Deadlock
and
Deadlock Freedom

by

Naiem Dathi

The Queen's College

Thesis submitted for the degree of Doctor of Philosophy
Oxford University

Trinity Term 1989

Abstract

Title: *Deadlock and Deadlock Freedom*
Author: *Naiem Dathi*
College: *The Queen's College, Oxford*
Degree: *Doctor of Philosophy*
Term of Submission: *Trinity 1989*

We introduce a number of techniques for establishing the deadlock freedom of concurrent systems. Our methods are based on the local analysis (or at worst a directed global analysis) of networks. We identify the relationships between these techniques and the range of their application within a framework of deadlock freedom types that we have defined. We also show that the problem of proving total correctness may be translated to one of proving deadlock freedom, with the consequence that our techniques for proving deadlock freedom may be utilised to effect a total correctness proof.

To my parents, Ali and Ruby,
brother Naj and sister Nuzy

Acknowledgements

First of all, I would like to thank Bill Roscoe for his guidance, encouragement and responsive supervision. More thanks than can be mentioned are due to Marie for her companionship and unfailing support. Finally, I would like to thank the Science and Engineering Research Council for their financial support.

Contents

Introduction

Deadlock occurs in a concurrent system when no further action can take place. This is usually because even though each component process is in a state in which it wishes to communicate, its neighbours however are not willing to accept this communication. Hence the evocative term "deadly embrace". This is a common problem in distributed systems and 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 use of concurrency not only introduces the possibility of pathological behaviour such as deadlock, but 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 and hence proofs of properties they possess are imperative, because the fact that 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 as 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. Many of the existing methods in the literature, such as [AFR, CM1, LG, OG, S] have taken this approach for the sake of completeness. Although in [A1, A2] attempts have been made to facilitate such analyses, the methods have remained intrinsically global and thus the degree of difficulty has not been diminished.

In this work our prime concern will be with methods that require only local analysis (examination of only single processes and pairs of processes

which communicate directly) or at the very most, directed global reasoning. This work continues the attempt begun in [BR2, BR3], and with this author in [RD], to obtain such techniques. As we shall see, this approach leads to rules for proving the absence of deadlock which are reasonably easy to apply in practice, and we display several examples of their use. The examples we shall give, although some are in an abstract form, are generally based on real situations. Many of them are similar in flavour to ones that may be found in [G, MS1, MS2, T].

We assume a certain familiarity with the version of CSP described in [H1, BHR, BR1], and the basic properties of its operators. The mathematics of this paper 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 work, although its basic structure is described in the Appendix. Even though we have cast our work in this particular framework, we imagine that it will transfer readily to other models of concurrency.

In Chapter 1 we meet the basic concepts (as first introduced in [BR2, BR3, RD]) which we shall require. This includes the representation of deadlock and deadlock freedom in the failures model and the definition of certain notions that apply to networks of processes, extracting the information relevant to deadlock analysis.

In the second Chapter we produce a framework within which to judge the capabilities of particular proof rules. This is carried out by defining and relating certain forms of deadlock freedom which seem to arise naturally. Within this context, we analyse the proof rule from [CM1] and the main proof rule from [RD].

Chapter 3 is concerned with providing extensions of the main result in [RD] which are common in spirit to it, but overcome many of its limitations as exposed in the previous chapter. The relationship between these new rules is also explored with a view to gaining a precise outline of their suitability in any particular case.

It is clear that the deadlock freedom of some systems is due inherently to some global property. In such cases, local checking alone cannot suffice. In Chapter 4 we see how such non-local checking can be conducted in a directed way so that the construction of any deadlock freedom proof is not made unnecessarily complicated. This is done by allowing the rules developed in the last chapter to access the necessary further information required to prove absence of deadlock. We then show some completeness results that

ensue from this generalisation of the rules. The chapter concludes with some simple transformations of networks that in general make deadlock analysis more tractable, as well as taming some "difficult" networks.

We turn to the more general question of total correctness in Chapter 5. This entails a brief but necessary discussion of divergence freedom together with the presentation of a useful technique to prove this. We then show that proving an implementation correct (in a sense made precise here) is equivalent to proving a certain deadlock freedom result. In particular, this means that we may use the rules developed above to check that an implementation meets a specification.

The final section contains some concluding remarks together with suggestions for future research.

Chapter 1

Introduction to Deadlock Analysis

This chapter is concerned with providing the foundations required for the study of deadlock and deadlock freedom.

As mentioned earlier, the semantics of CSP we shall use is that of [BR1] and in the first section we show the simple way in which deadlock can be represented in this model.

From the point of view of the user of a divergent program, such a system is deadlocked because it will never communicate with him again. In fact it is worse than that because the user can never detect that this is the case. However, divergence is operationally quite different from deadlock and different methods are required to prove the absence of divergence (see Section 5.2) and the absence of deadlock. We will therefore generally assume (unless stated otherwise) that all the processes we meet are already known to be divergence free (have an empty divergence set) and are thus fully characterised by their failure sets.

Another point to note is the question of *alphabet.* The CSP parallel operator depends crucially on this idea: each process in a network has an alphabet of communications, and no communication can occur until all the processes with it in their alphabet are willing to participate. Two methods have been used in the literature for introducing these alphabets. In [BHR, BR1] these were introduced explicitly into the the parallel operator, with $(P_A \|_B Q)$ denoting the parallel composition of the processes P, Q which have alphabets A, B respectively. In [H1] however, all processes are defined in such a way that each has an alphabet (usually, though not invariably, the

set of communications which it may wish to make) and thus alphabets need not be introduced explicitly in the parallel operator. Here like [BR3, RD] we shall follow [H1]. The alphabet of a process P will be denoted by αP with the traces, refusals and divergences of P being composed solely from the elements of αP.

In the second section we introduce the basic concepts from [BR2, BR3, RD] which in one form or another are essential for carrying out deadlock analysis in distributed systems. This includes the formal definition of an *ungranted request* (from [RD]) which can be considered as the building block of deadlock.

In the final section, we recall some results from the references mentioned above for proving the deadlock freedom of networks. This includes the main proof rule in [RD], the Variant Theorem (Theorem 1.3.5), which is of particular importance to us and to which much reference will be made in later chapters.

1.1 The Failures Model and Deadlock

This section summarises the basic facts we will need to know about the representation of deadlock in the failures model.

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

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

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 to permit processes that have terminated to be treated less harshly. In order to accommodate this more complex form, small modifications would have to be made to most of the definitions and results that follow. But as none of the example processes we use until the final section of Chapter 3 can terminate, we postpone these amendments till then.

Since the more a process P refuses after a trace, the more likely deadlock becomes, it is sufficient when considering potential deadlocks to consider failures $(s, X) \in \mathcal{F}[\![P]\!]$ such that X is maximal in $\mathit{refusals}(P \underline{\text{ after }} s)$. We will denote the maximal failures of P in the sense above by $\tilde{\mathcal{F}}[\![P]\!]$. We note

that for every failure of a process there is at least one corresponding maximal failure (where the traces of the two failures are the same and the refusal set of the former is a subset of the refusal set of the latter). The existence of such a maximal failure in general (i.e. when an alphabet is uncountable) relies on both axiom (F5) of the failures model (see Appendix) and the axiom of choice.

The following two laws are often very helpful in establishing that a process is deadlock free. The first is an invaluable aid in proving the deadlock freedom of individual processes that make up concurrent systems. It will allow us to deduce the deadlock freedom of most of the component processes in our example systems.

Lemma 1.1.1 Suppose the definition of the process P uses only the following syntax:

$$P ::= SKIP \mid a \rightarrow P \mid P; P \mid P \,\Box\, P \mid P \sqcap P \mid f(P) \mid p \mid \mu p.P$$

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.

Lemma 1.1.2 If $P \backslash C$ is divergence free, then it is deadlock free if and only if P is.

The latter result 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. Using this law, Lemma 5.1.1, the basic properties of parallel composition 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 system with a single hiding operation at the outermost syntactic level. Thus, from the point of view of deadlock analysis, all such systems are equivalent to a system without any hiding at all.

1.2 Networks

We now turn our attention to cases where many processes are working in parallel. If P_i are processes for $1 \leq i \leq n$, we denote their parallel composi-

tion by

$$\|_{i=1}^{n} P_i \qquad \text{or} \qquad P_1 \| P_2 \| \ldots \| P_n \quad .$$

The alphabet of $\|_{i=1}^{n} P_i$ 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 communication with the environment. When the P_i are all divergence free this operator is defined as follows:

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

where $s \restriction \alpha P_i$ denotes the sequence formed from s by removing all elements not in αP_i. Observe 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 $1 \leq m < n$,

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

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

Definition 1.2.1 A *network* V is a finite sequence $\langle P_1, \ldots, P_n \rangle$ where each P_i is a process. The corresponding process $\|_{i=1}^{n} P_i$ is denoted $PAR(V)$ and we say that V is *deadlock free* exactly when $PAR(V)$ is.

We will often denote $\alpha PAR(V)$ by αV. Note that, in view of the associative and symmetric properties of $\|$, we have that

$$PAR(V) = PAR(\langle PAR(U_1), \ldots, PAR(U_m) \rangle)$$

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

Definition 1.2.2 A *state* σ of a network $V = \langle P_1, \ldots, P_n \rangle$ consists of a pair $\langle s, \langle X_1, \ldots, X_n \rangle \rangle$ (which we shall sometimes abbreviate to $\langle s, \underline{X} \rangle$) such that $s \in (\bigcup_{i=1}^{n} \alpha P_i)^*$ and $(s \restriction \alpha P_i, X_i) \in \tilde{\mathcal{F}}[\![P_i]\!]$ for every i. We shall let $\mathcal{S}(V)$ denote the set of all the states of V.

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 the particular case of a network consisting of a single process P, we shall often abuse notation and confuse an element of $\tilde{\mathcal{F}}[\![P]\!]$ with the corresponding element of $\mathcal{S}(\langle P \rangle)$.

The next definition isolates the states of a network which exhibit deadlock.

Definition 1.2.3 If a state $\delta = \langle s, \langle X_1, \ldots, X_n \rangle \rangle$ of $V = \langle P_1, \ldots, P_n \rangle$ is such that

$$\bigcup_{i=1}^{n} X_i = \alpha V$$

then we say δ is a *deadlock state* of V.

Then it follows that a network is deadlock free if and only if it has no deadlock states.

We will generally restrict our attention to networks where no event requires the participation of more than two processes, that is all communication is point to point. It is also reasonable to suppose that the component processes of a network whose deadlock freedom we seek to prove are deadlock free. These two requirements are set out in the definition below.

Definition 1.2.4 A network $V = \langle P_1, \ldots, P_n \rangle$ is said to be *triple-disjoint* if $\alpha P_i \cap \alpha P_j \cap \alpha P_k = \emptyset$ whenever i, j and k are all distinct. A network will be called *busy* if all its component processes are deadlock free.

If W is a not necessarily proper but non-empty subsequence of a permutation of the sequence of processes representing a network V, we will term it a *subnetwork* of V. Notice that some of the communications of W which were with other elements of V may become external communications of W, that is communications that do not require the agreement of any other component process in execution. For any state σ of V there is a corresponding state of W. We shall denote this state by $\sigma\restriction W$. In the special case where $W = \langle P \rangle$ we may also write this as $\sigma\restriction \alpha P$.

We shall say that a property of a network is *hereditary* if it holds of all its subnetworks. For example, a triple-disjoint network is hereditarily triple-disjoint and a busy network is hereditarily busy.

Some useful static information of the following form can be gleaned from a preliminary examination of a network.

Definition 1.2.5 Let $V = \langle P_1, \ldots, P_n \rangle$ be a network. We may associate a graph with V, 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 processes are joined in the graph if and only if there is a possibility of communication between them. The *vocabulary* of V is defined to be the set $\bigcup\{\alpha P_i \cap \alpha P_j \mid i \neq j\}$.

The vocabulary of a network is an important set with reference to deadlock analysis because it is precisely the set of events for which agreement is necessary. If a network is deadlocked, it is clear that no component process can be willing to communicate outside of this set. Observe that the vocabulary of a subnetwork is always a subset of the vocabulary of the subsuming network.

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 1.2.6 Let $\sigma = \langle s, \underline{X} \rangle$ be a state of the network $\langle P_1, \ldots, P_n \rangle$. 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

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

that is if P_j is unwilling to respond to P_i's request.

Sometimes we are only interested in ungranted requests when neither process is able to communicate outside some set Λ, often 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

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

We will write

$$P_i \xrightarrow{\sigma} P_j \quad \text{or} \quad P_i \stackrel{\sigma}{\Longrightarrow} P_j$$

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

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

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

As the information required to determine whether processes have requests or ungranted requests can also be gained from the restriction of states to relevant subnetworks, we shall often use these restricted forms of states to annotate such relations. In the case of a request, this could conceivably even be the maximal failure of the requesting process.

Notice that if $\Gamma \subseteq \Lambda$, then $P_i \xrightarrow{\sigma,\Gamma}\bullet P_j \Rightarrow P_i \xrightarrow{\sigma,\Lambda}\bullet P_j$. Furthermore, when $\alpha P_i \cup \alpha P_j \subseteq \Lambda$, we see that $P_i \xrightarrow{\sigma,\Lambda}\bullet P_j \Leftrightarrow P_i \xrightarrow{\sigma}\bullet P_j$. Similar observations, of course, are true of strong requests.

We can consider deadlock as a chronic accumulation of ungranted requests. From the point of view of a single component process, this manifests itself in the following way.

Definition 1.2.7 Let $\langle P_1, \ldots, P_n \rangle$ be a triple-disjoint network with vocabulary Λ. P_i is said to be *blocked* in a state σ if

(i) $P_i \xrightarrow{\sigma} P_j$ for some j, and

(ii) $P_i \xrightarrow{\sigma,\Lambda}\bullet P_j$ whenever $P_i \xrightarrow{\sigma} P_j$.

The next result categorises deadlock states in these terms.

Lemma 1.2.1 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.

We now define some properties of a network which relate to, or are symptomatic of deadlock in a large class of networks.

Definition 1.2.8 In the network $\langle P_1, \ldots, P_n \rangle$ with vocabulary Λ, the sequence of distinct process indices $\langle i_0, \ldots, i_{r-1} \rangle$ 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). If in addition $r \geq 3$, we say $\langle i_0, \ldots, i_{r-1} \rangle$ is a *proper* cycle of ungranted requests. 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 proper cycle of ungranted requests corresponds to a cycle in the communication graph. The case of a non-proper cycle of ungranted requests (i.e. of length two), where each of a pair of processes asks the other for a communication, but where they cannot agree on anything, is sufficiently different to warrant special consideration. The next section is devoted to this.

1.3 Networks and Conflict

The notion of conflict between pairs of processes was introduced in [BR3], and was used there and in [RD] to prove results about deadlock. In this section we summarise the main ideas from these papers.

Definition 1.3.1 Suppose Λ is a set of communications. The processes P,Q are said to be *in conflict with respect to* Λ in the state σ of $\langle P, Q\rangle$ if and only if

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

They are *in strong conflict with respect to* Λ if, additionally

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

$\langle P, Q\rangle$ is said to be *conflict free* (respectively *strong conflict free*) *with respect to* Λ if it has no conflicts (respectively strong conflicts) with respect to Λ. We will say a network V is *conflict free* (respectively *strong conflict free*) if each of its subnetworks of size two is conflict free (respectively strong conflict free) with respect to the vocabulary of V.

A conflict is precisely the "non-proper cycle of ungranted requests" described at the end of the last section. A strong conflict is one where one of the pair of processes involved is completely blocked by the other. The communication pattern of most practical parallel systems are fundamentally conflict free in that, even though a given version of a program is not, it can trivially be redesigned to be so (see [BR3] for an example). For example, any pair of processes connected by a single occam-like channel is 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 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.

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

Theorem 1.3.1 ([BR3, RD]) A deadlock state of a triple-disjoint, busy, strong conflict free network has a proper 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.

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 next lemma reveals some simple methods which facilitate the deduction of conflict freedom for networks.

Lemma 1.3.2 ([BR3]) Suppose $\langle P, Q\rangle$ is such that $\alpha P \cap \alpha Q \subseteq \Lambda$.

(i) $\langle P, Q\rangle$ is conflict free with respect to Λ whenever $|\alpha P \cap \alpha Q| \leq 1$.

(ii) $\langle P, Q\rangle$ is conflict free with respect to Λ if there is an infinite trace $u \in (\alpha P \cap \alpha Q)^\omega$ such that for all $s \in traces(P) \cup traces(Q)$ we have $s\restriction(\alpha P \cap \alpha Q) \leq u$. (When u has the special form t^ω for some finite trace t, this is essentially a cyclic communication condition.)

(iii) $\langle P, Q\rangle$ is conflict free with respect to Λ if there is a function

$$\psi : \{s\restriction(\alpha P \cap \alpha Q) \mid s \in traces(P) \cup traces(Q)\} \rightarrow \{+, -\}$$

such that whenever $\psi(s\restriction\alpha Q) = +$ and $(s, X) \in \tilde{\mathcal{F}}[\![P]\!]$ where furthermore $X \not\supseteq \alpha Q$ and $\alpha P - X \subseteq \Lambda$ then

$$X \cap \{a \mid \exists t : traces(Q).\, t\restriction(\alpha P \cap \alpha Q) = (s\restriction(\alpha P \cap \alpha Q))\langle a\rangle\} = \emptyset \quad ,$$

and the similar corresponding condition holds when $\psi(s\restriction\alpha P) = -$ and $(s, X) \in \tilde{\mathcal{F}}[\![Q]\!]$.

Clearly, the methods above also show strong conflict freedom as this is always implied by conflict freedom. It is not hard to find conditions which entail strong conflict freedom, for instance if each process when it is willing

to communicate is willing to communicate with more than one process, then there can be no strong requests and hence no strong conflicts.

It has long been known that the deadlock analysis of "tree" networks (networks with no cycles in their communication graph) is especially easy. In practice, many parallel networks are of this form (for example pipeline systems and binary trees). The following corollary of Theorem 1.3.1 is usually all that one needs to prove their deadlock freedom.

Corollary 1.3.3 ([BR3]) A triple-disjoint, busy, strong conflict free network whose communication graph is a forest is deadlock free.

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 theoretical 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 forest are its individual processes, and that the essential components themselves always form a forest 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.

Lemma 1.3.4 ([BR3, RD]) Suppose V is a triple-disjoint network with subnetworks $V_1, \ldots, V_k$ corresponding to its essential components and where the two element subnetworks corresponding to disconnecting edges are conflict free. Then if each of the V_i is deadlock free, so is V.

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.

However, even a small network may well have many cycles in its communication graph often making the proof of deadlock freedom difficult. The following method from [RD] is one that is well-suited to such networks which have in addition the property that all the component processes have similar patterns of behaviour. These include a wide range of networks as can be seen by the examples treated there. This is an important theorem for us as the proof rules pertaining to deadlock freedom which we shall derive later will be similar in style to this one.

Theorem 1.3.5 (Variant Theorem [RD]) Let $V = \langle P_1, \ldots, P_n \rangle$ be a triple-disjoint, busy network with vocabulary Λ. Suppose that there exist functions

$$f_i : \tilde{\mathcal{F}}[\![P_i]\!] \to \Pi \qquad (1 \leq i \leq n)$$

where $(\Pi, >)$ is a partial order, such that whenever σ is a state of the subnetwork $\langle P_i, P_j \rangle$ then

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

Then V is deadlock free.

The preconditions of this rule are local, requiring at most the analysis of only a pair of processes at any one time. It is also easy to see that its preconditions imply the conflict freedom of a network for which they hold. The functions contained in the theorem are termed *variant functions*, in an analogy with those used to prove the termination of loops in standard sequential programming languages. We shall use the same terminology for functions which play a similar role in the proof rules for deadlock freedom to be expounded in later chapters.

Chapter 2

Forms of Deadlock Freedom

This chapter relates certain types of deadlock freedom for triple-disjoint, busy networks which hold when the networks are amenable to various proof techniques. These relationships provide a framework in which to judge the relative capabilities of proof techniques for deadlock freedom, as well as aiding in the choice of a particular technique for a particular network.

The rest of this chapter is organised as follows. In the first section we introduce the type of deadlock freedom, with allied proof technique, discussed in [CM1]. We show how for a conflict free network that method may be simplified. The second section presents three more types of deadlock freedom (one of these from [BR3]) . All the varieties of deadlock freedom defined are then related or contrasted, by the statement and proof of an implication or by the use of a counter-example, as appropriate. The final section concentrates on the classes of networks which are amenable to the the Variant Theorem.

2.1 CM-deadlock Freedom and Priorities

We start with some definitions based on those in [CM1]. They vary from the ones there in two respects. Firstly, the work there was not couched in any formal model of concurrency and thus we have had to translate the definitions into the failures model via our notation. Secondly, termination of processes was taken into account there, but we shall omit it here for clarity's sake.

Definition 2.1.1 Let $V = \langle P_1, \ldots, P_n \rangle$ be a triple-disjoint, busy network with vocabulary Λ. P_i is said to be *CM-blocked* in a state σ of V if

(i) $P_i \xrightarrow{\sigma} P_j$ for some j,

(ii) $P_i \xrightarrow{\sigma}\bullet P_j$ whenever $P_i \xrightarrow{\sigma} P_j$, and

(iii) $\alpha P_i - X_i \subseteq \Lambda$.

Clearly a blocked process is CM-blocked but the converse is untrue, as a CM-blocked process may be waiting on a process which is willing to do an event outside the vocabulary of the network.

Definition 2.1.2 Let W be a subnetwork of the triple-disjoint, busy network V. We say W is *CM-deadlocked* in σ if

(i) Every process in W is CM-blocked in σ, and

(ii) P_j is in W whenever P_i is in W and $P_i \xrightarrow{\sigma} P_j$.

Before we proceed to some more definitions we present a lemma that characterises CM-deadlock in our terminology.

Lemma 2.1.1 Let V be a triple-disjoint, busy network with a subnetwork W. Then if σ is a state of V with corresponding state δ of W, W is CM-deadlocked in σ if and only if δ is a deadlock state of W.

Proof. Let Γ, Λ be the vocabularies of W, V respectively. Firstly, assume W is CM-deadlocked in σ. Consider any process P_i in W. From Definition 2.1.2, P_i is CM-blocked in σ and so there is a process P_j such that $P_i \xrightarrow{\sigma} P_j$. Using Definition 2.1.2 again we note that P_j must be in W. Thus we deduce that for any process P_i in W there is some process P_j also in W such that $P_i \xrightarrow{\delta} P_j$. Using Definition 2.1.1, we know $\alpha P_i - X_i \subseteq \Lambda$. So if $x \in \alpha P_i - X_i$ it follows that $x \in \alpha P_i \cap \alpha P_k$, $i \neq k$, for some process P_k in V. Then $x \in (\alpha P_i - X_i) \cap \alpha P_k$ or $P_i \xrightarrow{\sigma} P_k$ and so using Definition 2.1.2 yields that P_k is in fact in W. So $x \in \alpha P_i \cap \alpha P_k$ where both P_i and P_k are in W showing $x \in \Lambda$. Thus, we have that $\alpha P_i - X_i \subseteq \Gamma$ for any process P_i in W. Now say $P_i \xrightarrow{\sigma} P_j$ where both P_i and P_j are in W. From Definition 2.1.1 it follows that $P_i \xrightarrow{\sigma}\bullet P_j$ and from above we see that $(\alpha P_i - X_i) \cup (\alpha P_j - X_j) \subseteq \Gamma$. Thus we may strengthen this to $P_i \xrightarrow{\sigma,\Gamma}\bullet P_j$. Restricting to the state δ, we conclude that $P_i \xrightarrow{\delta} P_j$ implies $P_i \xrightarrow{\delta,\Gamma}\bullet P_j$.

Referring to Definition 1.2.7 we note that every process in W is blocked and hence by Lemma 1.2.1 δ is a deadlock state of W.

To complete the proof of the lemma it remains to show that if δ is a deadlock state of W, then W is CM-deadlocked in σ. This follows readily from the fact that if a process is blocked then it is CM-blocked and that $\alpha P_i - X_i \subseteq \Gamma$ for every process P_i in W. □

The type of deadlock freedom and proof method are then defined as follows.

Definition 2.1.3 V is *CM-deadlock free* if no subnetwork of V is CM-deadlocked in any state of V.

Definition 2.1.4 A *priority function* for V is a map $g : \mathcal{S}(V) \times \Lambda \to \mathrm{N}$ such that whenever P_i is CM-blocked in a state σ of V then

$$\{e : \Lambda \mid \forall x : \alpha P_i \cap \Lambda . g(\sigma, e) \leq g(\sigma, x)\} \subseteq \alpha P_i - X_i.$$

So a priority function ensures that every CM-blocked process is willing to do an event of minimum priority in its alphabet.

The next theorem establishes the required connection between priority functions and CM-deadlock free networks.

Theorem 2.1.2 ([CM1]) Let V be a triple-disjoint, busy network. Then V is CM-deadlock free if and only if V has a priority function.

Thus, priority functions provide a complete proof technique for triple-disjoint, busy, CM-deadlock free networks. The technique requires global analysis of a network (as we deal with states of the whole network) and priorities have to be assigned to every event in the vocabulary of the network. However, we now see that adding conflict freedom as a pre-requisite to networks allows us to simplify this latter problem by defining priorities only on the edges of the communication graph of a network rather than on the whole vocabulary.

Definition 2.1.5 Let $V = \langle P_1, \ldots, P_n \rangle$ be a triple-disjoint, busy, conflict free network with a communication graph E. We say a map

$$g : \mathcal{S}(V) \times E \to T \quad (\text{where } (T, \leq) \text{ is a total order})$$

is an *edge priority function* for V if, whenever a process P_i is blocked in a state σ of V then $P_i \xrightarrow{\sigma} P_j$ if

$$\{i, j\} \in \{\{i, r\} : E \mid \forall k | \{i, k\} \in E . g(\sigma, \{i, r\}) \leq g(\sigma, \{i, k\})\}.$$

The corresponding theorem is then as follows.

Theorem 2.1.3 Suppose V is a triple-disjoint, busy, conflict free network. Then V is CM-deadlock free if and only if V has an edge priority function.

Proof. Let Λ be the vocabulary of $V = \langle P_1, \ldots, P_n \rangle$. Firstly, say V has an edge priority function and assume for the sake of contradiction that V is not CM-deadlock free. Then there is a subnetwork U of V and a state σ of V such that, U is CM-deadlocked in σ. Then from Lemma 2.1.1, if δ is the state of U corresponding to σ, δ is a deadlock state of U. Noting that every edge in the communication graph of U is also in the communication graph of V, choose $\{i, j\}$ to be an edge of minimum priority in U. As δ is a deadlock state of U, from Lemma 1.2.1 both P_i and P_j are blocked in δ and thus in σ. Moreover, again from Lemma 1.2.1, both P_i and P_j have requests only to processes in U. Then using the properties of an edge priority function and the choice of $\{i, j\}$ as an edge of minimum priority in U, we see that $P_i \xrightarrow{\sigma} P_j$ and $P_j \xrightarrow{\sigma} P_i$. As both processes are blocked in σ, we may strengthen this to $P_i \xrightarrow{\sigma,\Lambda}\bullet P_j$ and $P_j \xrightarrow{\sigma,\Lambda}\bullet P_i$. The restriction of σ to the subnetwork $\langle P_i, P_j \rangle$ is then a conflict state which contradicts the conflict freedom of V.

To prove the converse we use an algorithm to construct an edge priority function $g : \mathcal{S}(V) \times E \to \{0, 1, \ldots, n\}$, where E is the set of edges in the communication graph of V. The algorithm given is adapted from the one in [CM1] and an outline of its correctness may be deduced from the argument found there. Let σ be a state of V and define

$$u(l, \sigma) \mathrel{\widehat{=}} \{(i, j) \mid i \notin dom(l) \wedge j \in dom(l) \wedge P_i \text{ is blocked in } \sigma \wedge P_i \xrightarrow{\sigma} P_j\}.$$

Then $h_\sigma = g\restriction(\{\sigma\} \times E)$ is computed as follows:

$h_\sigma, l := \emptyset, \{i \mapsto 0 \mid P_i \text{ is not blocked in } \sigma\};$
$C := u(l, \sigma);$
do
  $C \neq \emptyset \to (i, j) := choose(C);$
    $l := l \cup \{i \mapsto l(j) + 1\};$
    $h_\sigma := h_\sigma \cup \{(\sigma, \{i, j\}) \mapsto l(i)\};$
    $C := u(l, \sigma)$
od;
$h_\sigma := h_\sigma \cup \{(\sigma, e) \mapsto n \mid e \in E - dom(g)\}$

where $choose(C)$ is an arbitrarily chosen element from the set C. Thus $g = \bigcup_{\sigma \in \mathcal{S}(V)} h_\sigma$ is an edge priority function for V. □

2.2 Deadlock Freedoms

We begin by defining three further types of deadlock freedom, the first from [BR3].

