281-305

[BR2] S.D. Brookes and A.W. Roscoe, *Deadlock Analysis in Networks of Communicating Processes.* Logics and Models of Concurrent Systems (ed. K.R. Apt) NATO ASI series F. Vol. 13. Springer (1985) 305-324

[BR3] S.D. Brookes and A.W. Roscoe, *Deadlock Analysis in Networks of Communicating Processes II.* To appear.

[CM] K.M. Chandy and J. Misra, *Deadlock Absence Proofs for Networks of Communicating Processes.* Information Processing Letters Vol. 9, No. 4 (November 1979)

[D] E.W. Dijkstra and C.S. Scholten, *A Class of Simple Communication Patterns.* Selected Writings on Computing. EWD643. Springer-Verlag (1982)

[H] C.A.R. Hoare, *Communicating Sequential Processes.* Prentice-Hall International, 1985.

[OG] S.S. Owicki and D. Gries, *Verifying Properties of Parallel Programs: An Axiomatic Approach.* Communications of the ACM Vol. 19, No. 5 (May 1976) 279-285

[S] J. Sifakis, *Deadlocks and Livelocks in Transition Systems.* Mathematical Foundations of Computer Science, Springer LNCS 88 (1980) 587-599