Definition 2.2.1 A network is *hereditarily deadlock free* if all its subnetworks (including itself) are deadlock free.

Definition 2.2.2 A triple-disjoint network is *acyclic deadlock free* if in addition to being deadlock free no state of the network has a proper cycle of ungranted requests.

Definition 2.2.3 A triple-disjoint, busy network is *strongly acyclic deadlock free* if no state of the network has a cycle of ungranted requests.

The following lemma ensures that the types of deadlock freedom defined in this chapter are consistent with our basic definition of deadlock freedom, which justifies our calling them "types of deadlock freedom". We shall concentrate on triple-disjoint and busy networks in what follows.

Lemma 2.2.1 Let V be a triple-disjoint, busy network. Then if V is either hereditarily deadlock free, acyclic deadlock free, strongly acyclic deadlock free or CM-deadlock free then it is also deadlock free.

Proof. That hereditary and acyclic deadlock freedom imply deadlock freedom follow trivially.

If V is CM-deadlock free but not deadlock free we know it must have a deadlock state. However, from Lemma 2.1.1, we conclude that V is then CM-deadlocked in that state which contradicts its CM-deadlock freedom. Thus CM-deadlock freedom implies deadlock freedom.

Now assume $V = \langle P_1, \ldots, P_n \rangle$ with vocabulary Λ is acyclic deadlock free but not deadlock free. Then V has a deadlock state, say δ. Then we may choose process indices i_r, where $r \in \mathbf{N}$, as follows:

(i) $i_0 = 1$, and

(ii) i_{r+1} is such that $P_{i_r} \xrightarrow{\delta,\Lambda}\bullet\, P_{i_{r+1}}$.

The existence of i_{r+1} is assured by Definition 1.2.7 and Lemma 1.2.1, as every process is blocked in δ. Now as the network is finite there exists a smallest integer k such that $i_{k+1} = i_j$ for some $0 \leq j < k$. But then $\langle i_j, \ldots, i_k \rangle$ is a

cycle of ungranted requests (it is not necessarily a proper cycle of ungranted requests) which contradicts the strong acyclic deadlock freedom of V. So the strong acyclic deadlock freedom of V implies its deadlock freedom. □

We observe that the strong acyclic deadlock freedom of a triple-disjoint, busy network implies its acyclic deadlock freedom.

There now follow some examples which will be used to compare and contrast these types of deadlock freedom.

Example 2.2.1 ([BR3]) This is a network based on the well-known "Dining Philosophers with Butler" in [H1] and defined fully here in Example 4.2.1. It differs from this one by a change in the butler process which ensures that the network is conflict free. The conflict which necessitates this change is described briefly in Example 4.2.1 and a fuller discussion together with details of the amended process may be found in [BR3]. In summary, the network is triple-disjoint, busy, conflict free and deadlock free. □

The communication graphs of the next three examples are displayed in Figures 1 to 3.

Example 2.2.2 Define processes as follows:

$$P_1 = \{1,3\} \rightarrow P_1 \qquad \alpha P_1 = \{\{1,3\},\{1,2\}\}$$

$$P_2 = \Box \begin{matrix} \{1,2\} \rightarrow P_2 \\ \{2,3\} \rightarrow P_2 \end{matrix} \qquad \alpha P_2 = \{\{2,3\},\{1,2\},\{2,4\}\}$$

$$P_3 = \Box \begin{matrix} \{2,3\} \rightarrow P_3 \\ \{3,4\} \rightarrow P_3 \end{matrix} \qquad \alpha P_3 = \{\{2,3\},\{1,3\},\{3,4\}\}$$

$$P_4 = \{2,4\} \rightarrow P_4 \qquad \alpha P_4 = \{\{2,4\},\{3,4\}\}$$

It is easy to establish the network $\langle P_1, P_2, P_3, P_4 \rangle$ is triple-disjoint, busy and conflict free. Moreover, the network is deadlock free as the processes P_2 and P_3 are never blocked. □

Example 2.2.3 Define processes as follows:

$$P_1 = \{1,2\} \rightarrow P_1 \qquad \alpha P_1 = \{\{1,2\},\{1,3\}\}$$

$$P_2 = \{2,3\} \rightarrow P_2 \qquad \alpha P_2 = \{\{2,3\},\{1,2\}\}$$

$$P_3 = \{1,3\} \rightarrow P_3 \qquad \alpha P_3 = \{\{1,3\},\{2,3\}\}$$

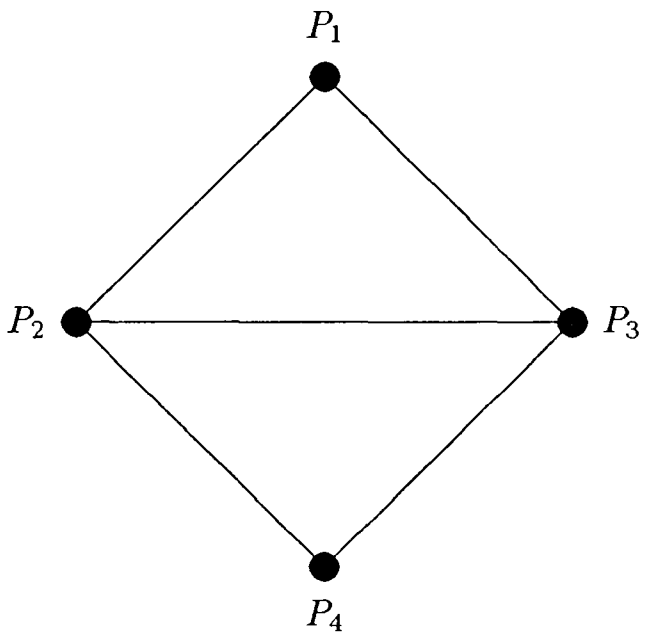

Figure 2.1: The communication graph of Example 2

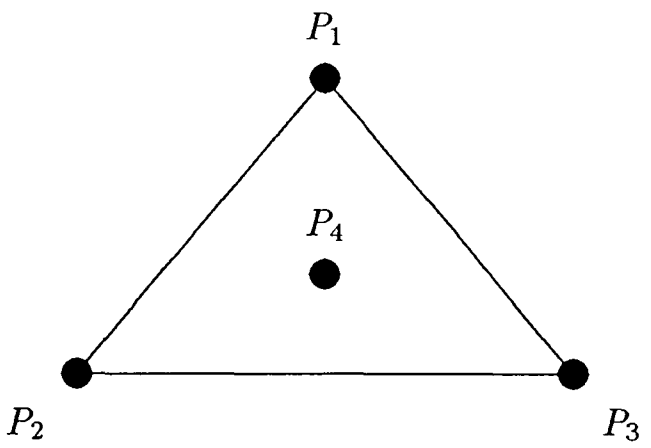

Figure 2.2: The communication graph of Example 3

and let P_4 be any deadlock free process such that $\alpha P_4 \cap \alpha P_j = \emptyset$, for $1 \leq j \leq 3$. It is clear that the network $\langle P_1, P_2, P_3, P_4 \rangle$ is triple-disjoint, busy and conflict free. It is also deadlock free as the process P_4 is never blocked. □

Example 2.2.4 Define processes as follows:

$$P_1 = \langle 1, 2 \rangle \rightarrow P_1 \qquad \alpha P_1 = \{\langle 1, 2 \rangle\}$$

$$P_2 = \langle 2, 1 \rangle \rightarrow P_2 \qquad \alpha P_2 = \{\langle 2, 1 \rangle\}$$

and let P_3 be any deadlock free process such that $\alpha P_3 \cap \alpha P_j = \emptyset$, for $1 \leq j \leq 2$.

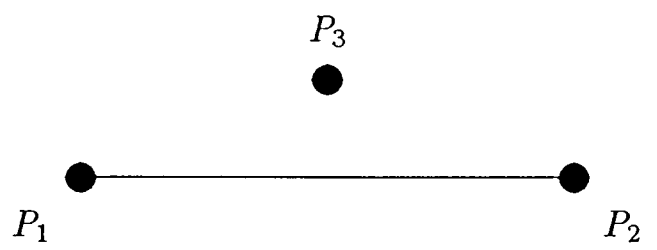

Figure 2.3: The communication graph of Example 4

The network $\langle P_1, P_2, P_3 \rangle$ is triple-disjoint, busy and deadlock free, the latter because the process P_3 is never blocked. □

The network in Example 1 is not hereditarily deadlock free as the removal of the butler yields a network which is not deadlock free. Hence deadlock freedom does not imply hereditary deadlock freedom.

The network in Example 2 has a proper cycle of ungranted requests $\langle 1, 2, 4, 3 \rangle$ in the state corresponding to the trace $\langle \rangle$ of the network so showing that deadlock freedom need not imply the acyclic or strong acyclic deadlock freedom of a triple-disjoint, busy network.

Example 3 is a network that is not CM-deadlock free, as in every state of the network, the subnetwork $\langle P_1, P_2, P_3 \rangle$ is CM-deadlocked in that state. Thus deadlock freedom does not imply CM-deadlock freedom. In fact this example is not acyclic deadlock free either, so neither of these properties are always necessary for the deadlock freedom of a triple-disjoint, busy network.

A positive result relating CM-deadlock freedom and hereditary deadlock freedom is as follows.

Theorem 2.2.2 A triple-disjoint, busy network that is hereditarily deadlock free is also CM-deadlock free.

Proof. Assume there is a triple-disjoint, busy network which is hereditarily deadlock free but not CM-deadlock free. Then the network has a subnetwork which is CM-deadlocked in some state of it. So from Lemma 2.1.1 we see the subnetwork has a deadlock state. Thus the network has a subnetwork which is not deadlock free so contradicting its hereditary deadlock freedom. □

However, a triple-disjoint, busy, hereditarily deadlock free network is not necessarily acyclic or strongly acyclic deadlock free. Consider Example 2. The network is deadlock free as are its proper subnetworks (by applying Lemma 1.3.1, as they are triple-disjoint, busy, conflict free and have no cycles of ungranted requests). So the network is hereditarily deadlock free but, as we saw from above, not acyclic or strongly acyclic deadlock free.

The following result though connects CM-deadlock freedom with strong acyclic deadlock freedom.

Theorem 2.2.3 A triple-disjoint, busy network that is strongly acyclic deadlock free is also CM-deadlock free.

Proof. Assume a triple-disjoint, busy network is strongly acyclic deadlock free but not CM-deadlock free. Then the network has a subnetwork which is CM-deadlocked in some state of it. So from Lemma 2.1.1 we see the subnetwork has a deadlock state. Now this subnetwork is triple-disjoint and busy so we may proceed, as in the proof of Lemma 2.2.1, to construct a cycle of ungranted requests in the deadlock state of the subnetwork. But this yields a cycle of ungranted requests in the corresponding state of the whole network so contradicting its strong acyclic deadlock freedom. □

The acyclic or strong acyclic deadlock freedom of a triple-disjoint, busy network does not entail its hereditary deadlock freedom. Example 1 may be shown to be strongly acyclic deadlock free by methods in Chapter 5, but we noted earlier that it is not hereditarily deadlock free.

The network in Example 4 has a cycle of ungranted requests $\langle 1, 2 \rangle$ in the state corresponding to the trace $\langle\rangle$ of the network, so it is not strongly acyclic deadlock free but as there are never any proper cycles of ungranted

requests it is acyclic deadlock free. Thus strong acyclic deadlock freedom does not in general imply acyclic deadlock freedom. This example also shows that an acyclic deadlock free network is not necessarily CM-deadlock free, because the subnetwork $\langle P_1, P_2 \rangle$ is always CM-deadlocked.

Within the structure set out above, there are no categories of networks which are void. For instance the version of the Dining Philosophers with Butler in Chapter 5 is both acyclic and CM-deadlock free but neither strongly acyclic or hereditarily deadlock free. Examples of acyclic and hereditarily deadlock free networks which are not strongly acyclic deadlock free are abundant in number, indeed many of the networks that we shall meet in the next chapter will be of this form. Finally, simply combining the networks of Examples 1 and 2 into a single network results in a network which is CM-deadlock free but neither hereditarily or acyclically so.

This completes our analysis of the relationships between the types of deadlock freedom defined in this chapter and a visual summary is given in Figure 2.4. Networks which are conflict or strong conflict free in addition to being triple-disjoint and busy naturally have stronger characterisations in terms of the types of deadlock freedom described above. Clearly, a conflict free network which is acyclic deadlock free is always strongly acyclic deadlock free, and it is not hard to see that a strong conflict free network which is acyclic deadlock free is also CM-deadlock free.

A result that identifies the type of deadlock freedom enjoyed by many ring-like networks is the following.

Lemma 2.2.4 A triple-disjoint, busy, strong conflict free network whose communication graph is a cycle is deadlock free if, and only if, it is acyclic and hereditarily deadlock free.

Proof. One direction of the proof is obvious so we only deal with the case where we need to deduce the networks acyclic and hereditary deadlock freedom. Now all proper subnetworks are also strong conflict free and as their communication graphs must be trees, we see by Corollary 1.3.3 that the network is hereditarily deadlock free. If it is not acyclic deadlock free then there is a state with a proper cycle of ungranted requests. But on further analysis this shows that every process is blocked thus contradicting the deadlock freedom of the network. □

2.3 The Variant Theorem

Firstly we classify the networks amenable to the Variant Theorem (Theorem 1.3.5) in the framework provided by the last section.

Lemma 2.3.1 A network that can be proved deadlock free by the Variant Theorem is strongly acyclic and hereditarily deadlock free.

The proof of the part of the lemma relating to hereditary deadlock freedom may be found in [RD] and that for acyclic deadlock freedom is also clear from there. We shall see later on that the converse of the lemma is not true.

The next corollary establishes a relationship between the proof method of variant functions and priority functions.

Corollary 2.3.2 A network that can be proved deadlock free by the Variant Theorem has an edge priority function.

Proof. If a network can be proved deadlock free by the Variant Theorem then we know it is triple-disjoint, busy and conflict free. Also, by the lemma above, the network is both hereditarily and strongly acyclic deadlock free. Then either by Theorem 2.2.2 or Theorem 2.2.3 we deduce that the network is CM-deadlock free. So by Theorem 2.1.3 the network has an edge priority function. □

This proof does not however construct an edge priority function explicitly and thus fails to observe an intimate connection between the variant functions and an edge priority function. The following alternative proof of the corollary remedies this.

Proof'. Say $V = \langle P_1, \ldots, P_n \rangle$ is a triple-disjoint, busy network with vocabulary Λ and variant functions f_i $(1 \leq i \leq n)$. As any partial order may be embedded in a total order, we may assume that the co-domain of the the variant functions is a total order, say $(T, >)$. Then if E is the set of edges in the communication graph of V, define $g : \mathcal{S}(V) \times E \rightharpoonup T$ as follows:

$$g(\sigma, \{i,j\}) = \begin{cases} f_j(\sigma \restriction \alpha P_j) & \text{if } P_i \xrightarrow{\sigma,\Lambda}\bullet\ P_j \\ f_i(\sigma \restriction \alpha P_i) & \text{otherwise} \end{cases}$$

We assert that g is an edge priority function for V. Say a process in the network is blocked. On the edges of E incident on it on which it has an

ungranted request, g assigns the value of of the variant of the process being waited upon. On all other edges incident on it, the value assigned is that of the variant of the process itself. As the process is blocked, from the properties of variant functions we see that the latter value is greater than the former values and thus the edge incident on the process with the minimum value of g is certainly being waited upon by the process. So we conclude that g is an edge priority function. □

The Variant Theorem is strong enough to prove the deadlock freedom of many systems as can be seen in [RD]. However there are many networks to which it cannot be applied, for instance networks that are not hereditarily deadlock free or strongly acyclic deadlock free. The following theorem identifies precisely those networks to which the Variant Theorem can be applied.

Theorem 2.3.3 Suppose $V = \langle P_1, \ldots, P_n \rangle$ is a triple-disjoint, busy network with vocabulary Λ. Let

$$\Pi = \bigcup_{i=1}^{n} (\tilde{\mathcal{F}}[\![P_i]\!] \times \{i\})$$

and define the relation $\succ$ on Π as follows: $(\sigma_i, i) \succ (\sigma_j, j)$ holds whenever there exists a state σ of the subnetwork $\langle P_i, P_j \rangle$ such that $\sigma\restriction\alpha P_i = \sigma_i$, $\sigma\restriction\alpha P_j = \sigma_j$ and $P_i \xrightarrow{\sigma,\Lambda}\bullet P_j$. Then V can be proved deadlock free by the Variant Theorem if and only if the transitive closure of $\succ$ on Π is a partial order.

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

$$(\sigma_0, i_0) \succ (\sigma_1, i_1) \succ \cdots \succ (\sigma_m, i_m)$$

where $\sigma_0 = \sigma_m$ and $i_0 = i_m$. Now, if the Variant Theorem can be used on V, by the properties of the variant functions and the definition of $\succ$, we would have

$$f_{i_0}(\sigma_0) > f_{i_1}(\sigma_1) > \cdots > f_{i_m}(\sigma_m)$$

and hence that $f_{i_0}(\sigma_0) > f_{i_0}(\sigma_0)$, which is a contradiction. Thus, if $(\Pi, \rhd)$ is not a partial order, the Variant Theorem cannot be applied to V.

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

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

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

So we conclude that V may be proved deadlock free by the Variant Theorem exactly when $(\Pi, \rhd)$ is a partial order. □

Thus when the Variant Theorem fails, from the theorem above, there exists a "local proper cycle of ungranted requests". However, this does not necessarily imply the existence of a global state in which there is a proper cycle of ungranted requests (although the converse is true). An example of such a network is the message passing ring in [R4] where each element is a restricted double buffer (see Example 4.3.2). This network has no global state in which there is a proper cycle of ungranted requests but does have local proper cycles that preclude the use of the Variant Theorem. The network is also strongly acyclic deadlock free and hereditarily deadlock free so showing that the converse to Lemma 2.3.1 is not true.

We now present two lemmas which introduce classes of networks that can be proved deadlock free by the Variant Theorem. These classes of networks are quite general and encompass networks that are often met in practice. Instead of producing explicit variant functions to prove their deadlock freedom, we shall use the theorem above instead.

Lemma 2.3.4 Let $V = \langle P_1, \ldots, P_n \rangle$ be a busy, triple-disjoint, conflict free network. Say there is a partial order $<$ on elements of V such that all neighbours are comparable and if $P_i \xrightarrow{\sigma} P_j$, with $P_j < P_i$, then $P_i \xrightarrow{\sigma} P_r$ where P_r is any neighbour of P_i with $P_r < P_i$. Then the network is deadlock free.

Proof. Say V has vocabulary Λ and assume for the sake of contradiction that the transitive closure of $\succ$ on Π (as defined in Theorem 2.3.3) is not a partial order. This can only be if there is a "local proper cycle of ungranted requests" say

$$(\sigma_0, i_0) \succ (\sigma_1, i_1) \succ \cdots \succ (\sigma_k, i_k) \succ (\sigma_0, i_0).$$

Choose P_{i_m} to be maximal with respect to $<$ in the cycle above. Then

$$P_{i_{m-1}} \xrightarrow{\sigma,\Lambda}\bullet\, P_{i_m} \xrightarrow{\sigma',\Lambda}\bullet\, P_{i_{m+1}} \quad \text{(arithmetic modulo } k+1\text{)}$$

where σ, σ' are states of the subnetworks $\langle P_{i_{m-1}}, P_{i_m} \rangle$, $\langle P_{i_m}, P_{i_{m+1}} \rangle$ respectively such that

- $\sigma\restriction\alpha P_{i_{m-1}} = \sigma_{m-1}$,
- $\sigma\restriction\alpha P_{i_m} = \sigma'\restriction\alpha P_{i_m} = \sigma_m$
- and $\sigma'\restriction\alpha P_{i_{m+1}} = \sigma_{m+1}$.

Now $P_{i_m} \xrightarrow{\sigma'} P_{i_{m+1}}$ and by the choice of P_{i_m} we have that $P_{i_{m+1}} < P_{i_m}$. So by the conditions on the network we deduce that $P_{i_m} \xrightarrow{\sigma'} P_{i_{m-1}}$ and as $\sigma\restriction\alpha P_{i_m} = \sigma'\restriction\alpha P_{i_m}$ this yields that $P_{i_m} \xrightarrow{\sigma} P_{i_{m-1}}$. Also as $P_{i_{m-1}} \xrightarrow{\sigma,\Lambda}\bullet P_{i_m}$, we may strengthen this to $P_{i_m} \xrightarrow{\sigma,\Lambda}\bullet P_{i_{m-1}}$. But this contradicts the conflict freedom of the network and so we conclude that the transitive closure of $\succ$ on Π is a partial order. Thus, using the theorem above, we see that the network is deadlock free and may be proved so by the Variant Theorem. □

Before proceeding to the second lemma we give a specific example of the type of network described above.

Example 2.3.1 ([Y]) The network implements a message passing system on a finite cartesian grid where a message can be input by the environment at a node (i,j) for transfer by the network to some specified node (k,l) where it will be output to the environment. For the purposes of this implementation we consider the grid as actually being two overlaid vertical grids (a "foreground" and a "background" one) of processors where each node (i,j) in one is connected to the corresponding node (i,j) in the other by a channel *over*. The "foreground" grid elements F_{ij} will input messages from the environment (on channels *in*) while the "background" elements B_{ij} will output them to the environment (on channels *out*). To get a message from (i,j) to (k,l), it is moved to $(\min\{i,k\},\min\{j,l\})$ in the "foreground" grid then over to the corresponding element in the "background" grid from where it continues its journey to its destination within the "background" grid. A typical "foreground" element is defined by the process

$$F = \begin{array}{l} \square \\ \square \\ \square \end{array} \begin{array}{l} in?m \rightarrow F(m) \\ up?m \rightarrow F(m) \\ right?m \rightarrow F(m) \end{array}$$

where

$$F(m) = \begin{array}{l} \\ \textbf{or} \\ \textbf{or} \\ \\ \end{array} \begin{array}{l} over!m \rightarrow F \\ down!m \rightarrow F \\ left!m \rightarrow F \end{array}$$

while the process representing a typical "background" element is

$$B = \begin{array}{l} \Box \\ \Box \end{array} \begin{array}{l} over?m \rightarrow B(m) \\ down?m \rightarrow B(m) \\ left?m \rightarrow B(m) \end{array}$$

where

$$B(m) = \begin{array}{l} \\ \textbf{or} \\ \textbf{or} \end{array} \begin{array}{l} out!m \rightarrow B \\ up!m \rightarrow B \\ right!m \rightarrow B \end{array}$$

The choices in $F(m)$ and $B(m)$ are dependent on the message header in m which has been constructed and is interpreted in accordance with the algorithm outlined above. For deadlock freedom proof purposes however, the argument below still holds even if these choices are non-deterministic.

The network is clearly triple-disjoint and busy. That it is conflict free follows easily from the fact that any two neighbouring processes are connected via only one channel.

Now define a partial order $<$ on all the elements as follows: every "foreground" element is less than every "background" element and furthermore

$$F_{ij} \leq F_{i'j'} \Leftrightarrow i' \leq i \wedge j' \leq j$$
$$B_{ij} \leq B_{i'j'} \Leftrightarrow i \leq i' \wedge j \leq j'$$

Then applying Lemma 2.3.4 with the above ordering to the network proves its deadlock freedom. □

We now state the second lemma which generalises a result in [DS].

Lemma 2.3.5 ([RD]) Suppose $V = \langle P_1, \ldots, P_n \rangle$ is a triple-disjoint, busy and 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.

The proof is similar to that of the previous lemma and an alternative proof is given in [RD]. An example of a network that falls into the class of networks described above is as follows.

Example 2.3.2 The network inputs pairs of integers $\langle a, b\rangle$ on a channel in and outputs $(a+b)^n$ on the channels out_n. This is implemented by processes P_{nr} $(0 \leq r \leq n \leq k)$, with k being a fixed constant, where each process P_{nr} is positioned in the communication graph as in Pascal's triangle, that is the rth element on the nth row. The channel in is connected to the process P_{00} and a channel out_n is connected to the process P_{nn}. Each process P_{nr} is connected to those processes $P_{(n-1)(r-1)}$, $P_{(n-1)r}$, $P_{n(r-1)}$, $P_{(n+1)r}$, $P_{(n+1)(r+1)}$ and $P_{n(r+1)}$ in which the process indexing is valid. The processes are defined as follows:

$$P_{00} = \left[\begin{array}{l} (\ in?\langle a, b\rangle \rightarrow SKIP\ )\,; \\ \\ \left(\begin{array}{ll} \| & out_0!1 \rightarrow SKIP \\ \| & \{00, 10\}!\langle a, 1\rangle \rightarrow SKIP \\ \| & \{00, 11\}!\langle b, 1\rangle \rightarrow SKIP \end{array} \right)\,; \\ \\ P_{00} \end{array} \right]$$

for $n > 0$ we put

$$P_{n0} = \left[\begin{array}{l} (\ \{n0, (n-1)0\}?\langle x, y\rangle \rightarrow SKIP\ )\,; \\ \\ \left(\begin{array}{ll} \| & \{n0, n1\}!\langle x, xy\rangle \rightarrow SKIP \\ \| & \{n0, (n+1)0\}!\langle x, xy\rangle \rightarrow SKIP \\ \| & \{n0, (n+1)1\}!1 \rightarrow SKIP \end{array} \right)\,; \\ \\ P_{n0} \end{array} \right]$$

and

$$P_{nn} = \left[\begin{array}{l} \left(\begin{array}{ll} \| & \{nn, (n-1)(n-1)\}?\langle u, v\rangle \rightarrow SKIP \\ \| & \{nn, n(n-1)\}?\langle x, y\rangle \rightarrow SKIP \end{array} \right)\,; \\ \\ \left(\begin{array}{ll} \| & out_n!(y + uv) \rightarrow SKIP \\ \| & \{nn, (n+1)(n+1)\}!\langle u, uv\rangle \rightarrow SKIP \\ \| & \{nn, (n+1)n\}!\langle uv, 1\rangle \rightarrow SKIP \end{array} \right)\,; \\ \\ P_{nn} \end{array} \right]$$

while for $0 < r < n$ we have

$$P_{nr} = \left[\begin{array}{l} \left(\begin{array}{ll} \| & \{nr, (n-1)(r-1)\}?c \rightarrow SKIP \\ \| & \{nr, (n-1)r\}?\langle w, d\rangle \rightarrow SKIP \\ \| & \{nr, n(r-1)\}?\langle x, y\rangle \rightarrow SKIP \end{array} \right); \\ \\ \left(\begin{array}{ll} \| & \{nr, n(r+1)\}!(y + [c+d]xw) \rightarrow SKIP \\ \| & \{nr, (n+1)(r+1)\}!(c+d) \rightarrow SKIP \\ \| & \{nr, (n+1)r\}!\langle xw, c+d\rangle \rightarrow SKIP \end{array} \right); \\ \\ P_{nr} \end{array} \right]$$

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

It is a trivial matter to see that the network is triple-disjoint and busy, while its conflict freedom follows from the fact that any two neighbouring processes are connected by just one channel.

Now define a partial order $<$ on the network elements in the following way:

$$P_{nr} \leq P_{n'r'} \Leftrightarrow n \leq n' \wedge r \leq r' \quad .$$

It is then easy to verify that with this ordering, the network satisfies the prerequisites of the lemma above and is thus deadlock free. □

To complete the chapter, we present Figure 2.4 which summarises the relationships between the various types of deadlock freedom and proof techniques discussed in this chapter.

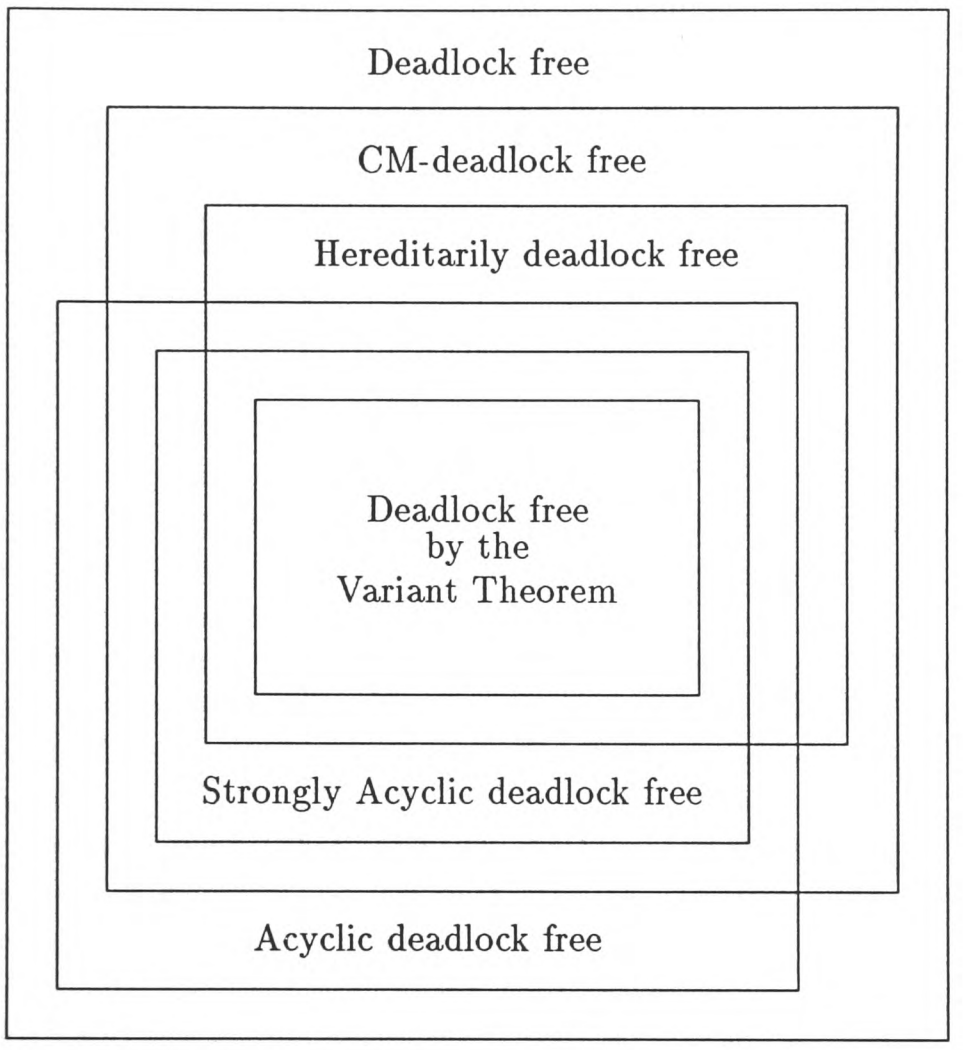

Figure 2.4: Deadlock freedoms for triple-disjoint, busy networks

Chapter 3

Extensions of the Variant Theorem

The ease of use of the Variant Theorem is a direct consequence of the need for only local checking in meeting its preconditions. In Section 2.3 its limitations were discussed and in this section we consider generalisations of it which overcome many of these, while still retaining the all-important consideration of local analysis.

In the first section we introduce a method for detecting potential deadlock states in a network. Clearly, if there are no such states it provides us with a proof of deadlock freedom, and if not, it is often a useful first step towards one.

In each of the next three sections, deadlock freedom proof rules that are extensions of the Variant Theorem are put forward and their use is illustrated with the help of examples.

In the penultimate section, these proof rules are compared and contrasted, with the aim of providing an insight into when the application of a particular rule is appropriate.

We conclude the chapter with a section on the treatment of deadlock freedom of networks which contain processes that can terminate. We also show how the rules we have developed earlier in this chapter can be easily adapted to cope with this change.

3.1 Detecting Deadlock States

Theorem 3.1.1 Let $V = \langle P_1, \ldots, P_n \rangle$ be a triple-disjoint, busy and strong

conflict free network with vocabulary Λ. Furthermore, suppose there exist functions

$$f_i : \tilde{\mathcal{F}}[\![P_i]\!] \longrightarrow \Pi \qquad (1 \leq i \leq n)$$

where $(\Pi, >)$ is a partial order, such that whenever σ is a state of the subnetwork $\langle P_i, P_j \rangle$, and $\{i, j\}$ is a non-disconnecting edge in the communication graph of V, then

$$P_i \xrightarrow{\sigma,\Lambda}\bullet\, P_j \quad \Rightarrow \quad f_i(\sigma{\restriction}\alpha P_i) \geq f_j(\sigma{\restriction}\alpha P_j)\,.$$

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

Proof. Suppose V satisfies the prerequisites of the theorem and δ is a deadlock state of V. By Theorem 1.3.1, δ has a proper cycle of ungranted requests, say $\langle i_0, \ldots, i_{m-1} \rangle$. Now define $g : \{0, \ldots, m-1\} \to \Pi$ by setting $g(j) = f_{i_j}(\delta{\restriction}\alpha P_{i_j})$. 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 $g(j) = g(0)$ for all j, so completing the proof that all the $f_{i_j}(\delta{\restriction}\alpha P_{i_j})$ are equal. □

Observe that when we define all the variant functions to be the same constant, the theorem reduces to Theorem 1.3.1.

The theorem above provides a useful tool for analysing "difficult" networks for potential deadlocks by placing bounds on the places deadlock might appear. The search for deadlocks is then restricted to cycles with equal variant. In practice, the theorem isolates states of individual processes in the network which may correspond to a deadlock state. To detect a deadlock state, we have to ensure that there does exist a global state corresponding to those individual states and that every process is in fact blocked in it. Clearly, the absence of any such global states implies the deadlock freedom of the network. Thus a certain amount of global analysis of a network is entailed in the use of the theorem, but this may be minimised by producing a system of variants which is as "refined" as possible (i.e. yields as many strict inequalities as possible).

Example 3.1.1 Define processes

$$P_i = C_i(black) \sqcap C_i(white) \qquad (0 \leq i < 2n)$$

where

$$C_i(x) = \Box \begin{array}{l} \{i, i+1\}.x \rightarrow P_i \\ \{i, i-1\}.x \rightarrow P_i \end{array}$$

and

$$\alpha P_i = \{\{i, i+1\}.black, \{i, i-1\}.black, \{i, i+1\}.white, \{i, i-1\}.white\}$$

with all arithmetic being modulo $2n$.

Consider the network $\langle P_0, \ldots, P_{2n-1} \rangle$, $n \geq 2$. The network is connected as a ring. Each process non-deterministically enters one of two "states" ("black" or "white") either initially or just after it has communicated with a neighbour. A process is then prepared to communicate with any neighbour which shares the same "state" as itself. It is trivial to establish that the network is triple-disjoint and busy. It is not conflict free, as a state of two adjacent processes in which one process is "black" and the other is "white" is a conflict state, however it is strong conflict free as there can be no strong requests.

Now define functions

$$f_i : \tilde{\mathcal{F}}[\![P_i]\!] \rightarrow \{0, 1\} \qquad (0 \leq i < 2n)$$

such that

$$f_{2r}((s, X)) = \begin{cases} 0 & \text{if } \{2r, 2r+1\}.black \notin X \\ 1 & \text{otherwise} \end{cases} \qquad (0 \leq r < n)$$

$$f_{2r-1}((s, X)) = \begin{cases} 0 & \text{if } \{2r-1, 2r\}.white \notin X \\ 1 & \text{otherwise} \end{cases} \qquad (1 \leq r \leq n)$$

where $\{0, 1\}$ has its usual order $>$.

The functions clearly satisfy the prerequisites of theorem and thus if δ is a deadlock state of the network it has to satisfy either $f_i(\delta \restriction \alpha P_i) = 0$ for all i, or $f_i(\delta \restriction \alpha P_i) = 1$ for all i.

Now define

$$X_{2r} = B_{2r}, Y_{2r} = W_{2r} \qquad (0 \leq r < n)$$

and

$$X_{2r-1} = B_{2r-1}, Y_{2r-1} = W_{2r-1} \qquad (0 \leq r \leq n)$$

where

$$\begin{aligned} W_i &= \{\{i, i+1\}.black, \{i, i-1\}.black\} \\ B_i &= \{\{i, i+1\}.white, \{i, i-1\}.white\} \end{aligned} \qquad (0 \leq i < 2n) \quad .$$

Then if s_i is a trace of P_i, we see that $f_i((s_i, X_i)) = 0$ while $f_i((s_i, Y_i)) = 1$, for any i. Global states of the network that correspond to either set of individual states specified above are plentiful and all these are readily seen to be deadlock states, such as $\langle\langle\rangle, \langle X_0, \ldots, X_{2n-1}\rangle\rangle$ or $\langle\langle\rangle, \langle Y_0, \ldots, Y_{2n-1}\rangle\rangle$.

We now explore two small variations on this network in relation to the theorem above.

The first variation is the network $\langle P_0, \ldots, P_{2n}\rangle$, $n \geq 1$, where the processes P_i are defined as above but arithmetic is modulo $2n+1$. Although this network is very similar to the original, the variant functions above do not satisfy the prerequisites of the theorem. In fact, every set of variant functions f_i for this network must satisfy the condition $f_i(\sigma\restriction\alpha P_i) = f_j(\sigma\restriction\alpha P_j)$ for any state σ of the subnetwork $\langle P_i, P_j\rangle$. So in this case, variant functions do not provide enough "resolution" to usefully distinguish possible deadlock states, as at least every reachable state of the network is isolated as a possible deadlock state by the theorem. In fact the network is deadlock free but, as it is not conflict free, we cannot apply the Variant Theorem to it. We shall return to this example in the next chapter when we will have enough machinery to prove its deadlock freedom.

Our second variation is obtained by altering the definitions of two processes in the original network. Let $P_{2j} = C_{2j}(black)$ and $P_{2k} = C_{2k}(white)$, where $0 \leq j, k < n$ and $j \neq k$. The same variant functions as for the original network still suffice to satisfy the prerequisites of the theorem. This network however is deadlock free and can be shown so with help from the theorem. We recall from above that for a deadlock state δ of the network, either $f_i(\delta\restriction\alpha P_i) = 0$ for all i, or $f_i(\delta\restriction\alpha P_i) = 1$ for all i. But $f_{2j}(\delta\restriction\alpha P_{2j}) = 0$ and $f_{2k}(\delta\restriction\alpha P_{2k}) = 1$, so neither condition can be met by a deadlock state. Again, because this network is not conflict free, we could not have used the Variant Theorem to prove its deadlock freedom. □

Some more examples of the use of the theorem (including an alternative proof for Lemma 2.3.5) may be found in [RD].

3.2 Selected Edge Rule

Theorem 3.2.1 (Selected Edge Rule) Let $V = \langle P_1, \ldots, P_n\rangle$ be a triple-

disjoint, busy and strong conflict free network with vocabulary Λ. Let E be a set of directed edges from the communication graph of V such that every directed cycle in the communication graph contains at least one edge from E. Suppose there exist functions

$$f_i : \tilde{\mathcal{F}}[\![P_i]\!] \longrightarrow \Pi \qquad (1 \leq i \leq n)$$

where $(\Pi, >)$ is a partial order, such that whenever σ is a state of the subnetwork $\langle P_i, P_j \rangle$ and $\{i, j\}$ is a non-disconnecting edge in the communication graph of V, we have

$$P_i \xrightarrow{\sigma,\Lambda}\bullet P_j \;\Rightarrow\; \begin{cases} f_i(\sigma\restriction\alpha P_i) > f_j(\sigma\restriction\alpha P_j) & \text{if } \langle i,j\rangle \in E \\ f_i(\sigma\restriction\alpha P_j) \geq f_j(\sigma\restriction\alpha P_j) & \text{otherwise} \end{cases}$$

Then V is acyclic and hereditarily deadlock free.

Proof. Easily follows from Theorem 3.1.1. □

Lemma 3.2.2 Any network that can be proved deadlock free by the Variant Theorem may also be proved so by the Selected Edge Rule.

Proof. Just use the same variant functions as for the Variant Theorem and let the set of selected edges be the set of all directed edges in the communication graph of the network. □

There now follow some examples of the use of this Rule.

Example 3.2.1 (The n Dining Philosophers [H1, RD]) 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). This system is not deadlock free and the addition of a $BUTLER$ (see [H1]) is one way of making it so. A proof of this is given in Example 4.2.1. Here we present another way of preventing deadlock (from [RD]) by ensuring that the system contains at least one left-handed philosopher (that is a $PHIL_i$ who will always seek to pick up $FORK_i$ before $FORK_{i-1}$) and one right-handed one. The rest may be ambidextrous, that is able to choose non-deterministically, 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.

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

The left-handed philosopher $PHIL_l$ is thus defined

$$PHIL_l = LEFTVISIT_l; PHIL_l$$

and the right-handed philosopher $PHIL_r$ is

$$PHIL_r = RIGHTVISIT_r; PHIL_r \quad .$$

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

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

Fork processes are described by

$$FORK_i = \Box \begin{array}{l} (i.picksup.i \rightarrow i.putsdown.i \rightarrow FORK_i) \\ (i+1.picksup.i \rightarrow i+1.putsdown.i \rightarrow FORK_i) \end{array} .$$

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

The network is triple-disjoint and busy, while Lemma 1.3.2 (ii) shows it is also conflict free as the communications between each pair of processes follows a strict cyclic pattern.

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 monotonicity is required. (Clearly every cycle includes one of these.) The variant functions are, as we will see, extremely simple.

For the variant function $f_i : \tilde{\mathcal{F}}[\![FORK_i]\!] \rightarrow \{0,1\}$ we put

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

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 $g_i : \tilde{\mathcal{F}}[\![PHIL_i]\!] \to \{0,1\}$ is defined by

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

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 monotonicity is clear in this case. Indeed, when $PHIL_l$ or $PHIL_r$ is waiting for one of these forks the monotonicity 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 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 action $l.eats$ (which is outside the vocabulary of the network), 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 the Selected Edge Rule 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 the Variant Theorem, but will inevitably find that this proof is rather harder than the above. □

The next example defines a network which is of technical interest to us and we shall return to it in a later section.

Example 3.2.2 Define processes

$$P_{00} = MID(left) \qquad \alpha P_{00} = \{mid, left * \top, left * \bot\}$$

$$P_{10} = MID(right) \quad \alpha P_{10} = \{mid, right * \top, right * \bot\}$$

$$P_{01} = TOP(left) \qquad \alpha P_{01} = \{left * mid * h, left * mid * t, left * \top\}$$

$$P_{12} = TOP(right) \quad \alpha P_{12} = \{right * mid * h, right * mid * t, right * \top\}$$

$$P_{02} = BOT(left) \qquad \alpha P_{02} = \{left * mid * h, left * mid * t, left * \bot\}$$

$$P_{11} = BOT(right) \quad \alpha P_{11} = \{right * mid * h, right * mid * t, right * \bot\}$$

where

$$MID(p) = \sqcap \begin{array}{l}(\mu x.\, mid \rightarrow x) \\ (\mu x.\, p * \top \rightarrow x \,\Box\, p * \bot \rightarrow x)\end{array}$$

$$TOP(p) = \Box \begin{array}{l}(\mu x.\, p * mid * h \rightarrow x) \\ p * mid * t \rightarrow (\mu x.\, p * \top \rightarrow x)\end{array}$$

$$BOT(p) = \Box \begin{array}{l}p * mid * h \rightarrow (\mu x.\, p * \bot \rightarrow x) \\ (\mu x.\, p * mid * t \rightarrow x)\end{array}$$

and consider the network $\langle P_{00}, P_{10}, P_{01}, P_{12}, P_{02}, P_{11}\rangle$. The network is triple-disjoint and busy; its communication graph is displayed in Figure 1. The network is also conflict free, this is established by Lemma 1.3.2 except for the connected pairs of "top" and "bottom" processes which require a little bit of separate analysis. The two "middle" processes initially decide non-deterministically in which direction to communicate, either to the "left" or to the "right". Having made this decision, they are then always prepared to accept any communication from the directions they respectively chose. The two connected "top" and "bottom" processes are initially only willing to communicate both the events common in their alphabets. Subsequently, depending on which event was agreed upon and communicated, either the "bottom" process is always willing to communicate only with the "top" one and the "top" one only to the "middle", or the "top" process to the "bottom" and the "bottom" to the "middle".

Let $\mathcal{W}$ be a predicate such that

$$\mathcal{W}(\sigma, ij) = \begin{cases} \mathbf{T} & \text{if } P_{ij} \xrightarrow{\sigma} P_{mn} \text{ with the edge } \langle ij, mn\rangle \text{ in } w_i \\ \mathbf{F} & \text{otherwise} \end{cases}$$

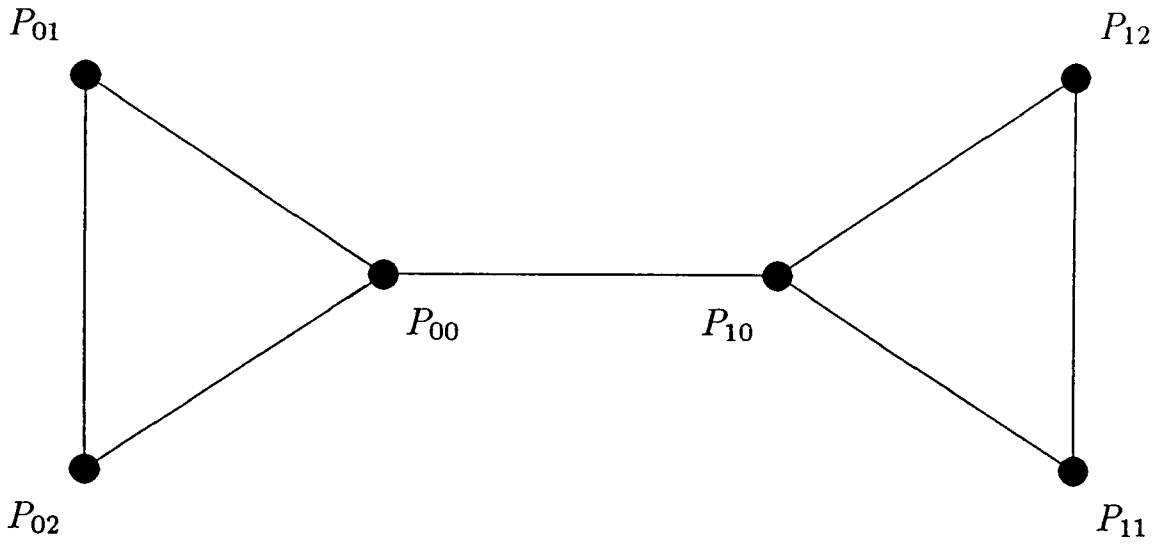

Figure 3.1: The communication graph of Example 1

where w_0 and w_1 are directed walks in the communication graph of the network represented by the sequences $\langle 02, 01, 00, 10\rangle$ and $\langle 12, 11, 10, 00\rangle$ respectively. Now define functions

$$f_{ij} : \tilde{\mathcal{F}}[\![P_{ij}]\!] \to \mathrm{N} \qquad (i \in \{0,1\},\, j \in \{0,1,2\})$$

by putting

$$f_{ij}(\sigma) = \begin{cases} j & \text{if } \mathcal{W}(\sigma, ij) \\ 3-j & \text{otherwise} \end{cases}$$

where N has its usual order $>$. Also define a set of selected directed edges $E = \{\langle 01, 02\rangle, \langle 02, 01\rangle, \langle 11, 12\rangle, \langle 12, 11\rangle\}$. With these definitions, the network is readily seen to be amenable to the Selected Edge Rule.

In fact, we could have chosen E to have been the set of all the directed non-disconnecting edges in the communication graph. We note that since $\{00, 10\}$ is a disconnecting edge, there was never any need to check the values of variant functions with respect to ungranted requests along this edge. In general, the Selected Edge Rule allows us to develop variant functions for subnetworks corresponding to essential components in the communication graph separately.

The Variant Theorem could not have been used on this network as there exist cycles such as

$$(\sigma_{00}, 00) \succ (\sigma_{01}, 01) \succ (\sigma_{02}, 02) \succ (\sigma'_{00}, 00) \succ (\sigma_{10}, 10) \succ$$

$$(\sigma_{11}, 11) \succ (\sigma_{12}, 12) \succ (\sigma'_{10}, 10) \succ (\sigma_{00}, 00)$$

which are precluded by Theorem 2.3.3. Of course, we could have applied the Variant Theorem separately to each of the subnetworks corresponding to the essential components (the functions defined above suffice for this) and then concluded the deadlock freedom of the whole network by Lemma 1.3.4, as the network is conflict free. □

The Selected Edge Rule has many advantages over the Variant Theorem. The one concerning disconnecting edges has already been mentioned above in Example 3.2.2. Also, even though a network may be treated by the Variant Theorem it is still often easier to use the Selected Edge Rule as the possibility of having non-strict inequalities leads to much simpler variant functions as in Example 3.2.1. Finally, it does not disallow the presence of strong conflict free but not conflict free subnetworks. For examples of its use in the latter case and also other examples of its use, we refer the reader to [RD].

3.3 Directed Cycle Rule

Theorem 3.3.1 (Directed Cycle Rule) Let $V = \langle P_1, \ldots, P_n \rangle$ be a triple-disjoint, busy and strong conflict free network with vocabulary Λ. Let D be the set of directed cycles in the communication graph of V. Suppose there exist functions

$$f_i : \tilde{\mathcal{F}}[\![P_i]\!] \times D \longrightarrow \Pi \qquad (1 \leq i \leq n)$$

where $(\Pi, >)$ is a partial order, such that whenever σ is a state of the subnetwork $\langle P_i, P_j \rangle$ and $\langle i, j \rangle$ is an edge in a directed cycle $d \in D$, we have

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

Then V is acyclic and hereditarily deadlock free.

Proof. Follows easily from Theorem 1.3.1. □

Lemma 3.3.2 Any network that can be proved deadlock free by the Variant Theorem may also be proved so by the Directed Cycle Rule.

Proof. If a network has functions h_i satisfying the Variant Theorem, merely define functions f_i such that $f_i(\sigma, d) = h_i(\sigma)$ for use in the Directed Cycle Rule. □

A couple of examples showing the application of the Rule are now appropriate.

Example 3.3.1 Define a process

$$P_0 = \begin{array}{ll} \Box & 0.to.1.white \rightarrow P_0 \\ \Box & 1.to.0.black \rightarrow P_0 \\ \Box & 0.to.(n-1).black \rightarrow P_0 \\ \Box & (n-1).to.0.white \rightarrow P_0 \end{array}$$

$$\alpha P_0 = \{0.to.1.white, 1.to.0.black, 0.to.n-1.black, n-1.to.0.white\}$$

and processes

$$P_i = C_i(black) \sqcap C_i(white) \qquad (0 < i < n)$$

$$\begin{array}{rl} \alpha P_i = & i.to.(i-1).\{black, white\} \cup (i-1).to.i.\{black, white\} \\ & \cup\ i.to.(i+1).\{black, white\} \cup (i+1).to.i.\{black, white\} \end{array}$$

where

$$C_i(x) = \begin{array}{ll} \Box & i.to.(i-1).x \rightarrow P_i \\ \Box & (i-1).to.i.\overline{x} \rightarrow P_i \\ \Box & i.to.(i+1).x \rightarrow P_i \\ \Box & (i+1).to.i.\overline{x} \rightarrow P_i \end{array}$$

with $\overline{black} = white$, $\overline{white} = black$ and all arithmetic being modulo n.

Consider the network $\langle P_0, \ldots, P_{n-1} \rangle$, $n \geq 3$. The network is connected as a ring. Each process P_i, $0 < i < n$, non-deterministically enters one of two "states" ("black" or "white") either initially or just after it has communicated with a neighbour. The process is then prepared to communicate with any neighbour which does not share the same "state" as itself. As for the process P_0, it is prepared to communicate with P_1 if this is "black" and with P_{n-1} if this is "white". The network is clearly triple-disjoint and busy, but it is not conflict free. A state of two adjacent processes (neither process being P_0) in which both processes share the same "state" is a conflict state.

Strong conflict freedom is however a property of the network which follows from the fact that there can be no strong requests.

Let $\hookrightarrow$ be the directed cycle in the communication graph of the network which is represented by $\langle 0, 1, \ldots, n-1 \rangle$ and $\hookleftarrow$ be the directed cycle in the reverse orientation. Now define functions

$$f_i : \tilde{\mathcal{F}}[\![P_i]\!] \times \{\hookrightarrow, \hookleftarrow\} \to \mathbf{Z} \qquad (0 \leq i < n)$$

such that

$$f_0(\sigma, \hookrightarrow) = f_0(\sigma, \hookleftarrow)$$

$$f_i((s, X), \hookrightarrow) = \begin{cases} n-i & \text{if } i.to.(i+1).black \notin X \\ -i & \text{otherwise} \end{cases} \qquad (0 < i < n)$$

$$f_i(\sigma, \hookleftarrow) = -f_i(\sigma, \hookrightarrow) \qquad (0 \leq < n)$$

where $\mathbf{Z}$ has its usual order $>$.

It is a straightforward matter to verify that these functions satisfy the required conditions for the Directed Cycle Rule and thus we conclude that the network is hereditarily deadlock free. We note that this network could not have been treated by the Variant Theorem due to its lack of conflict freedom. □

Example 3.3.2 Let n be a constant with $n > 2$ and let m_j be constants such that $m_j \geq 1$ for $1 \leq j < n$. Define processes

$$P_{00} = \mathop{\Box}_{1 \leq j < n} (\{0, 1\}.black.on.j \to P_{00})$$

$$\alpha P_{00} = \{0, 1\}.black.on.\{1, \ldots, n-1\}$$

$$P_{nn} = \mathop{\Box}_{1 \leq j < n} (\{2m_j, 2m_j - 1\}.white.on.j \to P_{nn})$$

$$\alpha P_{nn} = \bigcup_{j=1}^{n-1} \{2m_j, 2m_j - 1\}.white.on.\{1, \ldots, n-1\}$$

and

$$P_{ij} = C_{ij}(black) \sqcap C_{ij}(white) \qquad (0 < i < 2m_j \text{ and } 1 \leq j < n)$$

$$\alpha P_{ij} = \{i, i-1\}.\{black, white\}.on.j \cup \{i, i+1\}.\{black, white\}.on.j$$

where

$$C_{ij}(x) = \Box \begin{array}{l} \{i, i-1\}.x.on.j \rightarrow P_{ij} \\ \{i, i+1\}.x.on.j \rightarrow P_{ij} \end{array}$$

Consider the network

$$\langle P_{00}, P_{11}, \ldots, P_{(2m_1-1)1}, \ldots, P_{(2m_{n-1}-1)(n-1)}, P_{nn} \rangle$$

whose communication graph is shown in Figure 2. The network is in fact a generalisation of the second variation given in Example 3.1.1, thus the following description should be familiar. Each process is in one of two "states" ("black" or "white"). The processes P_{ij}, with $0 < i < 2m_j$ and $1 \leq j < n$, non-deterministically enter one of the two "states" either initially or after communication with a neighbour. The processes P_{00} and P_{nn} are always "black" and "white" respectively. Two adjacent processes are prepared to communicate only if they share the same "state". The network is triple-disjoint, busy and strong conflict free, but not conflict free.

Let $\mathcal{B}(X) \mathrel{\widehat{=}} \exists i, j.(\{i, i+1\}.black.on.j \notin X)$ and let $\overset{ab}{\hookrightarrow}$ be the directed cycle in the communication graph of the network which is represented by $\langle 00, 1a, \ldots, nn, (2m_b - 1)b, \ldots, 1b \rangle$. Now define functions

$$f_{ij} : \tilde{\mathcal{F}}[\![P_{ij}]\!] \times \{\overset{ab}{\hookrightarrow} \mid 0 < a, b < n \wedge a \neq b\} \rightarrow \mathbf{Z}$$

such that

$$f_{(2r)a}((s, X), \overset{ab}{\hookrightarrow}) = \begin{cases} -2r & \text{if } \mathcal{B}(X) \\ 2(m_a - r) & \text{otherwise} \end{cases}$$

$$(0 < r < m_a)$$

$$f_{(2r-1)a}((s, X), \overset{ab}{\hookrightarrow}) = \begin{cases} 2(m_a - r) + 1 & \text{if } \mathcal{B}(X) \\ -(2r-1) & \text{otherwise} \end{cases}$$

$$(0 < r \leq m_a)$$

$$f_{(2r)b}((s, X), \overset{ab}{\hookrightarrow}) = \begin{cases} 2r & \text{if } \mathcal{B}(X) \\ -2(m_b - r) & \text{otherwise} \end{cases}$$

$$(0 < r < m_b)$$

$$f_{(2r-1)b}((s, X), \overset{ab}{\hookrightarrow}) = \begin{cases} -[2(m_b - r) + 1] & \text{if } \mathcal{B}(X) \\ 2r - 1 & \text{otherwise} \end{cases}$$

$$(0 < r \leq m_a)$$

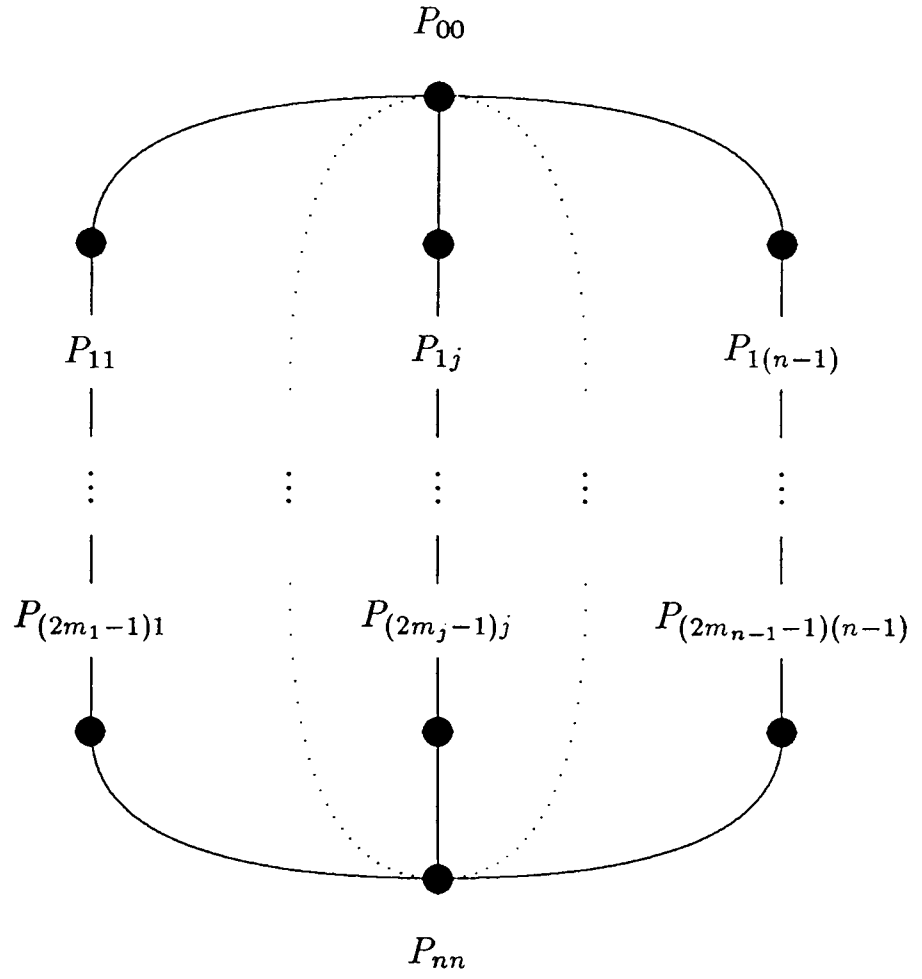

Figure 3.2: The communication graph of Example 2

and

$$f_{00}((s,X),\overset{ab}{\hookrightarrow}) = f_{nn}((s,X),\overset{ab}{\hookrightarrow}) = 0$$

where $\mathbf{Z}$ has its usual order $>$.

That the functions satisfy the conditions of the Directed Cycle Rule is readily verified as only one typical directed cycle needs to be examined. We conclude that the network is hereditarily deadlock free and again we note that, due to the lack of conflict freedom, the Variant Theorem could not have been applied. We also observe that the special case of the network in Example 3.1.1, which was mentioned earlier, could have been proved deadlock free directly by the Directed Cycle Rule. □

The main use of the Directed Cycle Rule is in networks where the number of directed cycles is small or where all the directed cycles and the behaviour of processes along them is quite similar, as in the examples above.

3.4 Selected Path Rules

Theorem 3.4.1 (Selected Path Rule) Let $V = \langle P_1, \ldots, P_n \rangle$ be a triple-disjoint, busy network with vocabulary Λ. Furthermore, suppose there exist functions

$$\begin{aligned} f_i &: \tilde{\mathcal{F}}[\![P_i]\!] \to \Pi \\ g_i &: \tilde{\mathcal{F}}[\![P_i]\!] \to \{1,\ldots,n\} \end{aligned} \qquad (1 \le i \le n)$$

where $(\Pi, >)$ is a partial order, such that if σ is a state of the subnetwork $\langle P_i, P_j \rangle$ then

$$P_i \xrightarrow{\sigma,\Lambda}\bullet\, P_j \;\Rightarrow\; P_i \xrightarrow{\sigma\restriction\alpha P_i} P_{g_i(\sigma\restriction\alpha P_i)}$$

and if $j = g_i(\sigma\restriction\alpha P_i)$,

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

Then V is hereditarily deadlock free.

Proof. We first show that the network is deadlock free. For the sake of contradiction let us assume it has a deadlock state δ. Define $r(i) = g_i(\delta\restriction\alpha P_i)$ for $1 \le i \le n$. From Lemma 1.2.1 we see every process is blocked in δ, thus $P_i \xrightarrow{\delta\restriction\langle P_i,P_j\rangle,\Lambda}\bullet P_j$ for some j. Then from the prerequisites of the theorem $P_i \xrightarrow{\delta\restriction\alpha P_i} P_{r(i)}$ which we may strengthen to $P_i \xrightarrow{\delta\restriction\langle P_i,P_{r(i)}\rangle,\Lambda}\bullet P_{r(i)}$ as P_i is blocked and which in turn yields $f_i(\delta\restriction\alpha P_i) > f_{r(i)}(\delta\restriction\alpha P_{r(i)})$.

Now observe that the process indices $i, r(i), r^2(i), \ldots, r^m(i), \ldots$ are distinct, because using the above in a trivial induction gives

$$f_{r^m(i)}(\delta\restriction\alpha P_{r^m(i)}) > f_{r^n(i)}(\delta\restriction\alpha P_{r^n(i)})$$

for $m < n$. But this contradicts the fact that our network is finite and so the network is deadlock free.

Now consider any subnetwork $\langle Q_1, \ldots, Q_k \rangle$ of V. Assume without loss of generality that it has vocabulary Γ ($\subseteq \Lambda$) and that $Q_i = P_{l(i)}$. Clearly the subnetwork is triple-disjoint and busy. Define functions

$$\begin{array}{l} \bar{f}_i : \tilde{\mathcal{F}}[\![Q_i]\!] \to \Pi \\ \bar{g}_i : \tilde{\mathcal{F}}[\![Q_i]\!] \to \{1, \ldots, k\} \end{array} \qquad (1 \leq i \leq k)$$

by putting

$$\bar{f}_i(\sigma) = f_{l(i)}(\sigma)$$

$$\bar{g}_i(\sigma) = \begin{cases} j & \text{if } g_{l(i)}(\sigma) = l(j) \text{ for } 1 \leq j \leq k \\ i & \text{otherwise} \end{cases}$$

We assert that these functions satisfy the conditions of the theorem for this subnetwork.

If $Q_i \xrightarrow{\sigma,\Gamma}\bullet\, Q_j$ then $P_{l(i)} \xrightarrow{\sigma,\Lambda}\bullet\, P_{l(j)}$ and so $P_{l(i)} \xrightarrow{\sigma\restriction\alpha P_{l(i)}} P_{g_{l(i)}(\sigma\restriction\alpha P_{l(i)})}$. But as $\alpha P_{l(i)} - X_{l(i)} \subseteq \Gamma$, we have $g_{l(i)}(\sigma\restriction\alpha P_{l(i)}) \in \{l(1), \ldots, l(k)\}$ showing that $Q_i \xrightarrow{\sigma\restriction\alpha Q_i} Q_{\bar{g}_i(\sigma\restriction\alpha Q_i)}$.

If $Q_i \xrightarrow{\sigma,\Gamma}\bullet\, Q_j$ we know $i \neq j$ and so if $j = \bar{g}_i(\sigma\restriction\alpha Q_i)$, we deduce from the definition of $\bar{g}_i$ that $l(j) = g_{l(i)}(\sigma\restriction\alpha P_{l(i)})$. Now as $Q_i \xrightarrow{\sigma,\Gamma}\bullet\, Q_j$, it follows that $P_{l(i)} \xrightarrow{\sigma,\Lambda}\bullet\, P_{l(j)}$ and so $f_{l(i)}(\sigma\restriction\alpha P_{l(i)}) > f_{l(j)}(\sigma\restriction\alpha P_{l(j)})$ which translates to $\bar{f}_i(\sigma\restriction\alpha Q_i) > \bar{f}_j(\sigma\restriction\alpha Q_j)$.

Thus, from the first part of the proof the subnetwork is deadlock free. We conclude that V is hereditarily deadlock free and that any subnetwork may be proved deadlock free by the Selected Path Rule. □

A function like g_i above will be termed a *selector* function, for obvious reasons.

Lemma 3.4.2 Any network that can be proved deadlock free by the Variant Theorem may also be proved so by the Selected Path Rule.

Proof. Just keep the same variant functions as for the Variant Theorem and let $g_i(\sigma) = \min(\{j \mid P_i \xrightarrow{\sigma} P_j\} \cup \{n\})$ say. □

An important point to note is that a network need not be strong conflict free to be amenable to the Selected Path Rule. Let $V = \langle P_1, \ldots, P_n \rangle$ be such a network with σ a strong conflict state of the subnetwork $\langle P_i, P_j \rangle$ and hence say $P_i \xrightarrow{\sigma,\Lambda}\bullet P_j$ and $P_j \xRightarrow{\sigma,\Lambda}\bullet P_i$, where Λ is the vocabulary of V. It follows that the selector function for P_i cannot be selecting P_j and the one for P_j must be selecting P_i. Furthermore, the request from P_i to P_j cannot also be strong, from the properties of the variant functions. (Alternatively, observe that if it was, then σ would be a deadlock state of $\langle P_i, P_j \rangle$, thus contradicting the hereditary deadlock freedom of V.) Hence the necessity of strong conflict freedom for subnetworks of size two has been weakened to that of just deadlock freedom for such subnetworks in the preconditions of this Rule. We shall not give an example of the application of the Selected Path Rule to a network with strong conflict, because although such networks are easy to contrive, they are harder to justify.

Again we have some example applications.

Example 3.4.1 Let k and m be constants such that $1 < m < k$. Define processes

$$P_0 = F_0(L_0)$$
$$\alpha P_0 = \{0.in, (k-1).to.0, 1.to.0, 0.to1, 0.out\}$$

$$P_m = F_m(R_m)$$
$$\alpha P_m = \{m.in, (m-1).to.m, (m+1).to.m, m.to.(m-1), m.out\}$$

$$P_i = F_i(L_i \sqcap R_i)$$
$$\alpha P_m = \{i.in, (i-1).to.i, (i+1).to.i, i.to.(i-1), i.to.(i+1), i.out\}$$
$$(1 \leq i < m) \text{ or } (m < i < k)$$

where

$$L_i = \Box \begin{array}{l} i.to.(i+1) \rightarrow SKIP \\ (i+1).to.i \rightarrow i.out \rightarrow L_i \end{array}$$

$$R_i = \Box \begin{array}{l} i.to.(i-1) \rightarrow SKIP \\ (i-1).to.i \rightarrow i.out \rightarrow R_i \end{array}$$

$$F_i(Q) \quad = \quad \begin{array}{l} \Box \\ \Box \end{array} \begin{array}{l} i.in \rightarrow Q\,;P_i \\ (i-1).to.i \rightarrow i.out \rightarrow P_i \\ (i+1).to.i \rightarrow i.out \rightarrow P_i \end{array}$$

with all arithmetic being modulo k. Further, let P_k be a deadlock free process such that

$$\alpha P_k \cap [\bigcup_{i=0}^{k-1} \alpha P_i\,] \subseteq \{0,\ldots,k-1\}.in$$

Consider the triple-disjoint and busy network $\langle P_0,\ldots,P_k\rangle$. Let W denote the subnetwork $\langle P_0,\ldots,P_{k-1}\rangle$. The subnetwork W is connected as a ring and each process in W is connected to the process P_k. W represents a network which is provided with data to process by P_k. Each datum requires processing by the first process it enters in W and also further processing by an adjacent process in W before leaving the ring via the latter process. An example is where k is even and processes in W are dedicated to computing one of two different functions (these functions being commutative on data) and where adjacent processes compute different functions. In detail, each process in W initially awaits receipt of data either from neighbours in the ring or from outside. On receiving data, this is processed and if it was received from another ring member it is sent out of the ring before the process returns to its initial state. If not, a decision on which neighbour it is going to be passed on to is taken. This decision is modelled by non-deterministic choice in the cases of P_i $(i \neq 0,m)$, but P_0,P_m always choose P_1,P_{m-1} respectively. If the process chosen has itself data to transfer on the same communication link, the processes are prepared to delay the transfer to later, after accepting, processing and passing out the other's data. A successful transfer returns the process to its initial state.

Now define functions

$$\begin{array}{ll} f_i : \tilde{\mathcal{F}}[\![P_i]\!] \rightarrow \mathbf{N} & \\ & (0 \leq i \leq k) \\ g_i : \tilde{\mathcal{F}}[\![P_i]\!] \rightarrow \{0,\ldots,k\} & \end{array}$$

by putting

$$\begin{array}{rcl} f_k(\sigma) & = & k \\ g_i(\sigma) & = & \max(\{0\} \cup \{j \mid P_i \xrightarrow{\sigma} P_j\}) \end{array}$$

and

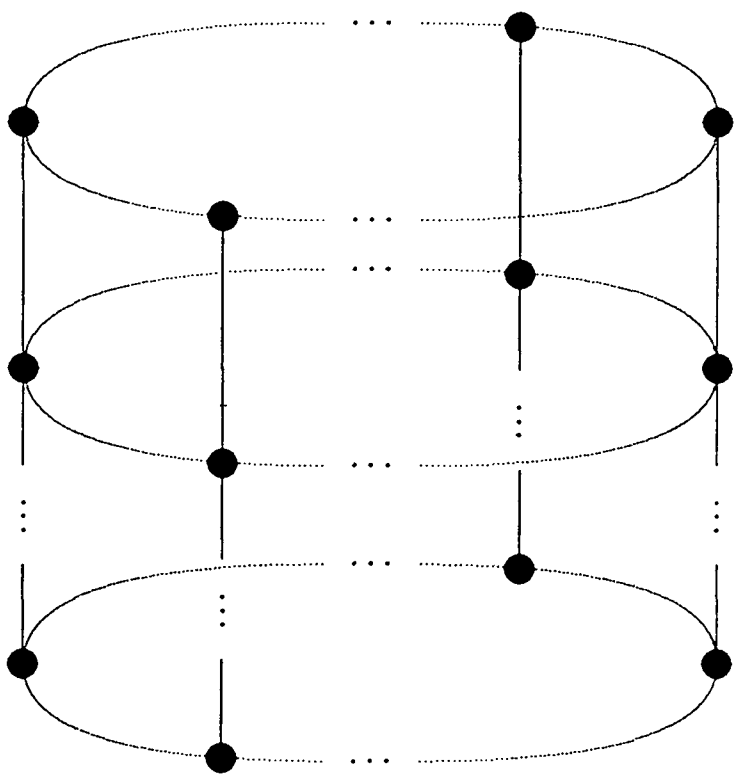

Figure 3.3: A special case of Example 1

$$f_i(\sigma) \;=\; \begin{cases} k+1 & \text{if } P_i \xrightarrow{\sigma} P_k \\ i & \text{if } P_i \xrightarrow{\sigma} P_{i-1} \wedge \neg(P_i \xrightarrow{\sigma} P_k) \\ m-i & \text{if } 0 \le i \le m \wedge \neg(P_i \xrightarrow{\sigma} P_{i-1} \vee P_i \xrightarrow{\sigma} P_k) \\ m+k-i & \text{if } m < i < k \wedge \neg(P_i \xrightarrow{\sigma} P_{i-1} \vee P_i \xrightarrow{\sigma} P_k) \end{cases}$$

$$(0 \le i < k)$$

To establish adherence to the conditions of the Selected Path Rule for the functions is routine and hence we conclude that the network is hereditarily deadlock free.

We note that the network is not conflict free (although it is strong conflict free) and so proof of deadlock freedom could not have been attempted by the Variant Theorem. Moreover, the state of the network corresponding to the trace $\langle\rangle$ has a proper cycle of ungranted requests $\langle 0, 1, \ldots, k\rangle$ and thus the network is not acyclic deadlock free.

Observe the communication graph in Figure 3. Each ring in the figure is a network like W. All the rings have the same constant k though m may vary. The "out" channels of a ring are connected to the corresponding "in" channels of the ring above, if there is one. Let V_n be such a network with n rings. We assert that V_n is hereditarily deadlock free for all n.

V_1 is just W which is a subnetwork of a hereditarily deadlock free network and thus itself hereditarily deadlock free. Inductively, say V_n is hereditarily deadlock free. Then putting $P_k = PAR(V_n)$, we see that V_{n+1} is a special case of the network proved hereditarily deadlock free above. So the assertion is proved. □

We revisit Example 3.3.1 in our next example.

Example 3.4.2 The network in Example 3.3.1 is triple-disjoint and busy. Now define functions

$$\begin{aligned} f_i &: \tilde{\mathcal{F}}[\![P_i]\!] \to \mathbf{Z} \\ g_i &: \tilde{\mathcal{F}}[\![P_i]\!] \to \{0, \ldots, n-1\} \end{aligned} \qquad (0 \le i < n)$$

such that

$$f_0(\sigma) \;=\; 0$$

$$f_i((s, X)) \;=\; \begin{cases} n-i & \text{if } i.to.(i+1).black \notin X \\ -i & \text{otherwise} \end{cases} \qquad (0 < i < n)$$

$$g_i(\sigma) \;=\; (i+1) \bmod n$$

where $\mathbf{Z}$ has its usual order $>$.

The conditions for the Selected Path Rule are then satisfied and so hereditary deadlock freedom is apparent for the network. The use of this rule here has enabled us to choose the directed cycle over which variants decrease, rather than being forced to ensure that this is so for all the directed cycles, as with the Directed Cycle Rule. This has resulted in the simplification of the variant functions given previously for the network. □

We now present another version of the Selected Path Rule. We postpone to the next section the discussion of its relationship to the original version.

Theorem 3.4.3 (Selected Path Rule*) Let $V = \langle P_1, \ldots, P_n \rangle$ be a triple-disjoint, busy network with vocabulary Λ. Furthermore, suppose there exist functions

$$\begin{array}{l} f_i : \tilde{\mathcal{F}}[\![P_i]\!] \to \Pi \\ g_i : \tilde{\mathcal{F}}[\![P_i]\!] \to \{1, \ldots, n\} \end{array} \qquad (1 \leq i \leq n)$$

where $(\Pi, >)$ is a partial order, such that if σ is a state of the subnetwork $\langle P_i, P_j \rangle$ then

$$P_i \xrightarrow{\sigma, \Lambda}\bullet\, P_j \quad \Rightarrow \quad P_i \xrightarrow{\sigma \restriction \alpha P_i} P_{g_i(\sigma \restriction \alpha P_i)}$$

and if $j = g_i(\sigma \restriction \alpha P_i)$,

$$P_i \xrightarrow{\sigma, \Lambda}\bullet\, P_j \quad \Rightarrow \quad \begin{cases} g_j(\sigma \restriction \alpha P_j) \neq i & \text{if } \{i, j\} \text{ is disconnecting} \\ f_i(\sigma \restriction \alpha P_i) > f_j(\sigma \restriction \alpha P_j) & \text{otherwise} \end{cases} .$$

Then V is hereditarily deadlock free.

Proof. We first show that the network is deadlock free. For the sake of contradiction let us assume it has a deadlock state δ. Define $r(i)$ as in the proof of Theorem 3.4.1 and recall from there that for any process index i, we have $P_i \xrightarrow{\delta \restriction \langle P_i, P_{r(i)} \rangle, \Lambda}\bullet\, P_{r(i)}$.

Now consider the digraph D which is constructed as follows: its nodes are the essential components in the communication graph of V and if C and C' are two such nodes, $\langle C, C' \rangle$ is an edge in D if and only if $\{i, r(i)\}$ is a disconnecting edge with i in C and $r(i)$ in C'. We note that there is no disconnecting edge $\{i, j\}$ in the communication graph of V such that $r(i) = j$ and $r(j) = i$ as then we would have $P_i \xrightarrow{\delta \restriction \langle P_i, P_{r(i)} \rangle, \Lambda}\bullet\, P_{r(i)}$ with $r(r(i)) = i$ contrary to the preconditions of the theorem. Thus we deduce that D is

an acyclic digraph and hence has a sink node C (a node with no outgoing edges). Then if i is any process index in C, we know $\{i, r(i)\}$ cannot be a disconnecting edge and furthermore that $r(i)$ is itself in C. So, as in the proof of Theorem 3.4.1, we may construct an infinite sequence $i, r^2(i), \ldots$ of distinct process indices which contradicts the finiteness of the network. Hence V is deadlock free.

We may prove V is hereditarily deadlock free, again using a similar argument to the proof of Theorem 3.4.1, by showing that every subnetwork of V is also amenable to the Selected Path Rule*. □

In the case of a network without disconnecting edges the two versions of the Rule coincide. Hence, the Selected Path Rule* is able to treat all the examples given in this section.

3.5 Comparison of the Rules

All the deadlock freedom rules given in this chapter are strictly more general than the Variant Theorem. This has been done by providing more information on the topology of the network, thus enabling the rules to focus more closely on areas directly relevant to the deadlock freedom of the network and the preconditions to be less stringent elsewhere.

In the Selected Edge Rule or Selected Path Rule, the provision of this information (via either selected edges or selector functions respectively) is reliant on the judicious choice of the person applying the rule. Thus simple categorisations of when exactly these rules can be applied are not apparent. In the absence of such a categorisation for the Selected Edge Rule, we note a useful necessary condition for its use. It is that every directed cycle in the communication graph contains at least one edge whose corresponding subnetwork is conflict free. In other words, the subgraph consisting of edges corresponding to subnetworks which are strong conflict free but not conflict free is a forest.

In the Directed Cycle Rule, the topological information is not provided selectively and a statement of precisely when its use is permissible is as follows.

Theorem 3.5.1 Suppose $V = \langle P_1, \ldots, P_n \rangle$ is a triple-disjoint, busy and strong conflict free network with vocabulary Λ. Let D be the set of directed cycles in the communication graph, and let Π and $\succ$ be as defined in Theorem 2.3.3. Now put $\Pi' = \Pi \times D$ and define the relation $\succ'$ on Π' as

follows:

$$((\sigma_i, i), d_i) \succ' ((\sigma_j, j), d_j) \quad \Leftrightarrow \quad \begin{cases} d_i = d_j \\ \langle i, j \rangle \text{ is in } d_i \\ \text{and } (\sigma_i, i) \succ (\sigma_j, j). \end{cases}$$

Then V can be proved deadlock free by the Directed Cycle Rule if and only if the transitive closure of $\succ'$ on Π' is a partial order.

Proof. This is similar to that for Theorem 2.3.3. □

We can now use this result to show the following.

Theorem 3.5.2 Any network that can be proved deadlock free by the Selected Edge Rule may also be proved so by the Directed Cycle Rule.

Proof. Let $V = \langle P_1, \ldots, P_n \rangle$ be a network which can be proved deadlock free by the Selected Edge Rule with variant functions f_i and selected set of directed edges E. Assume for the sake of contradiction that V cannot be proved deadlock free by the Directed Cycle Rule. Then by Theorem 3.5.1, the transitive closure of $\succ'$ is not a partial order. This can only be if there is a sequence

$$((\sigma_0, i_0), d) \succ' ((\sigma_1, i_1), d) \succ' \cdots \succ' ((\sigma_m, i_m), d)$$

where $\sigma_0 = \sigma_m$ and $i_0 = i_m$. As d is a directed cycle in the communication graph, we know it has an edge in E, say $\langle i_0, i_1 \rangle$ without loss of generality. But by the properties of the variant functions and the definition of $\succ'$, we would have

$$f_{i_0}(\sigma_0) > f_{i_1}(\sigma_1) \geq \cdots \geq f_{i_m}(\sigma_m)$$

and hence that $f_{i_0}(\sigma_0) > f_{i_0}(\sigma_0)$, which is the required contradiction. Thus the Directed Cycle Rule can be applied to V. □

Note that for both the networks proved deadlock free in Section 3 by the Directed Cycle Rule, no subnetwork of the networks was conflict free and thus the Selected Edge Rule could not have been used on either. So the Directed Cycle Rule is strictly more general than the Selected Edge Rule.

We now proceed with the promised discussion of the relationship between the Selected Path Rule and the Selected Path Rule*.

Theorem 3.5.3 Any network that can be proved deadlock free by the Selected Path Rule may also be proved so by the Selected Path Rule*.

Proof. Let $V = \langle P_1, \ldots, P_n \rangle$ be a network whose vocabulary is Λ and which may be proved deadlock free by the Selected Path Rule with variant functions f_i and selector functions g_i. Now define new functions

$$g_i' : \tilde{\mathcal{F}}[\![P_i]\!] \to \{1, \ldots, n\} \qquad (1 \leq i \leq n)$$

by putting

$$g_i'(\sigma) = \begin{cases} i & \text{if } (\sigma, i) \text{ is maximal with respect to } \succ \\ g_i(\sigma) & \text{otherwise} \end{cases}$$

where $\succ$ is as defined in Theorem 2.3.3. We assert that V can be proved deadlock free by the Selected Path Rule* with variant functions f_i and selector functions g_i'.

The verification of this is straightforward except for the following case. Say σ is a state of the subnetwork $\langle P_i, P_j \rangle$ such that $j = g_i'(\sigma \restriction \alpha P_i)$, $\{i, j\}$ is a disconnecting edge and $P_i \xrightarrow{\sigma,\Lambda}\bullet P_j$. We need to prove that $g_j'(\sigma \restriction \alpha P_j) \neq i$, so assume for the sake of contradiction that $g_j'(\sigma \restriction \alpha P_j) = i$. Thus we have $g_j'(\sigma \restriction \alpha P_j) \neq j$ (as $i \neq j$) and so we see that $g_j'(\sigma \restriction \alpha P_j) = g_j(\sigma \restriction \alpha P_j)$ and that $(\sigma \restriction \alpha P_j, j)$ is not maximal with respect to $\succ$ (by the definition of g_j'). The latter fact infers the existence of a state $\bar{\sigma}$ of some subnetwork $\langle P_j, P_k \rangle$ such that $P_j \xrightarrow{\bar{\sigma},\Lambda}\bullet P_k$ with $\bar{\sigma} \restriction \alpha P_j = \sigma \restriction \alpha P_j$. But by the properties of g_j this yields $P_j \xrightarrow{\bar{\sigma} \restriction \alpha P_j} P_{g_j(\bar{\sigma} \restriction \alpha P_j)}$ or $P_j \xrightarrow{\sigma \restriction \alpha P_j} P_i$. We may strengthen this to $P_j \xrightarrow{\sigma,\Lambda}\bullet P_i$ (as $P_i \xrightarrow{\sigma,\Lambda}\bullet P_j$) and noting that $i = g_j(\sigma \restriction \alpha P_j)$ shows that $f_j(\sigma \restriction \alpha P_j) > f_i(\sigma \restriction \alpha P_i)$. Now $(\sigma \restriction \alpha P_i, i)$ is also not maximal with respect to $\succ$ (as $(\sigma \restriction \alpha P_i, i) \succ (\sigma \restriction \alpha P_j, j)$) and so $g_i'(\sigma \restriction \alpha P_i) = g_i(\sigma \restriction \alpha P_i)$. But then $P_i \xrightarrow{\sigma,\Lambda}\bullet P_j$ with $j = g_i(\sigma \restriction \alpha P_i)$ and so we have $f_i(\sigma \restriction \alpha P_i) > f_j(\sigma \restriction \alpha P_j)$ which is a contradiction.

Hence we conclude that V may be proved deadlock free by the Selected Path Rule*. □

At first glance, it may appear that the Selected Path Rules are more powerful than both the Selected Edge Rule and the Directed Cycle Rule. Certainly, they proved deadlock free (in Example 3.4.2) the network handled by the Directed Cycle Rule in Example 3.3.1, and the reader may find it of interest to show that the network of Example 3.2.2 can also be treated by the Selected Path Rules. Moreover, the network in Example 3.4.1 is amenable to these Rules and not to the others (as it is not acyclic deadlock free) and the

preconditions of the Selected Path Rules do not even require strong conflict freedom, unlike the others. Surprisingly however, this conjecture is untrue.

Recall the network in Example 3.2.2. This network was proved deadlock free by the Selected Edge Rule there (and so by Theorem 3.5.2 can also be treated by the Directed Cycle Rule). We assert that it cannot be proved deadlock free by the Selected Path Rule, so assume for the sake of contradiction that it can. Most of the requests from the processes are strong requests and the only ambiguity in ascertaining what the selector functions are selecting occurs in a state of a "middle" process in which it is prepared to communicate with two of its neighbours. Let σ_{00} and σ_{10} be the initial such states of P_{00} and P_{10} respectively, and let σ'_{00} and σ'_{10} be the corresponding initial states in which this does not occur. Now say the application of the appropriate selector functions to σ_{00} and σ_{10} yields the process indices $0n_0$ and $1n_1$ respectively. Then it is simple to construct a cycle

$$(\sigma_{00}, 00) \succ (\sigma_{0n_0}, 0n_0) \succ (\sigma_{0\bar{n}_0}, 0\bar{n}_0) \succ (\sigma'_{00}, 00) \succ (\sigma_{10}, 10) \succ$$

$$(\sigma_{1n_1}, 1n_1) \succ (\sigma_{1\bar{n}_1}, 1\bar{n}_1) \succ (\sigma'_{10}, 10) \succ (\sigma_{00}, 00)$$

where $\bar{1} = 2$ and $\bar{2} = 1$. But then the values of the variant functions must be strictly decreasing along this cycle, which is a contradiction. Thus the Selected Path Rule is not more general than the Selected Edge Rule (and hence the Directed Cycle Rule). The Selected Path Rule* though can be applied to this network. We leave this for the reader to verify, the important factor here being that values of variant functions need not be compared across disconnecting edges for its use. We note in passing that this shows the Selected Path Rule* is strictly more general than the Selected Path Rule.

Now consider the following amendment of the network above. We simply replace the two "middle" processes with a single one so that the communication graph is now connected like a $\bowtie$ and thus has no disconnecting edges. The new process initially decides non-deterministically whether to communicate to the "left" or to the "right". Having made this choice, it is always prepared to accept any communication from the chosen direction. Using a similar argument to the one used earlier, we observe that the Selected Path Rule* cannot be used with this network, but it is a simple matter to check that the Directed Cycle Rule can. Thus the Selected Path Rule* too is not more powerful than the Directed Cycle Rule. We do however have the following positive result which although easy to state and intuitively apparent, does not admit an easy proof.

Theorem 3.5.4 Any network that can be proved deadlock free by the Selected Edge Rule may also be proved so by the Selected Path Rule*.

Proof. Let $V = \langle P_1, \ldots, P_n \rangle$ be a network (with vocabulary Λ) satisfying the preconditions of the theorem and let E be the set of selected edges used in its deadlock freedom proof by the Selected Edge Rule. We may assume, without loss of generality, that E contains no edge which corresponds to a disconnecting edge in the communication graph of V, for if there were one we could just as well remove it from E and still satisfy the preconditions. Further, let G denote the graph obtained from the communication graph of V by removing edges which have a corresponding directed edge in E. We note that G is a forest because every cycle in the communication graph of V contains an edge which has a corresponding directed edge in E. Thus we can define a partial order $>_G$ on the nodes of G such that: every node is comparable with all its neighbours and each node has at most one neighbour greater than it (its ancestor). Say Π and $\succ$ are as defined in Theorem 2.3.3. Consider the following definition of a function $\Psi : \Pi \to \{1, \ldots, n\}$.

$$\Psi((\sigma, i)) = \begin{cases} & i & \text{if } (\sigma, i) \text{ is maximal with respect to } \succ \\ \text{if not} & j & \text{if } P_i \xrightarrow{\sigma} P_j,\ \{i,j\} \text{ is not an edge in } G \text{ and } j \text{ is the smallest such index} \\ \text{if not} & j & \text{if } P_i \xrightarrow{\sigma} P_j,\ i >_G j \text{ and } j \text{ is the smallest such index} \\ \text{if not} & j & \text{if } P_i \xrightarrow{\sigma} P_j \text{ and } j \text{ is the unique node such that } j >_G i \end{cases}$$

Clearly the function is well-defined. The busy-ness of the network ensures that these are the only possibilities and thus the function is total.

We now define a relation $\succ_\Psi$ on Π as follows:

$$(\sigma_1, i_1) \succ_\Psi (\sigma_2, i_2) \quad \Leftrightarrow \quad \begin{cases} \Psi((\sigma_1, i_1)) = i_2 \\ (\sigma_1, i_1) \succ (\sigma_2, i_2) \\ \text{and } \{i_1, i_2\} \text{ is not disconnecting.} \end{cases}$$

We assert that the transitive closure of $\succ_\Psi$ on Π is a partial order. Assume for the sake of contradiction that this is not so. This can only be if there is a cycle

$$(\sigma_0, i_0) \succ_\Psi (\sigma_1, i_1) \succ_\Psi \cdots \succ_\Psi (\sigma_m, i_m) \succ_\Psi (\sigma_0, i_0).$$

In the following consideration let all arithmetic be modulo $m + 1$.

First note that none of the edges $\{i_{r-1}, i_r\}$ in the cycle are disconnecting. Furthermore, none of the edges $\langle i_{r-1}, i_r\rangle$ can be in E. If there were such an edge, we would have the values of the variant functions (provided by the Selected Edge Rule) at least comparable and not increasing along the edges in the cycle, and strictly decreasing along the edge in E, whence the contradiction.

We now show that $i_{r-1} \neq i_{r+1}$ for any of the indices in the cycle. Assume then, again for the sake of contradiction, that $i_{k-1} = i_{k+1}$ somewhere in the cycle. As neither of the edges $\langle i_{k-1}, i_k\rangle$ or $\langle i_k, i_{k+1}\rangle$ ($\langle i_k, i_{k-1}\rangle$) can be in E we see, from the construction of G, that $\{i_{k-1}, i_k\}$ must be an edge in G. We also observe that we have

$$(\sigma_{k-1}, i_{k-1}) \succ (\sigma_k, i_k) \succ (\sigma_{k+1}, i_{k-1})$$

and hence there is a conflict state σ of the subnetwork $\langle P_{i_{k-1}}, P_{i_k}\rangle$ where $\sigma{\restriction}\alpha P_{i_{k-1}} = \sigma_{k-1}$ and $\sigma{\restriction}\alpha P_{i_k} = \sigma_k$. As V is strong conflict free it follows that there are process indices j_{k-1} and j_k such that $j_{k-1} \neq i_k$, $j_k \neq i_{k-1}$, $P_{i_{k-1}} \xrightarrow{\sigma_{k-1}} P_{j_{k-1}}$ and $P_{i_k} \xrightarrow{\sigma_k} P_{j_k}$, where we choose j_{k-1} and j_k to be the smallest such indices. Now i_{k-1} and i_k are neighbouring nodes of G, so either $i_{k-1} >_G i_k$ or $i_k >_G i_{k-1}$ holds. In the first case we surmise that either $\{i_k, j_k\}$ is not in G or that $i_k >_G j_k$ (the latter being the case because j_k is a neighbouring node of i_k which is not the unique ancestor of i_k). But then $\Psi((\sigma_k, i_k)) = j_k$ which contradicts the fact that $\Psi((\sigma_k, i_k)) = i_{k-1}$. A similar contradiction results in the second case when $i_k >_G i_{k-1}$.

The result above then reveals that $\langle i_0, i_1, \ldots, i_m, i_0\rangle$ is in fact a directed cycle in the communication graph of V, and so via the preconditions of the Selected Edge Rule it must have an edge in E. However, we noted earlier that there can be no such edge in the cycle, and so from this contradiction we infer that the transitive closure of $\succ_\Psi$ on Π is a partial order.

Now define functions

$$\begin{aligned} f_i &: \tilde{\mathcal{F}}[\![P_i]\!] \to \Pi \\ g_i &: \tilde{\mathcal{F}}[\![P_i]\!] \to \{1, \ldots, n\} \end{aligned} \qquad (1 \leq i \leq n)$$

by putting

$$\begin{aligned} f_i(\sigma) &= (\sigma, i) \\ g_i(\sigma) &= \Psi((\sigma, i)) \end{aligned}$$

and order Π by the transitive closure of $\succ_\Psi$. That these definitions satisfy the preconditions of the Selected Path Rule* is easy to check except perhaps for the following case.

Say $P_i \xrightarrow{\sigma,\Lambda}\bullet P_j$ where $j = g_i(\sigma\restriction\alpha P_i)$ and $\{i,j\}$ is disconnecting. Assume for the sake of contradiction that $g_j(\sigma\restriction\alpha P_j) = i$. As $i \neq j$, we see by the definition of Ψ that $P_j \xrightarrow{\sigma} P_i$ and hence that $P_j \xrightarrow{\sigma,\Lambda}\bullet P_i$. Thus σ is a conflict state of the subnetwork $\langle P_i, P_j\rangle$ with $\Psi((\sigma\restriction\alpha P_i, i)) = j$ and $\Psi((\sigma\restriction\alpha P_j, j)) = i$. Moreover, from our assumption that E contains no directed edges corresponding to disconnecting edges, we see that neither $\langle i,j\rangle$ or $\langle j,i\rangle$ can be in E and hence that $\{i,j\}$ must be an edge in G. Then arguing as earlier we contradict the well-definedness of Ψ.

Therefore we finally conclude that V may be proved deadlock free by the Selected Path Rule*. □

Although the Selected Path Rule is not more general than the Selected Edge Rule, many networks encountered in practice which can be proved deadlock free by the latter may also be proved so by the former. For instance, we have observed that the preconditions of the Selected Path Rules coincide on networks with no disconnecting edges, so using the above result, if such a network can be proved deadlock free by the Selected Edge Rule then it may also be proved so by the Selected Path Rule. We can generalise this result slightly by noting that the presence of a disconnecting edge (in the network of Example 3.2.2) along which variant functions must increase with respect to ungranted requests (if such a comparison was forced), played a crucial role in its failure there.

Corollary 3.5.5 A network that can be proved deadlock free by the Selected Edge Rule in a way such that the values of its variant functions along ungranted requests on a disconnecting edge are now comparable and do not increase, may also be proved deadlock free by the Selected Path Rule.

The extra condition above merely ensures that the disconnecting edges of the network assume the same status as the other edges in the preconditions of the Selected Edge Rule.

Proof. Similar to that of Theorem 3.5.4. □

Of course in the case of a network whose subnetworks corresponding to disconnecting edges are conflict free, we may by Lemma 1.3.4 decompose the network into its essential components and analyse these separately. The amenability of these essential components to the Selected Edge Rule would then imply their amenability to the Selected Path Rule.

As mentioned earlier, the Rules in this chapter have been local in nature for ease of use. It is thus not surprising that networks which can be handled

by these rules must be hereditarily deadlock free. Indeed, the Rules too are hereditary, so that the subnetworks of a network proved deadlock free by a Rule are themselves amenable to the same Rule. We shall see in the next section that there are networks which are hereditarily and acyclic deadlock free but which still cannot be treated by these Rules. A visual summary of the scope of the Rules is provided in Figure 3.4.

3.6 Allowing for Termination

In the failures model, termination is represented by the event $\surd$ (tick). In the definition of a CSP process, this event must never be introduced explicitly but only via the use of the process *SKIP*, the process that immediately terminates (does a $\surd$) and then refuses to do anything else. For us, the notion of what a process does after termination is a meaningless one but we note that if $s\langle\surd\rangle$ is a trace of a process P, then P after $(s\langle\surd\rangle)$ is not deadlock free. Thus to distinguish successful termination from deadlock, we extend the definition of deadlock freedom (in Definition 1.1.1) to the following.

Definition 3.6.1 A process P is said to be *deadlock free* if and only if

$$\forall s : (\alpha P)^* \mid s \text{ tick-free}.\ (s, \alpha P) \notin \mathcal{F}[\![P]\!] \ .$$

We say that a process P is *willing to terminate* in a state $(s, X) \in \tilde{\mathcal{F}}[\![P]\!]$ if $\alpha P - X \supseteq \{\surd\}$, and that P is *willing only to terminate* in (s, X) if $\alpha P - X = \{\surd\}$.

Turning our attention to networks, we see that a network can terminate only when all its individual processes are willing to terminate. A network may include a process that will never terminate in which case the whole network will never terminate (this in no way prejudicing its deadlock freedom). In the latter case consider the subcase where the process that never terminates has no use of *SKIP* in its definition and thus need not have $\surd$ in its alphabet. This may give rise to an absurd situation where there is a non tick-free trace of the whole network with events after the $\surd$, so showing termination of the network when this is not so. To exclude such behaviour, we shall assume from now on that all the networks we deal with obey the following convention.

(Termination Convention) Every process in a network contains $\surd$ in its alphabet.

We now amend many of our definitions pertaining to networks in order to accommodate termination.

From our earlier observation, the only states that we need be interested in are the ones before termination. Hence our definition of states will now only encompass those whose corresponding trace is tick-free.

We redefine our notions of request, ungranted request, triple-disjoint, vocabulary and communication graph by simply substituting all occurrences of αP_i by $\alpha P_i - \{\surd\}$ in the original definitions. These changes reflect the fact that $\surd$ is an event in which many processes may participate rather than an event communicated along a link between two processes, the latter being the case which we want the definitions above to deal with. Our revised definition of a blocked process is then as follows.

Definition 3.6.2 Let $\langle P_1, \ldots, P_n \rangle$ be a triple-disjoint network with vocabulary Λ. We say P_i is *blocked* in a state σ if

(i) $P_i \xrightarrow{\sigma} P_j$ for some j, or P_i is willing to terminate.

(ii) $P_i \xrightarrow{\sigma,\Lambda}\bullet P_j$ whenever $P_i \xrightarrow{\sigma} P_j$, and if P_i is willing to terminate then not all the other processes are.

The deadlock state characterisation result of Lemma 1.2.1 now takes the following form.

Lemma 3.6.1 A state of a triple-disjoint, busy network is a deadlock state if, and only if, there is a process which is not willing only to terminate in the state and every such process is blocked.

Proof. Obvious. □

Requests to processes which are willing only to terminate are permissible within our definitions. Such requests are necessarily ungranted and moreover, will remain so unless the request is withdrawn. It is the existence of such requests in networks which contain the possibility of termination that increase the likelihood of deadlock and hence it is desirable that networks should be designed, as far as possible, to exclude their occurrence. This can often be done by making networks pursue a more careful course with regard to termination, such as having every process notify its neighbours of its impending termination and thus instructing them not to attempt any further communication with it. In general, verification of the absence of requests to

processes which are willing only to terminate needs global consideration, but for many networks this is unnecessary. Thus in keeping with our principle of local analysis, we present the following definition.

Definition 3.6.3 Let $V = \langle P_1, \ldots, P_n \rangle$ be a triple-disjoint network. We say V is *prudent* if whenever $P_i \xrightarrow{\sigma} P_j$ in a state σ of the subnetwork $\langle P_i, P_j \rangle$, then P_j is not willing only to terminate.

So a prudent network is one that can be shown not to contain requests to processes which are willing only to terminate by just pairwise analysis. The following result then shows that such networks fit into our methodology of deadlock analysis.

Theorem 3.6.2 The substitution of a triple-disjoint, prudent network for a triple-disjoint network in the deadlock freedom Rules preserves their validity.

Proof. Assume a triple-disjoint, busy and prudent network satisfying the preconditions of a Rule has a deadlock state. Then there is at least one process which is not willing only to terminate in this state. Consider the subnetwork of all such processes. There are no requests outside this subnetwork (as any such requests would have to be from or to processes which are willing only to terminate). Moreover, every process in the subnetwork is blocked in the old sense. Recalling that the preconditions of the rules are hereditary and noting that this is a deadlock state of the subnetwork in the old sense, we conclude that this circumstance is precluded by any of the Rules. It follows similarly that the type of deadlock freedom enjoyed by the network is the one stated in the Rule applied. □

A more general result is possible in the case of the Selected Path Rule.

Theorem 3.6.3 Suppose that a set of variant functions satisfies the condition that the value of each variant function for a state in which a process is willing only to terminate is maximal with respect to the partial order on its co-domain. Furthermore, if the functions also satisfy the preconditions of the Variant Theorem or the Selected Path Rule then the validity of that particular Rule is preserved.

Proof. In the case of the Variant Theorem, we see that the condition on the variant functions can only hold if the network concerned is prudent. Hence the result follows from above. For the Selected Path Rule, the proof is much the same as for the Rule in its original form. □

The condition above is equivalent to that of a network being prudent when used with the Variant Theorem, but allied with the Selected Path Rule, it does allow requests to processes which are willing only to terminate, as long as every process has a selected request which is not of this form. This condition on variant functions is insufficient for use with the other Rules. Consider the example of a deadlock state, in a triple-disjoint network of size two, where one process is willing only to terminate and is blocking the other process. The condition cannot exclude such a state as the values of the variant functions of the processes would not require comparison in this case.

Clearly, the deadlock freedom of a network does not guarantee its termination. However, in the special case where each individual process in a network always reaches a state in which it is willing only to terminate, we may deduce the eventual termination of the network from its deadlock freedom.

We now proceed to a few examples.

Example 3.6.1 We consider two networks based on Lemma 2.3.5.

V is a triple-disjoint, busy and conflict free network where each process communicates with each of its neighbours once on each "cycle" and terminates after j such "cycles", j being a fixed constant. Suppose further that there is a partial order on the elements of V 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.

W is a triple-disjoint, busy and conflict free network. Each process communicates only once with its neighbours on each "cycle" and which neighbours it communicates with is determined by the presence of the neighbour in a set kept by the process (initially this contains all its neighbours). If during the course of its communication the process receives a special message from a neighbour telling it not to communicate with it anymore, this neighbour is removed from its set. At the end of a "cycle", a process may non-deterministically decide to terminate at the end of the next "cycle". In this case, it sends its neighbours notice of its impending termination with its usual communications in the following "cycle". If at the end of a "cycle" the processes set is empty though, it immediately terminates. Suppose further that there is a partial order on the elements of W 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 and in its set in parallel.

V and W are both prudent and adapting the variant functions alluded to in Lemma 2.3.5, they are seen to be amenable to the Variant Theorem. V does terminate, as each process in it must, but W does not necessarily. □

Example 3.6.2 Recall the network of Example 2.3.2. We amend the network there so that now on the receipt of a signal *shutdown* the network terminates.

For each process P_{nr} in the earlier network define a corresponding process Q_{nr} as follows:

$$Q_{00} = shutdown \rightarrow halt.\{00, 10\} \rightarrow SKIP$$

for $n > 0$ we put

$$Q_{n0} = halt.\{n0, (n-1)0\} \rightarrow \left(\begin{array}{l} \| \quad halt.\{n0, (n+1)0\} \rightarrow SKIP \\ \quad\; halt.\{n0, n1\} \rightarrow SKIP \end{array} \right)$$

and

$$Q_{nn} = halt.\{nn, n(n-1)\} \rightarrow SKIP$$

while for $0 < r < n$ we have

$$Q_{nr} = halt.\{nr, n(r-1)\} \rightarrow halt.\{nr, n(r+1)\} \rightarrow SKIP \quad .$$

The new network is then that consisting of the processes $P_{nr} \stackrel{\Delta}{} Q_{nr}$ where the alphabet of each process is just the set of all events used in its definition together with $\surd$. (The $\stackrel{\Delta}{}$ operator is the "interrupt" operator defined in [H1] and whose failures semantics we give in the Appendix. The process $P \stackrel{\Delta}{} Q$ may deterministically become Q after any progress in P.)

The network is trivially triple-disjoint and busy. Now define a partial order $>$ on the set

$$\{(n, r) \mid 0 \leq r \leq n\} \cup \{\top\}$$

by putting

$$(n, r) \geq (n', r') \Leftrightarrow n \geq n' \wedge r \geq r'$$

and making $\top$ the maximum element.

Variant functions f_{nr} take the form

$$f_{nr}((s, X)) = \begin{cases} (n, r) & \text{if } \surd \in X \\ \top & \text{otherwise} \end{cases}$$

while we define selector functions g_{nr} such that $g_{nr}((s,X)) = n'r'$ where

$$\Big(\alpha(P_{nr} \stackrel{\Delta}{} Q_{nr}) - X\Big) \cap \alpha Q_{nr} \cap \alpha Q_{n'r'} \neq \emptyset \text{ and } (n,r) \neq (n',r') \quad .$$

It is clear that the functions are well-defined, satisfy the preconditions of the Selected Path Rule and further that when a process is willing only to terminate, its variant function value is maximal with respect to the partial order defined. Hence we conclude by Theorem 3.6.3 that the network is deadlock free.

We observe that this network is not prudent as for instance a process $P_{nr} \stackrel{\Delta}{} Q_{nr}$ $(0 < r < n)$ may have requests to the processes $P_{(n+1)r} \stackrel{\Delta}{} Q_{(n+1)r}$ and $P_{(n+1)(r+1)} \stackrel{\Delta}{} Q_{(n+1)(r+1)}$ when both of these are willing only to terminate.

An alternative proof of the network's deadlock freedom is the following. Consider the network consisting just of the processes Q_{nr} together with the order $>$ which is now used to order these processes in the obvious way. This network is then a case of a network V as described in Example 3.6.1 and is thus deadlock free. Now noting that the alphabet of this network and that of the one in Example 2.3.2 are disjoint, we may deduce from Lemma 4.3.1 (v) that the network consisting of the processes $P_{nr} \stackrel{\Delta}{} Q_{nr}$ is deadlock free as required.

This latter proof points to a general technique for augmenting networks with a distributed termination algorithm in a way as not to introduce deadlock. Suppose $T = \langle Q_1, \ldots, Q_n \rangle$ is a network that is guaranteed to terminate (so it is deadlock free) and $V = \langle P_1, \ldots, P_n \rangle$ is the network to which this "termination algorithm" is to be added, where $\alpha T \cap \alpha V = \emptyset$. Then again from Lemma 4.3.1 (v) the network $\langle P_1 \stackrel{\Delta}{} Q_1, \ldots, P_n \stackrel{\Delta}{} Q_n \rangle$ is deadlock free. Clearly we must still prove that this network does terminate under the appropriate conditions and in general this may require imposing some fairness constraint.

In the network consisting of the processes $P_{nr} \stackrel{\Delta}{} Q_{nr}$, the network is guaranteed to terminate after the occurrence of the event *shutdown* for the following reason. The processes P_{nr} $(P_{nr} \neq P_{00})$ can proceed infinitely only if the process P_{00} does and hence after the event *shutdown* the only progress that can eventually be made by the network is within the processes Q_{nr}. But as the network is deadlock free and these processes Q_{nr} are guaranteed to terminate, the whole network terminates. □

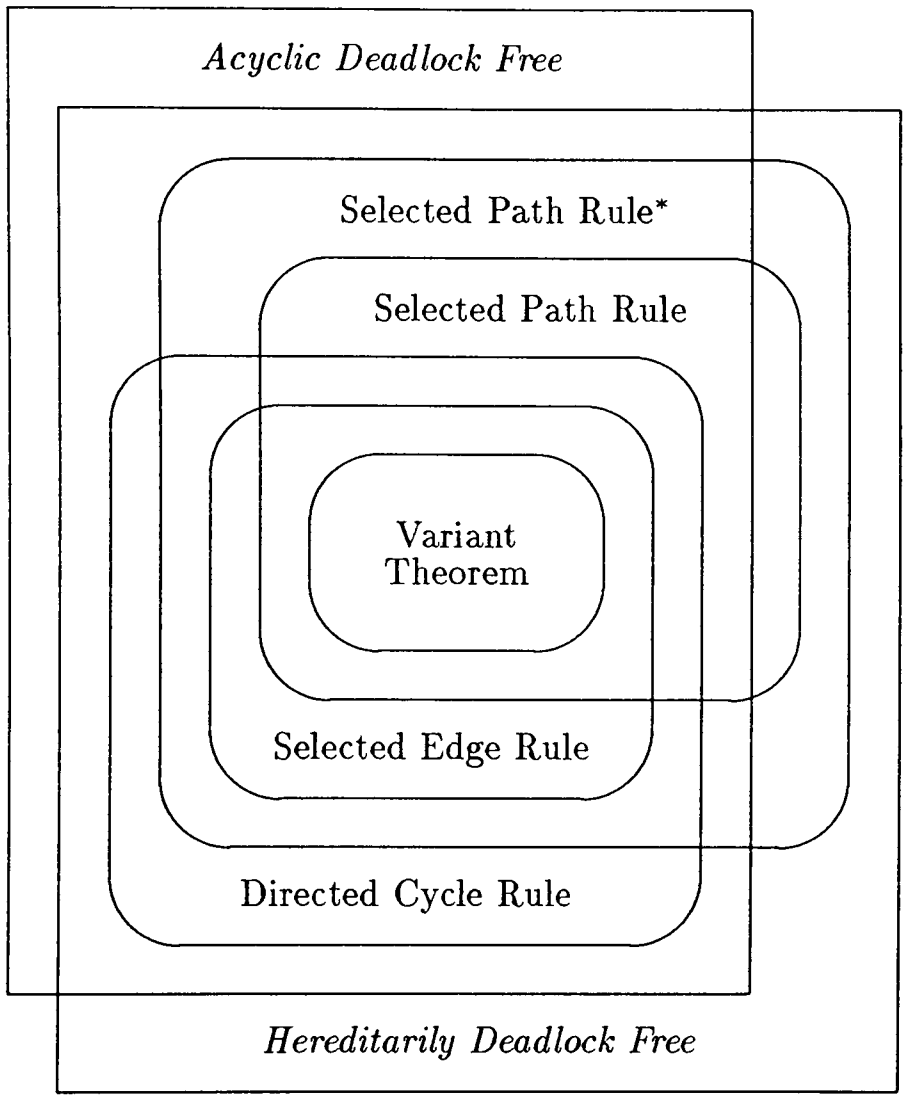

Figure 3.4: Scope of the Rules

Chapter 4

Non-local Deadlock Freedom

We have already observed that our deadlock freedom Rules can never prove the deadlock freedom of non-hereditarily deadlock free networks. In a network which is deadlock free but not hereditarily so, there is some subnetwork which is prevented from entering a deadlock state by the restraints imposed by one or more of the other processes in the network. It is clear that in such cases local checking alone will be inadequate for proofs of deadlock freedom.

In the first section we introduce a framework in which to describe the standard technique of using an invariant for global reasoning about deadlock freedom.

As global reasoning can be cumbersome, we can attempt to make it more manageable by only dealing with the information which is required specifically for proving deadlock freedom. With this aim in mind, in the second section we shall integrate a modified concept of an invariant into our deadlock freedom Rules so allowing us to deal with non-hereditarily deadlock free networks. We also prove some completeness results that ensue from this extension of the Rules.

However, there are networks which are hereditarily deadlock free but are not amenable to our original Rules, although we have not met many of these in practice. The deadlock freedom of most of these networks is still intrinsically a consequence of global behaviour, and thus we again need the full power of non-local Rules to deal with them. The first variation in Example 3.1.1 is such a network. It is desirable that networks which are not in this latter class, but hereditarily deadlock free, should be dealt with by essentially local methods. In the final section we see how trivial transformations which preserve the deadlock behaviour of networks can sometimes

coax networks into a form to which our local Rules can be applied, as well as make deadlock analysis in general more tractable.

4.1 Standard Global Reasoning

As mentioned earlier, when other authors have addressed the problem of proving deadlock freedom, they have tended to describe methods which are essentially global. Given that they were generally looking for complete methods (so that non-hereditarily deadlock free networks could also be treated), this is understandable. We have already discussed one such method, that of [CM1], in Chapter 2, and we now briefly rigorise those of [LG, OG] and [AFR] in terms of the failures model. Both these latter methods rely on proving that the "blocking" of all the processes in a network is not a possibility. It is easier to present these methods in their dual version, that is via the definition of a "non-blocked" process.

Definition 4.1.1 Let $V = \langle P_1, \ldots, P_n \rangle$ be a triple-disjoint network with vocabulary Λ. We say P_i has a *granted request* to P_j in a state σ of a network containing $\langle P_i, P_j \rangle$, if $i \neq j$ and

$$(\alpha P_i - X_i) \cap (\alpha P_j - X_j) \neq \emptyset$$

where $\sigma \upharpoonright \langle P_i, P_j \rangle = \langle s, \langle X_i, X_j \rangle \rangle$. We write $P_i \xleftrightarrow{\sigma} P_j$ when this is the case. We shall also write $P_i \xleftrightarrow{\sigma} *$ when $\sigma = (s_i, X_i) \in \tilde{\mathcal{F}}[\![P_i]\!]$ with $\alpha P_i - X_i \not\subseteq \Lambda$.

So a non-blocked process is one that has a granted request to another process or is willing to communicate outside the vocabulary of the network. The result central to the validity of the two methods above then takes the following form.

Theorem 4.1.1 A triple disjoint network $V = \langle P_1, \ldots, P_n \rangle$ is deadlock free if, and only if, for every state σ of the network we have

$$\exists P, Q : \{*, P_1, \ldots, P_n\}. P \xleftrightarrow{\sigma} Q \quad .$$

Proof. Follows easily from Definition 1.2.3. □

A proof of deadlock freedom then proceeds by the construction of an invariant (a statement which holds true in every global state of the network in question) that implies the existence of a non-blocked process in a state in which it holds (this implicate being described formally by the expression displayed above).

4.2 Invariants and Variants

The next three results display the form some of our Rules take when invariants are incorporated into them. The proofs of these results are similar to those for them in their original form.

Theorem 4.2.1 (Global Variant Theorem) Let $V = \langle P_1, \ldots, P_n \rangle$ be a busy, triple-disjoint network with vocabulary Λ. Suppose that there exist functions

$$f_i : \tilde{\mathcal{F}}[\![P_i]\!] \times T \to \Pi \qquad (1 \leq i \leq n)$$

where $(\Pi, >)$ is a partial order, and an *invariant* $\Im$ which is a predicate such that

$$\forall \sigma : \mathcal{S}(V); \exists \Delta : T. \Im(\sigma, \Delta)$$

holds. Furthermore, whenever σ is a state of V with $\Im(\sigma, \Delta)$, we have

$$P_i \xrightarrow{\sigma\restriction\langle P_i, P_j\rangle, \Lambda} \bullet P_j \;\Rightarrow\; f_i(\sigma\restriction\alpha P_i, \Delta) > f_j(\sigma\restriction\alpha P_j, \Delta)$$

for any subnetwork $\langle P_i, P_j \rangle$. Then V is strongly acyclic deadlock free.

Theorem 4.2.2 (Global Directed Cycle Rule) Let $V = \langle P_1, \ldots, P_n \rangle$ be a triple-disjoint, busy and strong conflict free network with vocabulary Λ. Let D be the set of directed cycles in the communication graph of V. Suppose there exist functions

$$f_i : \tilde{\mathcal{F}}[\![P_i]\!] \times D \times T \longrightarrow \Pi \qquad (1 \leq i \leq n)$$

where $(\Pi, >)$ is a partial order, and an invariant $\Im$ such that

$$\forall \sigma : \mathcal{S}(V); \exists \Delta : T. \Im(\sigma, \Delta)$$

holds. Furthermore, whenever σ is a state of V with $\Im(\sigma, \Delta)$ and $\langle i, j \rangle$ is an edge in a directed cycle $d \in D$, we have

$$P_i \xrightarrow{\sigma\restriction\langle P_i, P_j\rangle, \Lambda} \bullet P_j \;\Rightarrow\; f_i(\sigma\restriction\alpha P_i, d, \Delta) > f_j(\sigma\restriction\alpha P_j, d, \Delta)$$

for any subnetwork $\langle P_i, P_j \rangle$. Then V is acyclic deadlock free.

Theorem 4.2.3 (Global Selected Path Rule) Let $V = \langle P_1, \ldots, P_n \rangle$ be a triple-disjoint, busy network with vocabulary Λ. Suppose that there exist functions

$$\begin{array}{l} f_i : \tilde{\mathcal{F}}[\![P_i]\!] \times T \to \Pi \\ g_i : \tilde{\mathcal{F}}[\![P_i]\!] \times T \to \{1, \ldots, n\} \end{array} \qquad (1 \leq i \leq n)$$

where $(\Pi, >)$ is a partial order, and an invariant $\Im$ such that

$$\forall \sigma : \mathcal{S}(V); \exists \Delta : T. \Im(\sigma, \Delta)$$

holds. Furthermore, whenever σ is a state of V with $\Im(\sigma, \Delta)$, we have for any subnetwork $\langle P_i, P_j \rangle$

$$P_i \xrightarrow{\sigma\restriction\langle P_i, P_j\rangle, \Lambda} \bullet P_j \quad \Rightarrow \quad P_i \xrightarrow{\sigma\restriction\alpha P_i} P_{g_i(\sigma\restriction\alpha P_i, \Delta)}$$

and if $j = g_i(\sigma\restriction\alpha P_i, \Delta)$,

$$P_i \xrightarrow{\sigma\restriction\langle P_i, P_j\rangle, \Lambda} \bullet P_j \quad \Rightarrow \quad f_i(\sigma\restriction\alpha P_i, \Delta) > f_j(\sigma\restriction\alpha P_j, \Delta)\,.$$

Then V is CM-deadlock free.

An invariant $\Im(\sigma, \Delta)$ in our sense is not an invariant in the sense of the authors mentioned above, although $\exists \Delta : T. \Im(\sigma, \Delta)$ is. The reason for this difference is to facilitate our theme of keeping to local analysis as far as possible. In a network which is non-hereditarily deadlock free, it is often not too difficult to identify the property which makes it deadlock free even though a subnetwork is not. Δ in an invariant then represents this property which has been extracted from a global state by the invariant. Thus rather than consulting global states directly in the construction of variant functions, we instead use Δ and then only when information yielded by local states does not suffice.

We now apply this technique to a well-known network.

Example 4.2.1 (Dining Philosophers with Butler [H1]) Let n be a constant with $n \geq 1$. Then if arithmetic is modulo $n + 1$, the processes are defined as follows:

$$\begin{array}{rcl} BUTLER & = & B_0 \\ \alpha BUTLER & = & \{0.sits, 0.leaves, \ldots, n.sits, n.leaves\} \end{array}$$

$$
\begin{aligned}
FORK_i &= \Box \begin{array}{l} i.picks.i \rightarrow i.puts.i \rightarrow FORK_i \\ i-1.picks.i \rightarrow i-1.puts.i \rightarrow FORK_i \end{array} \\
\alpha FORK_i &= \{i.picks.i, i.puts.i, i-1.picks.i, i-1.puts.i\} \\
\\
PHIL_i &= \begin{array}{l} i.sits \rightarrow i.picks.i \rightarrow i.picks.i+1 \rightarrow i.eats \\ \quad \rightarrow i.puts.i \rightarrow i.puts.i+1 \rightarrow i.leaves \\ \quad \rightarrow PHIL_i \end{array} \\
\alpha PHIL_i &= \{i.sits, i.picks.i, i.picks.i+1, i.eats, \\
&\qquad i.puts.i, i.puts.i+1, i.leaves\}
\end{aligned}
$$

where

$$B_0 = \Box_{0 \le i \le n} \; i.sits \rightarrow B_1$$

$$B_j = (\Box_{0 \le i \le n} \; i.sits \rightarrow B_{j+1}) \Box (\Box_{0 \le i \le n} \; i.leaves \rightarrow B_{j-1}) \qquad (0 < j < n)$$

$$B_n = \Box_{0 \le i \le n} \; i.leaves \rightarrow B_{n-1}$$

and $0 < i < n$. The network

$$\langle FORK_0, PHIL_0, \ldots, FORK_n, PHIL_n, BUTLER \rangle$$

is triple-disjoint and busy. The subnetwork consisting just of philosophers and forks has an oft-quoted example of a deadlock state (corresponding to the trace $\langle 0.sits, \ldots, n.sits, 0.picks.0, \ldots, n.picks.n \rangle$) and thus the network is not hereditarily deadlock free. The addition of the *BUTLER* is what prohibits such behaviour by preventing the seating of all the philosophers at any one time. The invariant we shall use

$$\Im(\langle s, \underline{X} \rangle, \Delta) \mathrel{\widehat{=}} (0 \le \Delta \le n) \wedge (s\restriction\{\Delta.sits\} = s\restriction\{\Delta.leaves\})$$

utilises this fact to extract an unseated philosopher $PHIL_\Delta$ from any state of the network. Now let L denote the set of labels of processes in the network, that is $\{FORK_0, PHIL_0, \ldots, FORK_n, PHIL_n, BUTLER\}$, and define functions

$$\begin{aligned} f_P &: \tilde{\mathcal{F}}[\![P]\!] \times \{0, \ldots, n\} \rightarrow \mathbf{N} \\ g_P &: \tilde{\mathcal{F}}[\![P]\!] \times \{0, \ldots, n\} \rightarrow L \end{aligned} \qquad (P \in L)$$

by putting

$$f_{PHIL_i}((s,X),\Delta) =$$

$$2\left\{\begin{array}{ll} (\Delta - i) \bmod (n+1) & \text{if } \alpha PHIL_i - X = \{i.picks.i+1\} \\ n+1 & \text{if } \alpha PHIL_i - X = \{i.sits\} \\ 1 & \text{if } \alpha PHIL_i - X = \{i.picks.i\} \\ 0 & \text{otherwise} \end{array}\right\}$$

$$f_{FORK_i}((s,X),\Delta) =$$

$$2\left\{\begin{array}{ll} (\Delta - i) \bmod (n+1) & \text{if } \alpha FORK_i - X = \{i.puts.i\} \\ n+1 & \text{if } \alpha FORK_i - X \supseteq \{i.picks.i\} \\ 0 & \text{if } \alpha FORK_i - X = \{i-1.puts.i\} \end{array}\right\} + 1$$

$$f_{BUTLER}((s,X),\Delta) = 2n+1$$

$$g_{PHIL_i}(\sigma,\Delta) \;=\; \left\{\begin{array}{ll} P & \text{if } PHIL_i \xrightarrow{\sigma} P \\ PHIL_i & \text{otherwise} \end{array}\right.$$

$$g_{FORK_i}(\sigma,\Delta) \;=\; \left\{\begin{array}{ll} PHIL_i & \text{if } FORK_i \xrightarrow{\sigma} PHIL_i \\ PHIL_{i-1} & \text{otherwise} \end{array}\right.$$

$$g_{BUTLER}(\sigma,\Delta) \;=\; PHIL_i \text{ where } i \neq \Delta$$

where N is ordered by $>$ in the usual way. Hence the network is deadlock free by the Global Selected Path Rule.

It is interesting to note that the network is not even strong conflict free. The strong conflict occurs when the butler has seated all but one philosopher. In this state the butler is insisting that the latter philosopher leave despite the fact that he is not seated. As has been pointed out in [BR3], this behaviour can be considered a design fault of the network and a trivial redesign of the *BUTLER* process results in the network becoming conflict free. With this amendment, the network can also then be treated by the Global Variant Theorem. □

The first variation of Example 3.1.1 can be dealt with by the Global Directed Cycle or the Global Selected Path Rule with an invariant $\Im(\sigma,\Delta)$

which asserts that there is a process index Δ such that P_Δ and $P_{\Delta+1}$ are the same "colour" in σ. In this straightforward case however, using the corresponding classical invariant with Theorem 4.1.1 is simpler.

We noted earlier that the global proof methods of other authors have tended to be complete. We now show that our extended Rules too are complete with respect to various types of deadlock freedom.

Theorem 4.2.4 A triple-disjoint, busy network is strongly acyclic deadlock free if, and only if, it can be proved deadlock free by the Global Variant Theorem.

Proof. Theorem 1 has dealt with one direction of the proof already and so we need only prove the converse. Hence suppose $V = \langle P_1, \ldots, P_n \rangle$ is a triple-disjoint, busy and strongly acyclic deadlock free network with vocabulary Λ. Let $\Phi = \mathcal{S}(V) \times \{1, \ldots, n\}$ and define the relation $\succ_\Phi$ on Φ as follows:

$$(\sigma, i) \succ_\Phi (\sigma', j) \quad \Leftrightarrow \quad \sigma = \sigma' \text{ and } P_i \xrightarrow{\sigma,\Lambda}\bullet\, P_j \,.$$

Thus if $\rhd_\Phi$ is the transitive closure of $\succ_\Phi$ on Φ, it follows that $(\Phi, \rhd_\Phi)$ is a partial order, as V is strongly acyclic deadlock free. Then defining variant functions

$$f_i : \tilde{\mathcal{F}}[\![P_i]\!] \times \mathcal{S}(V) \to \Phi \qquad (1 \leq i \leq n)$$

and an invariant $\Im$ by putting $f_i(\sigma', \sigma) = (\sigma, i)$ and $\Im(\sigma, \Delta) \mathrel{\widehat{=}} \sigma = \Delta$, we see that V satisfies the conditions of the Global Variant Theorem. □

Theorem 4.2.5 A triple-disjoint, busy and strong conflict free network is acyclic deadlock free if, and only if, it can be proved deadlock free by the Global Directed Cycle Rule.

Proof. We need only prove the converse to Theorem 2 and so we suppose that $V = \langle P_1, \ldots, P_n \rangle$ is a triple-disjoint, busy and strong conflict free network with vocabulary Λ. Let D be the set of directed cycles in the communication graph, and let Φ and $\succ_\Phi$ be as defined in in the proof of Theorem 4. Now put $\Phi' = \Phi \times D$ and define the relation $\succ'_\Phi$ on Φ' as follows:

$$((\sigma_i, i), d_i) \succ'_\Phi ((\sigma_j, j), d_j) \quad \Leftrightarrow \quad \begin{cases} d_i = d_j \\ \langle i, j \rangle \text{ is in } d_i \\ \text{and } (\sigma_i, i) \succ_\Phi (\sigma_j, j). \end{cases}$$

The transitive closure of $\succ'_\Phi$ on Φ' is clearly a partial order as V is acyclic deadlock free. Then defining variant functions

$$f_i : \tilde{\mathcal{F}}[\![P_i]\!] \times D \times \mathcal{S}(V) \to \Phi \qquad (1 \leq i \leq n)$$

and an invariant $\Im$ by putting $f_i(\sigma', d, \sigma) = ((\sigma, i), d)$ and $\Im(\sigma, \Delta) \mathrel{\widehat{=}} \sigma = \Delta$, we see that V satisfies the conditions of the Global Directed Cycle Theorem. □

Theorem 4.2.6 A triple-disjoint, busy network is CM-deadlock free if, and only if, it can be proved deadlock free by the Global Selected Path Rule.

Proof. It only remains to prove the converse of Theorem 3, and so we suppose that $V = \langle P_1, \ldots, P_n \rangle$ is a triple-disjoint, busy and CM-deadlock free network with vocabulary Λ. Let Φ, $\succ_\Phi$ and $\rhd_\Phi$ be as defined in the proof of Theorem 4. Note that $\rhd_\Phi$ is a preorder and not necessarily a partial order in this case. Consider an equivalence class $\mathcal{E}$ in the partial order $(\Phi/\sim, \rhd_\Phi/\sim)$ where

$$(\sigma, i) \sim (\sigma', j) \;\Leftrightarrow\; (\sigma, i) \trianglerighteq_\Phi (\sigma', j) \text{ and } (\sigma', j) \trianglerighteq_\Phi (\sigma, i)$$

is the usual preorder equivalence. We assert that there is a $(\sigma, k) \in \mathcal{E}$ such that $\exists j : \{*, 1, \ldots, n\}.\, P_k \xleftrightarrow{\sigma} P_j$ or there exists $(\sigma, i) \notin \mathcal{E}$ with $P_k \xrightarrow{\sigma} P_i$. Assume for the sake of contradiction that this is not true. Then we see that the state resulting from the restriction of σ to the subnetwork identified with $\mathcal{E}$ is a deadlock state. But this contradicts the CM-deadlock freedom of V via Lemma 2.1.1. Let (σ, k) be an element of $\mathcal{E}$ whose existence is guaranteed by the assertion. Define relations $\gg^r_\mathcal{E}$ $(r \in \mathrm{N})$ on $\mathcal{E}$ inductively as follows:

$$(\sigma, i) \gg^0_\mathcal{E} (\sigma, j) \;\Leftrightarrow\; j = k \text{ and } P_i \xrightarrow{\sigma} P_j$$

$$(\sigma, i) \gg^{r+1}_\mathcal{E} (\sigma, j) \;\Leftrightarrow\; \begin{cases} \text{either } (\sigma, j) \in dom(\gg^r_\mathcal{E}),\ (\sigma, i) \notin dom(\gg^r_\mathcal{E}), \\ P_i \xrightarrow{\sigma} P_j \text{ and } j \text{ is the smallest such process} \\ \text{index or } (\sigma, i) \gg^r_\mathcal{E} (\sigma, j). \end{cases}$$

As $\mathcal{E}$ is finite, there is a $m \in \mathrm{N}$ such that $\gg^m_\mathcal{E} = \gg^j_\mathcal{E}$ for all $j \geq m$. We shall simply write $\gg_\mathcal{E}$ for $\gg^m_\mathcal{E}$. Easy arguments by induction show that $\gg_\mathcal{E}$ is a partial order and that if $(\sigma, i) \in dom(\gg_\mathcal{E})$ then there is a unique $(\sigma, j) \in \mathcal{E}$ such that $(\sigma, i) \gg_\mathcal{E} (\sigma, j)$. We now prove that $\mathcal{E} = dom(\gg_\mathcal{E}) \cup \{(\sigma, k)\}$ and

so we assume for the sake of contradiction that this is not so. Then there is a $(\sigma, i) \in \mathcal{E} - [dom(\gg_\mathcal{E}) \cup \{(\sigma, k)\}]$ and hence a sequence

$$(\sigma, i) \succ_\Phi (\sigma, j) \succ_\Phi \ldots \succ_\Phi (\sigma, k)$$

as $(\sigma, i) \sim (\sigma, k)$. We may assume without loss of generality that (σ, i) is an element with the shortest such sequence. But then $(\sigma, j) \in dom(\gg_\mathcal{E})$ or $j = k$. In the first case we would have $(\sigma, i) \in dom(\gg_\mathcal{E}^{m+1})$, contradicting the fact that $\gg_\mathcal{E}^m = \gg_\mathcal{E}^{m+1}$; and in the second case $(\sigma, i) \in dom(\gg_\mathcal{E}^0)$ contradicting the fact that $(\sigma, i) \notin dom(\gg_\mathcal{E})$.

Consider the function $\Psi : \Phi \to \{1, \ldots, n\}$ defined by putting

$$\Psi((\sigma, i)) = \begin{cases} & i & \text{if } P_i \xleftrightarrow{\sigma} P_* \\ \text{if not} & j & \text{if } P_i \xleftrightarrow{\sigma} P_j \text{ and } j \text{ is the smallest such index} \\ \text{if not} & j & \text{if } P_i \xrightarrow{\sigma} P_j,\ (\sigma, i) \not\sim (\sigma, j) \text{ and } j \text{ is the smallest such index} \\ \text{if not} & j & \text{if } P_i \xrightarrow{\sigma} P_j,\ (\sigma, i) \sim (\sigma, j) \text{ and } j \text{ is the unique index such that } (\sigma, i) \gg_{[(\sigma,i)]_\sim} (\sigma, j) \end{cases}$$

By the busy-ness of V and the orders defined on equivalence classes, we note that Ψ is well-defined and total.

We now define a relation $\succ_\Psi$ on Φ as follows:

$$(\sigma, i) \succ_\Psi (\sigma', j) \quad \Leftrightarrow \quad \Psi((\sigma, i)) = j \text{ and } (\sigma, i) \succ_\Phi (\sigma', j)$$

We assert that the transitive closure of $\succ_\Psi$ on Φ is a partial order. Assume for the sake of contradiction that this is not so. This can only be if there is a cycle

$$(\sigma, i_0) \succ_\Psi (\sigma, i_1) \succ_\Psi \ldots \succ_\Psi (\sigma, i_0)$$

Not all the elements in the cycle can be in the same equivalence class $\mathcal{E}$ as otherwise, by the definition of Ψ, we would be contradicting the fact that $\gg_\mathcal{E}$ is a partial order. However, if all the elements are not in the same equivalence class, the cycle above corresponds to a cycle of equivalence classes ordered by $\rhd_\Phi / \sim$, which is again a contradiction. Thus the transitive closure of $\succ_\Psi$ on Φ is a partial order.

Next define functions

$$\begin{aligned} f_i &: \tilde{\mathcal{F}}[\![P_i]\!] \times \mathcal{S}(V) \to \Phi \\ g_i &: \tilde{\mathcal{F}}[\![P_i]\!] \times \mathcal{S}(V) \to \{1, \ldots, n\} \end{aligned} \qquad (1 \leq i \leq n)$$

by putting

$$f_i(\sigma', \sigma) = (\sigma, i)$$
$$g_i(\sigma', \sigma) = \Psi((\sigma, i))$$

and further let $\Im(\sigma, \Delta) \mathrel{\widehat{=}} \sigma = \Delta$. Finally, ordering Φ by the transitive closure of $\succ_\Psi$ and using the definitions above, it is a straightforward matter to verify that V satisfies the Global Selected Path Rule. □

From the discussion of the types of deadlock freedom in Chapter 2, we see that the most general of these global methods is the Selected Path Rule. We also recall that because of Theorem 2.1.2, networks that can be proved deadlock free by the Global Selected Path Rule also have priority functions. In general, the relationship between the functions yielded by the two methods is not as straightforward as in the alternative proof of Corollary 2.3.2. This is not surprising considering the fact that the two proof rules have been derived from two essentially different methodologies of proving deadlock freedom.

We have omitted the global versions of the Selected Edge Rule and Selected Path Rule* as they do not extend the range of networks that can already be treated by the Rules above. The Global Selected Path Rule* is complete for CM-deadlock free networks, but there is no completeness result for the Global Selected Edge Rule for any of the types of deadlock freedom. This is because of the static nature of the set of directed edges required by the Rule. However, for certain cases these two Rules may be useful, by allowing us to develop variant functions separately for essential components and in the case of the Global Selected Edge Rule by simplifying variant functions, because strict inequality is not always required.

We observe that a network that can be dealt with by the Global Variant Theorem need not be conflict free unlike with its local counterpart. However, such a network does satisfy the requirement that every state of it contains no conflict state. This is the global version of conflict freedom. Similar analogues of busy and strong conflict freedom are then also apparent. We formalise these properties of networks in the following definition.

Definition 4.2.1 Let $V = \langle P_1, \ldots, P_n \rangle$ be a triple-disjoint network. We say V is *globally busy* if for any state σ of V, $\sigma{\restriction}\alpha P_i$ is never a deadlock state of P_i for any P_i in the network. We say V is *globally conflict free* (respectively *globally strong conflict free*) if for any state σ of V, $\sigma{\restriction}\langle P_i, P_j \rangle$ is never a conflict (respectively strong conflict) state of $\langle P_i, P_j \rangle$ for any subnetwork $\langle P_i, P_j \rangle$.

In practice, we prove these properties of networks again by invariants. For some examples of this, see [RD]. The next result (whose proof is obvious) relaxes the preconditions of our global Rules that are purely local in order to permit global satisfaction of them.

Lemma 4.2.7 The replacement of busy and strong conflict freedom by their global versions in the Rules above still preserves the validity of the Rules.

Clearly, the completeness results for the Rules may be extended in a similar way and the definition of a prudent network, when globalised, also generalises the relevant results. Even though the definitions in Definition 1 generalise our Rules slightly, it is a mistake to rely on them regularly. It is better to design elements of a network so that their local behaviour is good rather than to prove this by global analysis of the network. Some methods for such a correct (re-)construction of a network are given in [RD].

4.3 Deadlock Transformations

We first define a notion of network transformation that is useful for the purpose of deadlock analysis.

Definition 4.3.1 For networks $V, V_1, \ldots, V_n$ we say $V_1, \ldots, V_n$ is a *deadlock transformation* of V if for every deadlock state of V there is a deadlock state of some V_i. In this case we shall write $V_1, \ldots, V_n \vdash_{\mathrm{df}} V$.

Thus if $V_1, \ldots, V_n \vdash_{\mathrm{df}} V$, then the deadlock freedom of all the networks on the left hand side entails the deadlock freedom of V. The next result lists some trivial transformations which are local in nature and that are commonly used to prepare a network for the application of some deadlock freedom proof Rule.

Lemma 4.3.1 Let $V = \langle P_1, \ldots, P_n \rangle$ be a triple-disjoint network with vocabulary Λ.

(i) If $P_i \sqsubseteq Q_i$ for $1 \leq i \leq n$, then $\langle Q_1, \ldots, Q_n \rangle \vdash_{\mathrm{df}} V$.

(ii) Then $\langle P_1 \backslash (\alpha P_1 - \Lambda), P_2, \ldots, P_n \rangle \vdash_{\mathrm{df}} V$. Further, if $P_1 \backslash (\alpha P_1 - \Lambda)$ is divergence free then $V \vdash_{\mathrm{df}} \langle P_1 \backslash (\alpha P_1 - \Lambda), P_2, \ldots, P_n \rangle$.

(iii) If V has subnetworks $V_1, \ldots, V_k$ corresponding to essential components, and the two element subnetworks corresponding to disconnecting edges are conflict free, then $V_1, \ldots, V_k \vdash_{\mathrm{df}} V$.

(iv) If $P_1 = Q_1 \sqcap R_1$ then $\langle Q_1, P_2, \ldots, P_n\rangle, \langle R_1, P_2, \ldots, P_n\rangle \vdash_{\mathrm{df}} V$.

(v) If W is a network $\langle Q_1, \ldots, Q_n\rangle$ such that we have $\alpha V \cap \alpha W = \emptyset$ then

$$W \vdash_{\mathrm{df}} \langle P_1 \| Q_1, \ldots, P_n \| Q_n\rangle, \langle P_1 \stackrel{\Delta}{} Q_1, \ldots, P_n \stackrel{\Delta}{} Q_n\rangle$$

and $\langle P_1 \stackrel{\Delta}{} Q_1, \ldots, P_n \stackrel{\Delta}{} Q_n\rangle \vdash_{\mathrm{df}} W$.

Proof.

(i) $PAR(\langle Q_1, \ldots, Q_n\rangle) \sqsubseteq PAR(V)$ and so every state of V is a state of $\langle Q_1, \ldots, Q_n\rangle$, including deadlock states.

(ii) Follows easily from Lemmas 1.1.2 and 5.1.1.

(iii) A re-statement of Lemma 1.3.4.

(iv) $PAR(\langle Q_1, P_2, \ldots, P_n\rangle) \sqcap PAR(\langle R_1, P_2, \ldots, P_n\rangle) = PAR(V)$ and so a deadlock state of V must be a deadlock state of $\langle Q_1, P_2, \ldots, P_n\rangle$ or $\langle R_1, P_2, \ldots, P_n\rangle$.

(v) Follows easily from the failures semantics of the operators.

□

Although the use of (i) may introduce deadlock into a transformed network, used carefully it serves to simplify networks in which processes make complex deterministic choices based on their traces; by replacing these choices with non-deterministic ones. Surprisingly often, a network is robust enough to withstand this transformation without introducing new deadlock states such as the network in Example 2.3.1 and other networks in [RD]. Removing or introducing the hiding of internal events of processes in a network, whichever facilitates the deadlock analysis of the particular network in question, can be carried out via (ii). The use of (v) in augmenting a network without introducing deadlock was given in Example 3.6.2. Our next example shows an application of (iv).

Example 4.3.1 Define a process

$$Q_0 = \begin{array}{ll} \Box & 0.to.1.black \rightarrow Q_0 \\ \Box & 1.to.0.white \rightarrow Q_0 \\ \Box & 0.to.n-1.white \rightarrow Q_0 \\ \Box & n-1.to.0.black \rightarrow Q_0 \end{array}$$

$$\alpha Q_0 = \{0.to.1.black, 1.to.0.white, 0.to.n-1.white, n-1.to.0.black\}$$

and let processes P_i for $0 \leq i < n$ be as defined in Example 3.3.1. Now put $P_0' = P_0 \sqcap Q_0$ with $\alpha P_0' = \alpha P_0 \cup \alpha Q_0$ in the network $V = \langle P_0', P_1, \ldots, P_{n-1} \rangle$,where $n \geq 3$. If σ_0^P and σ_0^Q are the initial states of P_0 and Q_0 respectively, then we see that there are cycles

$$(\sigma_0^P, 0) \succ (\sigma_1, 1) \succ \cdots (\sigma_{n-1}, n-1) \succ (\sigma_0^Q, 0) \succ$$

$$(\sigma_1', 1) \succ \cdots \succ (\sigma_{n-1}', n-1) \succ (\sigma_0^P, 0)$$

and

$$(\sigma_0^P, 0) \succ (\sigma_{n-1}', n-1) \succ \cdots (\sigma_1', 1) \succ (\sigma_0^Q, 0) \succ$$

$$(\sigma_{n-1}, n-1) \succ \cdots \succ (\sigma_1, 1) \succ (\sigma_0^P, 0)$$

where $\succ$ is defined as in Theorem 2.3.3. Then by Theorem 3.5.1, the Directed Cycle Rule cannot be applied to V and in addition, as the variant functions for the Selected Path Rule would have to decrease along one of the two cycles above, neither can the Selected Path Rule.

Now Lemma 4.3.1 (iv) yields that

$$\langle P_0, P_1, \ldots, P_{n-1} \rangle, \langle Q_0, P_1, \ldots, P_{n-1} \rangle \vdash_{\mathrm{df}} V$$

but $\langle P_0, \ldots, P_{n-1} \rangle$ was proved deadlock free in Example 3.5.1 and by symmetry, swapping *black* and *white* in process definitions, $\langle Q_0, P_1, \ldots, P_{n-1} \rangle$ is also deadlock free, and thus we conclude that V is deadlock free. We note that V is then acyclic and hereditarily deadlock free, by Lemma 2.2.4. □

We now present a less obvious example of a transformation.

Example 4.3.2 ([R4]) The network is a message passing ring in which a number of users can send mail to one another. Each user is associated with a node and the nodes are connected as a ring. A node may accept a message from its user and pass it to its clockwise neighbour, or accept a message from its anticlockwise neighbour and give it to its own user or pass it clockwise as appropriate. Each node has the capacity to store two messages and will only accept a message from its own user when it has no messages still left to be passed on. Ignoring the details of messages, the node processes may be defined as follows:

$$P_i = \Box \begin{array}{l} i.in \rightarrow Q_i \\ i+1 \rightarrow [(i.out \rightarrow P_i) \sqcap Q_i] \end{array} \qquad (0 \leq i \leq n)$$

$$\alpha P_i = \{i.in, i.out, i, i+1\}$$

where

$$Q_i = \Box \begin{array}{l} i+1 \to [(i.out \to Q_i) \sqcap (i \to Q_i)] \\ i \to P_i \end{array}$$

and all arithmetic is modulo $n+1$. The network $V = \langle P_0, \ldots, P_n \rangle$ is triple-disjoint, busy and conflict free. Now let σ_i denote the state of P_i where P_i is only willing to communicate with P_{i-1}, having already received two messages from its own user (the first of which was passed on) and a message from P_{i+1} to pass on. That is $\sigma_i = (\langle i.in, i, i.in, i+1 \rangle, \{i.in, i.out, i+1\})$. Then we see that there is a cycle

$$(\sigma_n, n) \succ (\sigma_{n-1}, n-1) \succ \cdots \succ (\sigma_0, 0) \succ (\sigma_n, n)$$

which prevents the use of the Directed Cycle Rule or the Selected Path Rule.

Now define processes

$$P_i'(x) = \Box \begin{array}{l} i.in \to Q_i'(x) \\ i+1.x \to [(i.out \to P_i'(x)) \sqcap Q_i'(x)] \end{array} \qquad \left(\begin{array}{c} 0 \le i \le n \\ x \in \mathbf{N} \end{array} \right)$$

$$\alpha P_i'(x) = \{i.in, i.out\} \cup \{i, i+1\}.\mathbf{N}$$

where

$$Q_i'(x) = \Box \begin{array}{l} i+1.x \to [(i.out \to Q_i'(x)) \sqcap (i?y \to Q_i'(y+1))] \\ i?y \to P_i'(y+1) \end{array}$$

and consider the network $V' = \langle P_0'(0), \ldots, P_n'(0) \rangle$. V' is triple-disjoint, busy, conflict free and moreover, as each $P_i'(0)$ has the same communication pattern as P_i, in the sense that we have just labelled some communications, we see that $V' \vdash_{\mathrm{df}} V$. Then using variant functions

$$f_i : \tilde{\mathcal{F}}[\![P_i]\!] \to \mathbf{N} \qquad (0 \le i \le n)$$

defined by putting

$$f_i((s, X)) = \begin{cases} 0 & \text{if } s\restriction i.\mathbf{N} = \langle\rangle \\ y+1 & \text{if } i.y \text{ is the last element in } s\restriction i.\mathbf{N} \end{cases}$$

and ordering $\mathbf{N}$ by $>$ in the usual way, it is straightforward to verify that V' is deadlock free by the Variant Theorem. Thus we conclude that V is deadlock free, and hence acyclic and hereditarily deadlock free by Lemma 2.2.4. $\Box$

The following example generalises the latter and is based on results which may be found in [R4].

Example 4.3.3 Let V be a triple-disjoint, busy message passing network such that whenever a process P has an ungranted request to a second process Q, Q has not passed on a message to a process since P last passed on a message to it. Then V is deadlock free. □

Proof. Note that V is conflict free. We now use a similar ruse as in the last example by altering each process in the following way:

- Each process keeps a local counter which is zeroised initially.
- When a message is passed on by a process, the value of the counter of the receiving process is simultaneously obtained and its own counter is set to the maximum of this value and the current value of its own counter and then incremented.

The resulting network is a deadlock transformation of V. Observe that the value of a counter changes only when a message is passed on and then it strictly increases. Thus, using the values of these counters as variant functions allows us to prove the latter network deadlock free by the Variant Theorem, and hence V is deadlock free. □

In the last two examples, the transformations have enabled the use of the Variant Theorem by providing the traces of processes sufficient information about neighbours. This is because the relation $\succ$ for the networks is in some sense "refined" by the transformations.

Chapter 5

Checking Implementations

In this chapter we develop methods which allow us to check that a specification is met by an implementation. As there are many notions of what a specification is and what its relation to a proposed implementation should be, we define and motivate our notions of these concepts in the first section.

Much of the existing work in this field has been termed *proving correctness*. In general terms, the strength of a correctness condition is decided according to whether it addresses the issues of liveness (divergence freedom) and deadlock freedom or not. If it does not, it is said to be *partial* or *weak*, and if it does, we call it *strong* or *total*. A strong correctness criterion is stronger than a weak one in the sense that the latter is implied by the former. Details of the distinction between the two forms of correctness for concurrent programs may be found in [LG] and [OH]. Our notion of correctness is a strong one and thus we shall need to give consideration to the problem of divergence. Consequently, in the second section we present a useful method for proving the divergence freedom of a wide class of frequently met networks.

As with deadlock freedom, other authors approach to proving total correctness (see [AFR, C, H3, LG, So]) has veered towards global methods. For us, having developed methods that require only local or directed global reasoning for showing deadlock freedom, a natural question is to ask is whether such an approach would be feasible for this more general problem. In the third section, this question is answered positively (and pleasingly) by showing that proving an implementation satisfies a specification is equivalent to proving a certain deadlock freedom result.

In the final section, we generalise the latter technique to reflect more

directly the natural partition between partial and total correctness aspects that occur in its use and so as to further ease its application.

5.1 Specifications and Implementations

Much discussion has taken place on what form a specification language should take and as the need for more powerful specification primitives has arisen, (see [Broo, Ge, LT]) the distinction between implementation language and specification language has blurred. It is the level of abstraction at which a system is described that seems to distinguish a specification from an implementation rather than the language used. An instance of this higher level of abstraction is the implicit non-determinism in specifications that is removed by making design choices in reaching the implementation. Also in the implementation we may need to use mechanisms not mentioned in the specification so as to achieve a result that is mentioned by it.

The following definition shows that the ideas mentioned above can be encapsulated neatly in the failures model.

Definition 5.1.1 If S and P are two processes such that

$$S \sqsubseteq P \backslash (\alpha P - \alpha S)$$

then we say P *implements* S and we may refer to S as a *specification* for P. We say a network V *implements* S precisely when the process $PAR(V)$ does.

This is a simple and highly intuitive definition of an implementation, the role of the non-determinism ordering $\sqsubseteq$ is clear while the hiding operator enables us to make the necessary leap away from the explicit implementation detail not given in the specification. Care however must be taken in the use of the latter operator in case divergence is introduced. Of course this is not a problem if divergence is allowed by the specification, but as one never deals with such specifications we shall require proofs of divergence freedom in checking such a relation holds. We briefly turn to this question in the next section.

Specifications, as well as being divergence free, are invariably also meant to be deadlock free. We note that in a case where we have proved the relation above, the implementation will inherit its deadlock freedom from that of the specification. Thus in some sense we will have carried out a deadlock freedom proof for the implementation as part of the larger proof.

It should be mentioned that an argument is given in [LT] against the use of Process Algebras as their own specification language. The main objection there being that any such specification limits its possible implementations to a single equivalence class with respect to the semantic equivalence defined on the process algebra. However, this is clearly untrue in the case of our defined relationship between specifications and implementations, this refinement ordering not being just one of equivalence.

5.2 Divergence Freedom

Divergence can be introduced into guarded processes defined in CSP syntax only via the use of hiding. It is thus appropriate to recall the following result which is fundamental to the manipulation of hiding for networks.

Lemma 5.2.1 ([BR1]) If $C \cap \alpha Q = \emptyset$, then $(P\backslash C \parallel Q) = (P \parallel Q)\backslash C$.

From the point of view of divergence freedom, the result above allows us to reduce the problem of proving $(P \parallel Q)\backslash C$ divergence free to that of proving $P\backslash C$ divergence free, when $C \cap \alpha Q = \emptyset$ and Q is non-divergent.

The following is an obvious fact about divergence freedom.

Lemma 5.2.2 If $P\backslash C$ is divergence free and $B \subseteq C$, then $P\backslash B$ is divergence free.

We now present a generalisation of a result in [R1]. The result there was concerned only with networks whose communication graph was a chain.

Theorem 5.2.3 Suppose $V = \langle P_1, \ldots, P_n \rangle$ is a triple-disjoint network of non-divergent processes with a partial order $\ll$ on the elements of the network such that every pair of neighbours is comparable. Suppose further that

$$P_i\backslash[\{\bigcup_{j \ll i}(\alpha P_i \cap \alpha P_j) \cup (\alpha P_i - \Lambda)\} - \beta] \quad \text{is divergence free}$$

for every P_i in V, where β is a fixed set. Then $PAR(V)\backslash(\alpha V - \beta)$ is divergence free.

Proof. We proceed by induction on n. The base case when $n = 1$ is trivial. Now say $n > 1$ and assume without loss of generality that n is maximal with

respect to $\ll$. Consider the subnetwork $V' = \langle P_1, \ldots, P_{n-1} \rangle$ with vocabulary Γ. Note that

$$\Lambda = \bigcup_{j \neq n} (\alpha P_n \cap \alpha P_j) \cup \Gamma$$

and so $\Lambda \subseteq \Gamma \cup \alpha P_n$. Moreover, if $i \neq n$, we observe that

$$\alpha P_n \cap [\bigcup_{j \ll i} (\alpha P_i \cap \alpha P_j)] = \emptyset$$

from the maximality of n with respect to $\ll$. Thus when $i \neq n$, we see that

$$\{\bigcup_{j \ll i} (\alpha P_i \cap \alpha P_j) \cup (\alpha P_i - \Gamma)\} - (\alpha P_n \cup \beta) \subseteq \{\bigcup_{j \ll i} (\alpha P_i \cap \alpha P_j) \cup (\alpha P_i - \Lambda)\} - \beta$$

and hence by Lemma 5.2.2 that

$$P_i \backslash [\{\bigcup_{j \ll i} (\alpha P_i \cap \alpha P_j) \cup (\alpha P_i - \Gamma)\} - (\alpha P_n \cup \beta)]$$

is divergence free. Then applying the inductive hypothesis to V' yields that $PAR(V') \backslash [\alpha V' - (\alpha P_n \cup \beta)]$ is divergence free. Now

$$\Lambda \cap \alpha P_n = \bigcup_{j \ll n} (\alpha P_n \cap \alpha P_j)$$

so

$$\{\bigcup_{j \ll n} (\alpha P_n \cap \alpha P_j) \cup (\alpha P_n - \Lambda)\} - \beta = \alpha P_n - \beta$$

and thus $P_n \backslash (\alpha P_n - \beta)$ is also divergence free. We assert that

$$PAR(V) \backslash (\alpha V - \beta) = [PAR(V') \parallel P_n] \backslash [(\alpha V' \cup \alpha P_n) - \beta]$$

is divergence free, and so let us assume for the sake of contradiction that this is not so. Then there is an infinite chain of traces $sw_0 < sw_1 < \cdots$ such that $sw_i \in traces(PAR(V))$, $w_i \in (\alpha V - \beta)^*$ and $|w_{i+1}| = |w_i| + 1$. Let p_i denote $(sw_i) \restriction \alpha P_n$ and consider the chain of traces $p_0 \leq p_1 \leq \cdots$. The set $\{p_i \mid i \in \mathbf{N}\}$ cannot be infinite, as otherwise we could construct an infinite chain of traces $p_{r_0} < p_{r_1} < \cdots$ with $p_{r_i} \in traces(P_n)$, $p_{r_i} \in (\alpha P_n - \beta)^*$ and $|p_{r_{i+1}}| = |p_{r_i}| + 1$ which would contradict the divergence freedom of $P_n \backslash (\alpha P_n - \beta)$. So $\{p_i \mid i \in \mathbf{N}\}$ is finite and thus there is a $k \in \mathbf{N}$ such that $p_r = p_{r+1}$ for every $r \geq k$. Letting $sw_r = sw_k v_r$ for $r \geq k$, we see that $v_r \in [\alpha V' - (\alpha P_n \cup \beta)]^*$ as $v_r \restriction \alpha P_n = \langle \rangle$. But then the infinite chain

of traces $((sw_k)\restriction\alpha V')v_k < ((sw_k)\restriction\alpha V')v_{k+1} < \cdots$ contradicts the fact that $PAR(V')\backslash[\alpha V' - (\alpha P_n \cup \beta)]$ is divergence free. □

This theorem may seem a little daunting at first sight, however generally the set β is $\alpha V - \Lambda$, that is the set of communications external to the network. In this case the theorem says that given a partial order on the elements of a network, if each element cannot communicate forever with the neighbours less than it, then when the vocabulary of the network is hidden, the resultant process is non-divergent.

The following is a typical example of its use.

Example 5.2.1 We return again to the network in Example 2.3.2. Let $\ll$ be the same partial order as the order $<$ defined there and choose β to be the set of external communications of the network. The preconditions of the theorem above may then be translated to saying that no element in the network can input infinitely from its neighbours without producing an output. This is clearly true and thus we may deduce that if all the channels except *in* and the out_n are hidden, the resulting process is divergence free. □

In general, it seems that establishing the divergence freedom of a process is less difficult than establishing its deadlock freedom and the techniques above are sufficient to deduce the former for a wide variety of examples.

5.3 Checking Processes

Referring to Definition 5.1.1, we see that the processes on both sides of $\sqsubseteq$ have the same alphabet and are both divergence free (the latter under our assumption that specifications are divergence free). Thus we shall assume that all processes considered here have a common alphabet Σ and are divergence free. As a consequence, the failures of the parallel operator take the simpler form

$$\mathcal{F}[\![P \parallel Q]\!] = \{(s, X \cup Y) \mid (s, X) \in \mathcal{F}[\![P]\!] \wedge (s, Y) \in \mathcal{F}[\![Q]\!]\} \quad .$$

We shall also assume that all processes are non-terminating for the sake of clarity.

The essence of the idea now is that given a specification S, we need to easily obtain a *checking process* $\delta(S)$ such that if the parallel composition

of this with another process is deadlock free, then the latter process implements (is more deterministic than) the specification. This is an extremely appealing idea and hence it was of no surprise to discover the use of a similar idea in [Brin]. However, the motivation and details of our work are quite different from those there and we discuss these differences at the end of the section.

It is not obvious that such a function $\delta : CSP \to CSP$ even exists let alone that it will be well-behaved. Let us consider the function

$$\delta(P) = \sqcap C(P) \quad \text{where} \quad C(P) \mathrel{\widehat{=}} \{Q \mid P \parallel Q \text{ is deadlock free}\}$$

as a possible candidate. Note that we use $\sqcap$ here in an informal sense as the greatest lower bound operator with respect to $\sqsubseteq$. Regrettably the function above is not always defined even when $C(P) \neq \emptyset$. This is because of the possible introduction of unbounded non-determinism. An account of this problem and two remedies may be found in [R2] and [R3]. Unfortunately, both of the newer semantics in the references above require that axiom (F5) of the failures model be necessarily discarded with the result that we would quite rightly no longer be guaranteed the existence of maximal refusal sets. However if Σ is finite, the failures model and the model in [R2] coincide and this technical difficulty is resolved. For this reason, we shall assume henceforth that Σ is finite.

We may now formally define the operator $\sqcap \mathcal{S}$, where $\mathcal{S}$ is an arbitrary set of processes, in the obvious way by

$$\mathcal{F}[\![\sqcap \mathcal{S}]\!] = \bigcup_{P \in \mathcal{S}} \mathcal{F}[\![P]\!] \quad .$$

As an example, we see that if $\perp_{\text{df}}$ denotes the most non-deterministic deadlock free process then $\mathcal{F}[\![\perp_{\text{df}}]\!] = \{(s, X) \mid X \neq \Sigma\}$.

Returning to the subject of δ, we can now conclude that $\delta(P)$ is defined if and only if $C(P) \neq \emptyset$ and when defined it is divergence free.

Here are some basic properties that δ possesses.

Lemma 5.3.1 If $\delta(P)$ is defined, then the following conditions hold:

(i) $P \parallel Q$ deadlock free $\Rightarrow \delta(P) \sqsubseteq Q$.

(ii) $\delta(P) \parallel P$ is deadlock free.

(iii) $\delta^2(P)$ is defined and $\delta^2(P) \sqsubseteq P$.

Proof.

(i) Follows from the fact that $\delta(P)$ is the most non-deterministic process that when put in parallel with P, results in a deadlock free process.

(ii) Assume for the sake of contradiction that $\delta(P) \parallel P$ is not deadlock free. It follows that there is a failure (s, X) such that $(s, X) \in \delta(P)$ and $(s, \Sigma - X) \in P$. But then there is a $Q \in C(P)$ with $(s, X) \in Q$, and hence $P \parallel Q$ is not deadlock free, contradicting the definition of $C(P)$.

(iii) From (ii), $\delta(P) \parallel P$ is deadlock free and so $P \in C(\delta(P))$. Hence $\delta^2(P)$ is defined as $C(\delta(P)) \neq \emptyset$. Using (i) and (ii) together shows that $\delta^2(P) \sqsubseteq P$.

□

The following lemma shows the behaviour of δ with respect to the non-determinism ordering and the choice operators.

Lemma 5.3.2

(i) If $\delta(P)$ is defined, then $P \sqsubseteq Q \Rightarrow \delta(Q) \sqsubseteq \delta(P)$.

(ii) If $\delta(P \sqcap Q)$ is defined, then $\delta(P) \sqsubseteq \delta(P \sqcap Q)$.

(iii) If $\delta(P)$, $\delta(Q)$, $\delta(P \,\Box\, Q)$ are all defined, then $\delta(P) \sqcap \delta(Q) \sqsubseteq \delta(P \,\Box\, Q)$.

Proof.

(i) It is easy to see that $C(P) \subseteq C(Q)$, thus $\delta(Q)$ exists and $\delta(Q) \sqsubseteq \delta(P)$.

(ii) Follows from (i) by noting that $P \sqcap Q \sqsubseteq P$.

(iii) Say $(s, X) \in \mathcal{F}[\![\delta(P \,\Box\, Q)]\!]$ and assume for the sake of contradiction that $(s, X) \notin \mathcal{F}[\![\delta(P) \sqcap \delta(Q)]\!]$. First say $s \neq \langle\rangle$. We may assume without loss of generality that s is a trace of minimal length in

$$\mathcal{F}[\![\delta(P \,\Box\, Q)]\!] - \mathcal{F}[\![\delta(P \sqcap Q)]\!]$$

and hence any strict prefix of s is a trace of $\delta(P)$ or $\delta(Q)$, say (again without loss of generality) $\delta(P)$. Now let $(s, W) \in \tilde{\mathcal{F}}[\![\delta(P \,\Box\, Q)]\!]$ such that $X \subseteq W$, and consider the set

$$\begin{aligned} &\mathcal{F}[\![\delta(P)]\!] \cup \{(s, Y) \mid Y \subseteq W\} \\ &\quad \cup \{(s \langle a \rangle t, Y) \mid a \notin W \wedge (s \langle a \rangle t, Y) \in \mathcal{F}[\![\delta(P \,\Box\, Q)]\!]\} \end{aligned}$$

Observe that if $(s\langle a\rangle, \emptyset) \notin \mathcal{F}[\![\delta(P \,\Box\, Q)]\!]$ then it follows from the properties of the failures model that $(s, W \cup \{a\}) \in \mathcal{F}[\![\delta(P \,\Box\, Q)]\!]$, and thus $a \in W$, from the maximality of W. It is then easy to verify that the set above is a failure set, so let $\delta(P)_{(s,W)}$ denote the corresponding process. We assert that $\delta(P)_{(s,W)} \parallel P$ is deadlock free. If it is not, there is a failure (u, V) such that $(u, V) \in \mathcal{F}[\![\delta(P)_{(s,W)}]\!]$ and $(u, \Sigma - V) \in \mathcal{F}[\![P]\!]$. Now $(u, V) \notin \mathcal{F}[\![\delta(P)]\!]$, as $\delta(P) \parallel P$ is deadlock free, and so from the construction of $\delta(P)_{(s,W)}$ we may deduce that $(u, V) \in \mathcal{F}[\![\delta(P \,\Box\, Q)]\!]$. But as $u \neq \langle\rangle$ we see $(u, \Sigma - V) \in \mathcal{F}[\![P \,\Box\, Q]\!]$ which contradicts the fact that $\delta(P \,\Box\, Q) \parallel (P \,\Box\, Q)$ is deadlock free. Thus $\delta(P)_{(s,W)} \parallel P$ is deadlock free and hence $\delta(P) \sqsubseteq \delta(P)_{(s,W)}$. However, as $(s, X) \in \mathcal{F}[\![\delta(P)_{(s,W)}]\!] - \mathcal{F}[\![\delta(P)]\!]$, this too is a contradiction. Now say $s = \langle\rangle$. Then $(\langle\rangle, X) \in \mathcal{F}[\![\delta(P \,\Box\, Q)]\!]$ and so we must have $(\langle\rangle, \Sigma - X) \notin \mathcal{F}[\![P \,\Box\, Q]\!]$. We may assume, without loss of generality, that $(\langle\rangle, \Sigma - X) \notin \mathcal{F}[\![P]\!]$ and let $(\langle\rangle, W) \in \tilde{\mathcal{F}}[\![\delta(P \,\Box\, Q)]\!]$ be such that $X \subseteq W$. Defining $\delta(P)_{(\langle\rangle,W)}$ as above and pursuing a similar argument, we obtain a contradiction again.

□

It is interesting to note that the behaviour of δ is non-determinism inverting in some sense.

Recalling our motivation for the use of δ, we see that δ would be applied to a deadlock free process (specification). Thus turning our attention to the behaviour of δ with respect to deadlock free processes, we would desire at least that it is defined for such processes. The next lemma is important and surprising in that not only is this true but that we have far more.

Lemma 5.3.3 If P is deadlock free, then the following conditions hold:

(i) $\delta(P)$ is defined.

(ii) $(s, X) \in \mathcal{F}[\![P]\!] \Leftrightarrow (s, \Sigma - X) \notin \mathcal{F}[\![\delta(P)]\!]$.

(iii) $\delta^2(P) = P$.

Proof.

(i) As P is deadlock free we have that $P \parallel RUN$ too is deadlock free, showing $RUN \in C(P)$. Thus $C(P) \neq \emptyset$ and hence $\delta(P)$ is defined.

(ii) From above, $\delta(P)$ is defined. Thus it follows from Lemma 5.3.1 (ii) that $(s, X) \in \mathcal{F}[\![P]\!]$ implies $(s, \Sigma - X) \notin \mathcal{F}[\![\delta(P)]\!]$. Now assume that $(s, X) \notin \mathcal{F}[\![P]\!]$ when $(s, \Sigma - X) \notin \mathcal{F}[\![\delta(P)]\!]$, for the sake of contradiction. Consider the set

$$\mathcal{F}[\![RUN]\!] \cup \{(s, Y) \mid Y \subseteq \Sigma - X\} \quad .$$

It is easy to verify that this is a failure set, so let Q denote the corresponding process. Now as P is deadlock free, $P \parallel RUN$ is deadlock free and noting that $(s, X) \notin \mathcal{F}[\![P]\!]$, we see that $P \parallel Q$ is deadlock free. Then by Lemma 5.3.1 (i), $\delta(P) \sqsubseteq Q$, which contradicts the fact that $(s, \Sigma - X) \in \mathcal{F}[\![Q]\!]$ but $(s, \Sigma - X) \notin \mathcal{F}[\![\delta(P)]\!]$.

(iii) As $\delta(P)$ is defined, using Lemma 5.3.1 (iii) we deduce that $\delta^2(P) \sqsubseteq P$. Thus if $\delta^2(P) \neq P$, there is a $(s, X) \in \mathcal{F}[\![\delta^2(P)]\!] - \mathcal{F}[\![P]\!]$. Applying (ii) once yields that $(s, \Sigma - X) \notin \mathcal{F}[\![\delta(P)]\!]$, and applying it again shows that $(s, X) \in \mathcal{F}[\![P]\!]$, which is a contradiction.

□

The idempotence of δ on deadlock free processes is remarkable as well as being crucial in the proof of the main theorem of this section. The result seems even more startling when we notice that δ is not even closed with respect to application on the set of deadlock free processes.

The next result realises our goals for δ and is thus the main theorem of the section.

Theorem 5.3.4 Let S be a deadlock free process. Then

$$\delta(S) \parallel P \text{ is deadlock free } \Leftrightarrow \; S \sqsubseteq P \quad .$$

Proof. If $S \sqsubseteq P$, we see that $(\delta(S) \parallel S) \sqsubseteq (\delta(S) \parallel P)$ as $\parallel$ is monotonic. By Lemma 5.3.1 (ii) we have that $\delta(S) \parallel S$ is deadlock free, and hence $\delta(S) \parallel P$ is also deadlock free. Now say $\delta(S) \parallel P$ is deadlock free. Then by Lemma 5.3.1 (i) we see that $\delta^2(S) \sqsubseteq P$, so using Lemma 5.3.3 (iii) we conclude that $S \sqsubseteq P$. □

Although this is the main theoretical result of the section, the practical details of its use as a proof technique are made more visible in Theorem 5.2.8 (Checking Theorem).

Of course for δ to be useful, it should as mentioned earlier be easy to calculate. We now present some results that show that this is so. The first shows that its failures are extremely simple to state.

Lemma 5.3.5 Let S be a deadlock free process. Then

$$\mathcal{F}[\![\delta(S)]\!] = \{(s,X) \mid (s,\Sigma - X) \notin \mathcal{F}[\![S]\!]\} \quad .$$

Proof. Follows immediately from Lemma 5.3.3 (ii). □

The failures of its application on some of the most frequently used operators are then as follows.

Lemma 5.3.6 Let P, Q be deadlock free and $\mu x.F(x)$ be divergence free, where F is a continuous operator such that $F(P)$ is deadlock free whenever P is. Then the following equalities hold:

$$\begin{aligned}
\mathcal{F}[\![\delta(a \to P)]\!] &= \{(\langle\rangle, X) \mid a \notin X\} \cup \{(\langle b\rangle s, X) \mid a \neq b\} \\
&\quad \cup \{(\langle a\rangle s, X) \mid (s,X) \in \mathcal{F}[\![\delta(P)]\!]\} \\
\mathcal{F}[\![\delta(P \,\Box\, Q)]\!] &= \{(\langle\rangle, X) \mid (\langle\rangle, X) \in \mathcal{F}[\![\delta(P)]\!] \cup \mathcal{F}[\![\delta(Q)]\!]\} \\
&\quad \cup \{(s,X) \mid s \neq \langle\rangle \wedge (s,X) \in \mathcal{F}[\![\delta(P)]\!] \cap \mathcal{F}[\![\delta(Q)]\!]\} \\
\mathcal{F}[\![\delta(P \sqcap Q)]\!] &= \mathcal{F}[\![\delta(P)]\!] \cap \mathcal{F}[\![\delta(Q)]\!] \\
\mathcal{F}[\![\delta(\bot_{\mathrm{df}})]\!] &= \mathcal{F}[\![RUN]\!]
\end{aligned}$$

and

$$\mathcal{F}[\![\delta(\mu x.F(x))]\!] = \bigcup_{n=0}^{\infty} \mathcal{F}[\![(\delta \circ F^n)(\bot_{\mathrm{df}})]\!] \quad .$$

Proof. Using Lemma 5.3.5, it is straightforward to derive all the equalities above except perhaps for the last one which is proved as follows. First note that $\bot_{\mathrm{df}} \sqsubseteq F(\bot_{\mathrm{df}})$, as $F(\bot_{\mathrm{df}})$ is deadlock free, and proceeding inductively we obtain a chain

$$\bot_{\mathrm{df}} \sqsubseteq F(\bot_{\mathrm{df}}) \sqsubseteq F^2(\bot_{\mathrm{df}}) \sqsubseteq \cdots$$

which consequently has a least upper bound P_{df} such that

$$\mathcal{F}[\![P_{\mathrm{df}}]\!] = \bigcap_{n=0}^{\infty} \mathcal{F}[\![F^n(\bot_{\mathrm{df}})]\!] \quad .$$

By the usual argument, as F is continuous, P_{df} is a fixed point of F. Now from results on classes of predicates like "is deadlock free" in [R1, R2], we may deduce that $\mu x.F(x)$ is deadlock free. Then $\bot_{\mathrm{df}} \sqsubseteq \mu x.F(x)$ and

thus, again by a trivial induction, we see that $F^n(\perp_{\mathrm{df}}) \sqsubseteq \mu x.F(x)$ whenever $n \in \mathbf{N}$. So $\mu x.F(x)$ is an upper bound for the chain above and hence $P_{\mathrm{df}} \sqsubseteq \mu x.F(x)$. But $\mu x.F(x)$ is the least fixed point of F and so we must have $\mathcal{F}[\![P_{\mathrm{df}}]\!] = \mathcal{F}[\![\mu x.F(x)]\!]$. Now from Lemma 5.3.5 we have

$$\mathcal{F}[\![\delta(\mu x.F(x))]\!] = \{(s,X) \mid (s,\Sigma - X) \notin \mathcal{F}[\![\mu x.F(x)]\!]\}$$

and so from the definition of P_{df} we see that

$$\mathcal{F}[\![\delta(\mu x.F(x))]\!] = \{(s,X) \mid (s,\Sigma - X) \notin \bigcap_{n=0}^{\infty} \mathcal{F}[\![F^n(\perp_{\mathrm{df}})]\!]\} \quad .$$

Moreover, $F^n(\perp_{\mathrm{df}})$ is deadlock free showing that $(\delta \circ F^n)(\perp_{\mathrm{df}})$ exists and thus applying Lemma 5.3.5 once more and simplifying yields that

$$\mathcal{F}[\![\delta(\mu x.F(x))]\!] = \bigcup_{n=0}^{\infty} \mathcal{F}[\![(\delta \circ F^n)(\perp_{\mathrm{df}})]\!] \quad .$$

□

It would of course be useful to be able to calculate the syntactic representation of the application of δ as well as the semantic. A statement that is often true in practice is that a specification is often sequential even though the implementation may be highly distributed. This is because a specification is generally written at a level where it is fully understood and this frequently corresponds to a sequential overview of a system. (This is always possible to do when an interleaving semantics is given to the parallel operator as in the failures model.) With this in mind, we have the following procedure for deriving a syntactic representation for the application of δ.

Lemma 5.3.7 Suppose the definition of the process S uses only the following syntax:

$$P ::= SKIP \mid a \rightarrow P \mid P;P \mid P \,\Box\, P \mid P \sqcap P \mid p \mid \mu p.P$$

(where p denotes a process variable), but S contains no free process variables. Further suppose S is divergence free, has every occurrence of $SKIP$ directly or indirectly followed by a ";", and if S contains a construct of the form $P \,\Box\, Q$ or $P \sqcap Q$ then

$$x \in initials(P) \cap initials(Q) \;\Rightarrow\; P \underline{\text{ after }} \langle x \rangle = Q \underline{\text{ after }} \langle x \rangle.$$

Now define functions $\delta : CSP \to CSP$, $\Xi : \mathcal{P}(\Sigma) \to CSP$ as follows:

$$\begin{aligned}
\bar{\delta}(SKIP) &= SKIP \\
\bar{\delta}(a \to P) &= a \to \delta(P) \\
\bar{\delta}(P;Q) &= \bar{\delta}(P);\delta(Q) \\
\bar{\delta}(P \,\Box\, Q) &= \bar{\delta}(P) \sqcap \bar{\delta}(Q) \\
\bar{\delta}(P \sqcap Q) &= \bar{\delta}(P) \,\Box\, \bar{\delta}(Q) \\
\bar{\delta}(\mu p.P) &= \bar{\delta}([(\mu p.P)\backslash p]P)
\end{aligned}$$

$$\Xi(X) = STOP \sqcap \left(\prod_{x \notin X} x \to CHAOS \right)$$

Then $\delta(S) = \bar{\delta}(S) \,\Box\, \Xi(initials(S))$ is satisfied.

Proof. From Lemma 1.1.1, S is deadlock free and hence $\delta(S)$ certainly exists. It is straightforward to prove that $initials(S) = initials(\bar{\delta}(S))$ by structural induction. Using the same method, we may prove that $\bar{\delta}(S) \parallel S$ is deadlock free. In the case where $S = P \,\Box\, Q$ the proof proceeds as follows. Assume for the sake of contradiction that the condition does not hold. Now

$$\bar{\delta}(P \,\Box\, Q) \parallel (P \,\Box\, Q) = [\bar{\delta}(P) \parallel (P \,\Box\, Q)] \sqcap [\bar{\delta}(Q) \parallel (P \,\Box\, Q)]$$

so we may assume without loss of generality that $\bar{\delta}(P) \parallel (P \,\Box\, Q)$ is not deadlock free. From the inductive hypothesis that $\bar{\delta}(P) \parallel P$ is deadlock free it follows that there is a failure (s, X) such that $s \neq \langle\rangle$, $(s, X) \in \mathcal{F}[\![\bar{\delta}(P)]\!]$ and $(s, \Sigma - X) \in \mathcal{F}[\![Q]\!]$. Say $s = \langle a \rangle t$. As $initials(P) = initials(\bar{\delta}(P))$, we see that $a \in initials(P) \cap initials(Q)$ and hence $P \underline{\text{ after }} \langle a \rangle = Q \underline{\text{ after }} \langle a \rangle$. But then $(s, \Sigma - X) \in \mathcal{F}[\![P]\!]$ contradicting the fact that $\bar{\delta}(P) \parallel P$ is deadlock free.

The deadlock freedom of $[\bar{\delta}(S) \,\Box\, \Xi(initials(S))] \parallel S$ is then an easy consequence of that of $\bar{\delta}(S) \parallel S$, and hence

$$\delta(S) \sqsubseteq \bar{\delta}(S) \,\Box\, \Xi(initials(S))$$

by Lemma 5.3.1 (i). Thus it only remains to prove the reverse inequality. We do this again by structural induction and the case where $S = P \,\Box\, Q$ is as follows. Assume for the sake of contradiction that $(s, X) \in \mathcal{F}[\![\delta(S)]\!]$ and $(s, X) \notin \mathcal{F}[\![\bar{\delta}(S) \,\Box\, \Xi(initials(S))]\!]$. We now use the relevant equality in Lemma 5.3.6. If $s = \langle\rangle$ then $(\langle\rangle, X) \in \mathcal{F}[\![\delta(P)]\!] \cup \mathcal{F}[\![\delta(Q)]\!]$, without loss of generality assume that $(\langle\rangle, X) \in \mathcal{F}[\![\delta(P)]\!]$. But from the inductive hypothesis on P, $\bar{\delta}(P) \,\Box\, \Xi(initials(P))$ so it follows that $(\langle\rangle, X) \in \mathcal{F}[\![\bar{\delta}(P)]\!]$

and thus $(s,X) \in \mathcal{F}[\![\bar{\delta}(S) \,\Box\, \Xi(initials(S))]\!]$ which is a contradiction. Now say $s \neq \langle\rangle$ and so $s = \langle a \rangle t$. Then $(s,X) \in \mathcal{F}[\![\delta(P)]\!] \cap \mathcal{F}[\![\delta(Q)]\!]$ and using $(s,X) \notin \mathcal{F}[\![\bar{\delta}(S) \,\Box\, \Xi(initials(S))]\!]$ and the inductive hypotheses on P and Q, we see that $(s,X) \in \mathcal{F}[\![\Xi(initials(P))]\!]$ and $(s,X) \in \mathcal{F}[\![\Xi(initials(Q))]\!]$. But this yields that $a \notin initials(P)$ and $a \notin initials(Q)$ so we must have $(s,X) \in \mathcal{F}[\![\Xi(initials(S))]\!]$, which is a contradiction. □

The condition on the choice operators in the lemma above is in practice not a restriction as we can rewrite an expression so that the condition applies to it, if it did not earlier.

We have so far assumed that all processes have the same alphabet Σ. In practice, when using $\delta(S)$ in the sense of Theorem 5.3.4, we need only ensure that S, $\delta(S)$ and the process being checked have the same alphabet. Thus we use $\Sigma = \alpha S$ in the calculation of $\delta(S)$.

Here is an example calculation of a checking process which displays the ease with which this can generally be done.

Example 5.3.1 The specification for a buffer (i.e. the most non-deterministic buffer) using the channels *in* and *out* is defined as follows:

$$BUFF\text{-}S = BUFF(in, out, \langle\rangle) \qquad \alpha BUFF\text{-}S = in.\Sigma \cup out.\Sigma$$

where

$$BUFF(in, out, \langle\rangle) = in?x \rightarrow BUFF(in, out, \langle x \rangle)$$

$$BUFF(in, out, \langle y \rangle s) = \quad out!y \rightarrow BUFF(in, out, s) \;\Box\; \boxed{out!y \rightarrow BUFF(in, out, s) \;\sqcap\; \boxed{in?x \rightarrow BUFF(in, out, \langle y \rangle s \langle x \rangle)}}$$

As $BUFF\text{-}S$ is deadlock free, a checking process exists for it. A checking process for $BUFF(in, out, s)$ exists for the same reason, thus if

$$P(s) = \delta(BUFF(in, out, s))$$

then Lemma 5.3.7 yields the following definition for $P(s)$.

$$P(\langle\rangle) \;=\; \boxed{\sqcap_{x \in \Sigma} \; in.x \rightarrow P(\langle x \rangle)} \;\Box\; \Xi(in.\Sigma)$$

$$P(\langle y\rangle s) =$$

$$\sqcap \Big[\; out!y \to P(s) \;\Big[\; out!y \to P(s) \;\Big[\; \sqcap_{x\in\Sigma}\; in.x \to P(\langle y\rangle s\langle x\rangle) \;\Big]\Big]\Big] \;\Box\; \Xi(in.\Sigma \cup \{out.y\})$$

Hence we conclude that

$$\delta(BUFF\text{-}S) = P(\langle\rangle)$$

with $\alpha\delta(BUFF\text{-}S) = \alpha BUFF\text{-}S$. □

Having shown that both the theory and the calculation of checking processes is tractable, we present the proof technique for checking implementations we promised, in its most directly applicable form.

Theorem 5.3.8 (Checking Theorem) Let S be a deadlock free process specification and $V = \langle P_1, \ldots, P_n\rangle$ be some network. Further suppose that

- the process $PAR(V)\backslash(\alpha V - \alpha S)$ is divergence free, and
- the network $\langle\delta(S), P_1, \ldots, P_n\rangle$ is deadlock free.

Then V implements the specification S.

Proof. As $\alpha\delta(S) = \alpha S$, we observe via Lemma 5.2.1 that

$$[\delta(S) \parallel PAR(V)]\backslash(\alpha V - \alpha S) = \delta(S) \parallel [PAR(V)\backslash(\alpha V - \alpha S)] \quad .$$

Now $PAR(V)\backslash(\alpha V - \alpha S)$ is divergence free inferring the divergence freedom of $[\delta(S) \parallel PAR(V)]\backslash(\alpha V - \alpha S)$, and thus noting that

$$\delta(S) \parallel PAR(V) = PAR(\langle\delta(S), P_1, \ldots, P_n\rangle)$$

we deduce by Lemma 1.1.2 that $\delta(S) \parallel [PAR(V)\backslash(\alpha V - \alpha S)]$ is deadlock free. Then by using Theorem 5.3.4, we conclude

$$S \sqsubseteq PAR(V)\backslash(\alpha V - \alpha S)$$

as required. □

The example below illustrates a use of this technique.

Example 5.3.2 We prove the following well-known result.

Theorem. A pipeline of buffers is a buffer.

Proof. Let $n \geq 2$ be a constant and define processes

$$BUFF_1 = BUFF(in, mid_1, \langle\rangle) \qquad \alpha BUFF_1 = in.\Sigma \cup mid_1.\Sigma$$

$$BUFF_n = BUFF(mid_n, out, \langle\rangle) \qquad \alpha BUFF_n = mid_n.\Sigma \cup out.\Sigma$$

$$BUFF_i = BUFF(mid_{i-1}, mid_i, \langle\rangle) \qquad \alpha BUFF_i = mid_{i-1}.\Sigma \cup mid_i.\Sigma$$

where $2 \leq i \leq n-1$ and $BUFF$ is defined as in Example 5.3.1. Now consider the network $V = \langle BUFF_1, \ldots, BUFF_n \rangle$. Clearly, $PAR(V)$ is the specification for a pipeline (of length n) of buffers, and hence to prove the theorem it is sufficient to prove that V implements $BUFF\text{-}S$.

First we see that $PAR(V)\backslash(\alpha V - \alpha BUFF\text{-}S)$ is divergence free. This is proved by using the usual order $<$ on the process indices in V for the order required by Theorem 5.2.3, and noting that a buffer cannot output values infinitely without doing an input. We now assert that the network $\langle \delta(BUFF\text{-}S), BUFF_1, \ldots, BUFF_n \rangle$ is deadlock free. This network is triple-disjoint but not busy or conflict free, although V is. We define predicates $EMPTY_i(\sigma)$, $FULL_i(\sigma)$ which hold when $BUFF_i$ is only willing to input or output respectively, in the state σ of $BUFF_i$. Now define the invariant

$$\begin{aligned} &\Im(\langle s, \underline{X} \rangle, \Delta) \mathrel{\widehat{=}} \\ &\qquad \underline{s\restriction in.\Sigma} \leq \underline{s\restriction mid_1.\Sigma} \leq \cdots \leq \underline{s\restriction mid_{n-1}.\Sigma} \leq \underline{s\restriction out.\Sigma} \\ &\wedge \quad \Delta = \max(\{0\} \cup \{i : \{1, \ldots, n\} \mid \neg EMPTY_i(\langle s, \underline{X} \rangle \restriction \alpha BUFF_i)\}) \end{aligned}$$

where $\underline{s}$ denotes the trace derived from a trace s by stripping all channel names from its elements. The validity of the invariant follows easily from the fact that the output trace of a buffer is a prefix of the input trace. Then the first part of the invariant shows that the network is globally busy and globally conflict free. Observe that Δ in the invariant denotes the process index of a non-empty buffer when not all the buffers are empty. Further define variant functions

$$f_\delta : \mathcal{F}[\![\delta(BUFF\text{-}S)]\!] \to \mathbf{N}$$

$$f_i : \mathcal{F}[\![BUFF_i]\!] \to \mathbf{N} \qquad (1 \leq i \leq n)$$

by putting

$$f_\delta(\sigma,\Delta) = n+1-\begin{cases} \Delta & \text{if } \neg[\delta(BUFF\text{-}S) \xrightarrow{\sigma} BUFF_1] \\ 0 & \text{otherwise} \end{cases}$$

$$f_i(\sigma,\Delta) = \begin{cases} (i-\Delta) \bmod (n+1) & \text{if } EMPTY_i(\sigma) \\ n-i & \text{if } FULL_i(\sigma) \\ n+1 & \text{otherwise} \end{cases}$$
$$(1 \leq i \leq n)$$

where **N** has its usual order $>$. Then we infer via the Global Variant Rule that the network is deadlock free, and hence by the Checking Theorem that V implements $BUFF\text{-}S$, as required. □

Finally, we return to the question of the differences between our work and that in [Brin]. The work there is motivated by the need to generate testing programs that are actually run in parallel with potential implementations, the execution of the system being observed for the appearance of deadlock. We do not have this practical limitation (as we prove deadlock freedom) and thus our work can be more general. In addition to this, the notion of implementation there is one that is different (weaker) than ours and consequently the corresponding technical details are also quite different. Another difference is that the work there is not concerned with proving any liveness properties of the implementation.

5.4 Checking Pairs

In this section we generalise the notion of a checking process that was discussed in the last section.

We shall again assume that processes are neither divergent or ever willing to terminate, and also that the events in the alphabets of processes are drawn from a finite set Σ. The parallel operator will also take the form described in the last section as we shall always assume that processes put in parallel have the same alphabet. While on the subject of alphabets, we reiterate that the failures of a process are composed solely from events in its alphabet.

Consider the following definition.

Definition 5.4.1 Let $\mathcal{S}$ be a set of processes. We say that $(\mathcal{C},\mathcal{R})$ is a

checking pair for $\mathcal{S}$ if $(\mathcal{C}, \mathcal{R}) : \mathcal{S} \to \mathrm{CSP} \times \mathcal{P}(\Sigma^*)$ is a function such that

$$S \sqsubseteq P \Leftrightarrow \begin{cases} traces(P) \subseteq \mathcal{R}(S) \\ \text{and } \mathcal{C}(S) \parallel P \text{ is deadlock free} \end{cases}$$

whenever $S \in \mathcal{S}$ and $P \in \mathrm{CSP}$.

The components of a checking pair relate to the two correctness criteria which we want a process that is being checked to meet. $\mathcal{R}$ corresponds to a partial correctness condition (such as trace containment, see [CH, CM2, OH]) while $\mathcal{C}$ deals with the residual properties necessary for total correctness.

It is easy to see that $traces(S) \subseteq \mathcal{R}(S)$ and that if $\mathcal{R}'$ is a function such that

$$traces(S) \subseteq \mathcal{R}'(S) \subseteq \mathcal{R}(S)$$

then $(\mathcal{C}, \mathcal{R}')$ too is a checking pair for $\mathcal{S}$. This shows that we can always strengthen a partial correctness condition as long as it is not the strongest such condition, which is $\mathcal{R} = traces$.

We already have an example of a checking pair.

Example 5.4.1 Define $\alpha^* : \mathrm{CSP} \to \mathcal{P}(\Sigma^*)$ by $\alpha^*(P) = (\alpha P)^*$. Then (δ, α^*) is a checking pair for deadlock free processes, by results in the last section. We observe that the role of α^* is in fact redundant in the checking, as it is only reiterating the condition that the alphabet of a checked process is the same as that of the specification process. Thus the burden of checking rests entirely on δ. □

The application of δ does not always result in a deadlock free process. It would be desirable if $\mathcal{C}$ in a checking pair would produce deadlock free processes, as this would facilitate the use of our deadlock freedom Rules in checking. The following result shows that we cannot, in general, have both checking processes deadlock free and the partial correctness requirement redundant.

Lemma 5.4.1 Let $(\mathcal{C}, \mathcal{R})$ be a *checking pair* for $\mathcal{S}$ where $S \in \mathcal{S}$ is such that $\mathcal{C}(S)$ is deadlock free. Then $\mathcal{R}(S) \supseteq (\alpha S)^*$ if and only if $traces(S) = (\alpha S)^*$.

Proof. Follows from the properties of $RUN_{\alpha S}$ and the fact that S can be successfully checked by the pair. □

Thus to achieve deadlock free checking processes, the role of the partial correctness component in a checking pair will have to be non-trivial.

We now give examples of checking pairs where $\mathcal{C}$ does produce deadlock free processes.

Example 5.4.2 Let $S = (\mu x.\, a \to x)$ and define

$$\begin{aligned} \mathcal{C}(S) &= \mu x.((a \to x) \sqcap (a \to x \,\Box\, b \to x)) \\ \mathcal{R}(S) &= (\alpha S)^* - \{\langle a \rangle^n \langle b \rangle \mid n \in \mathbf{N}\} \end{aligned}$$

where $\alpha S = \{a, b\}$. Then $(\mathcal{C}, \mathcal{R})$ is a checking pair for $\{S\}$ such that $\mathcal{C}(S)$ is deadlock free. We note that $\mathcal{R}$ is not redundant, and as $\mathcal{R}(S) \neq traces(S)$ it is not the strongest partial correctness condition. We also observe that $\mathcal{C}(S)$ does not have the same traces as S although in general, if S is deadlock free, then $traces(\mathcal{C}(S)) \supseteq traces(S)$. $\Box$

The next example arises from our desire to obtain a function that behaves much like δ, but whose application produces deadlock free processes, thus the development of this theory follows closely that of δ's.

Example 5.4.3 Consider the function

$$\delta^*(P) = \sqcap \{Q \mid P \parallel Q \text{ is deadlock free} \wedge traces(Q) \subseteq traces(P)\}$$

where $\alpha\delta^*(P) = \alpha P$. Then $\delta^*(P)$ is defined if and only if the set whose non-deterministic composition is being taken in the definition is non-empty, and when δ^* is defined, it is divergence free.

Lemma 5.4.2 If $\delta^*(P)$ is defined, then the following conditions hold:

(i) $P \parallel Q$ deadlock free $\wedge$ $traces(Q) \subseteq traces(P) \Rightarrow \delta^*(P) \sqsubseteq Q$.

(ii) $\delta^*(P) \parallel P$ is deadlock free.

(iii) $traces(\delta^*(P)) \subseteq traces(P)$.

(iv) $\delta^*(P)$ is deadlock free.

(v) $P \sqsubseteq Q \wedge traces(P) = traces(Q) \Rightarrow \delta^*(Q) \sqsubseteq \delta^*(P)$.

(vi) $(\delta^*)^2(P)$ is defined and $(\delta^*)^2(P) \sqsubseteq P$.

Proof. The first three facts follow easily from the definition of $\delta^*(P)$ and so we shall only display proofs for the remainder.

(iv) Assume for the sake of contradiction that $(s, \alpha P) \in \mathcal{F}[\![\delta^*(P)]\!]$. Then by (iv), $(s, \emptyset) \in \mathcal{F}[\![P]\!]$ and hence $\delta^*(P) \parallel P$ is not deadlock free contradicting (ii).

(v) Note that if R is a process such that $P \parallel R$ is deadlock free and $traces(R) \subseteq traces(P)$, then it follows that $Q \parallel R$ is deadlock free and $traces(R) \subseteq traces(Q)$. Thus $\delta^*(Q)$ is defined, as $\delta^*(P)$ is, and $\delta^*(Q) \sqsubseteq \delta^*(P)$.

(vi) For any process Q we may define a process Q_{RUN} as follows:

$$\mathcal{F}[\![Q_{RUN}]\!] = \{(s, X) \mid s \in traces(Q) \wedge X \subseteq \alpha Q - initials(Q \underline{\text{ after }} s)\}.$$

Observe that $traces(Q) = traces(Q_{RUN})$, and if Q is deadlock free then so is $Q \parallel Q_{RUN}$. Now $\delta^*(P)$ is deadlock free by (iv), so we infer the deadlock freedom of $\delta^*(P) \parallel \delta^*(P)_{RUN}$. Furthermore, we have that $traces(\delta^*(P)_{RUN}) \subseteq traces(\delta^*(P))$, so we deduce that $(\delta^*)^2(P)$ is defined. Using (ii) and (iii) in (i) then yields $(\delta^*)^2(P) \sqsubseteq P$.

□

Lemma 5.4.3 If P is deadlock free, then the following conditions hold:

(i) $\delta^*(P)$ is defined.

(ii) $traces(\delta^*(P)) = traces(P)$.

(iii) $(s, X) \in \mathcal{F}[\![P]\!] \Leftrightarrow (s, \alpha P - X) \notin \mathcal{F}[\![\delta^*(P)]\!] \wedge s \in traces(P)$.

(iv) $(\delta^*)^2(P) = P$.

Proof.

(i) Follows from the consideration of the process P_{RUN} as defined in the proof of Lemma 5.4.1 (vi).

(ii) The properties of P_{RUN} yield that $traces(P) \subseteq traces(\delta^*(P))$. Then from Lemma 5.4.1 (iii) we conclude that $traces(\delta^*(P)) = traces(P)$.

(iii) One direction of the implication is clear from Lemma 5.4.1 (ii). Now let us assume that $(s, X) \notin \mathcal{F}[\![P]\!]$ when $(s, \alpha P - X) \notin \mathcal{F}[\![\delta^*(P)]\!]$ and $s \in traces(P)$, for the sake of contradiction. Consider the process Q defined by the failure set

$$\mathcal{F}[\![P_{RUN}]\!] \cup \{(s, Y) \mid Y \subseteq (\alpha P - X) \cup (\alpha P - initials(P))\} \quad .$$

Then $P \parallel Q$ is deadlock free and $traces(Q) \subseteq traces(P)$, so it follows that $\delta^*(P) \sqsubseteq Q$. But this shows that $(s, \alpha P - X) \in \mathcal{F}[\![\delta^*(P)]\!]$, which is a contradiction.

(iv) From Lemma 5.4.1 (vi) we have that $(\delta^*)^2(P) \sqsubseteq P$. Now say there is a failure $(s, X) \in \mathcal{F}[\![(\delta^*)^2(P)]\!] - \mathcal{F}[\![P]\!]$. Note that Lemma 5.4.1 (iii) shows that

$$traces((\delta^*)^2(P)) \subseteq traces(\delta^*(P)) \subseteq traces(P) \quad .$$

Then applying (iii) once yields that $(s, \alpha P - X) \notin \mathcal{F}[\![\delta^*(P)]\!]$, and applying it again shows that $(s, X) \in \mathcal{F}[\![P]\!]$, which is a contradiction.

□

We can now prove the theorem that we were looking for.

Theorem 5.4.4 $(\delta^*, traces)$ is a checking pair for deadlock free processes such that all the checking processes themselves are deadlock free.

Proof. Let S be a deadlock free process. Firstly, if $S \sqsubseteq P$ then clearly $traces(P) \subseteq traces(S)$ and it follows from the monotonicity of $\parallel$ and from Lemma 5.4.1 (ii) that $\delta^*(S) \parallel P$ is deadlock free. Now say $\delta^*(S) \parallel P$ is deadlock free and $traces(P) \subseteq traces(S)$. Then by Lemma 5.4.2 (ii) we have $traces(P) \subseteq traces(\delta^*(S))$ and so using Lemma 5.4.1 (i) we obtain $(\delta^*)^2(S) \sqsubseteq P$. Thus applying Lemma 5.4.2 (iv) yields that $S \sqsubseteq P$. □

Again we need to be able to easily calculate the result of applying δ^*.

Lemma 5.4.5 Let S be a deadlock free process. Then

$$\mathcal{F}[\![\delta^*(S)]\!] = \{(s, X) \mid (s, \alpha S - X) \notin \mathcal{F}[\![S]\!] \wedge s \in traces(S)\} \quad .$$

Proof. Follows immediately from Lemma 5.4.2 (iii). □

Lemma 5.4.6 Let P, Q be deadlock free and $\mu x.F(x)$ be divergence free, where F is a continuous operator such that $F(P)$ is deadlock free whenever P is. Then the following equalities hold:

$$\begin{aligned}
&\mathcal{F}[\![\delta^*(a \to P)]\!] = \\
&\quad \{(\langle\rangle, X) \mid a \notin X\} \\
&\quad \cup \{(\langle a\rangle s, X) \mid (s, X) \in \mathcal{F}[\![\delta^*(P)]\!]\}
\end{aligned}$$

$$\begin{aligned}
&\mathcal{F}[\![\delta^*(P \,\Box\, Q)]\!] = \\
&\quad \{(\langle\rangle, X) \mid (\langle\rangle, X) \in \mathcal{F}[\![\delta^*(P)]\!] \cup \mathcal{F}[\![\delta^*(Q)]\!]\} \\
&\quad \cup \{(s, X) \mid s \neq \langle\rangle \wedge (s, X) \in (\mathcal{F}[\![\delta^*(P)]\!] \cap \mathcal{F}[\![\delta(Q)]\!]) \\
&\qquad\qquad \cup (\mathcal{F}[\![\delta(P)]\!] \cap \mathcal{F}[\![\delta^*(Q)]\!])\}
\end{aligned}$$

$$\mathcal{F}[\![\delta^*(P \sqcap Q)]\!] = (\mathcal{F}[\![\delta^*(P)]\!] \cap \mathcal{F}[\![\delta(Q)]\!]) \cup (\mathcal{F}[\![\delta(P)]\!] \cap \mathcal{F}[\![\delta^*(Q)]\!])$$

$$\mathcal{F}[\![\delta^*(\perp_{\mathrm{df}})]\!] = \mathcal{F}[\![RUN]\!]$$

and

$$\begin{aligned}
&\mathcal{F}[\![\delta^*(\mu x.F(x))]\!] = \\
&\quad \{(s, X) \mid (s, X) \in \bigcup_{n=0}^{\infty} \mathcal{F}[\![(\delta^* \circ F^n)(\perp_{\mathrm{df}})]\!] \wedge s \in \bigcap_{n=0}^{\infty} traces(F^n(\perp_{\mathrm{df}}))\}\,.
\end{aligned}$$

Proof. Similar to that of Lemma 5.3.6. □

Lemma 5.4.7 Suppose the definition of the process S uses only the following syntax:

$$P ::= SKIP \mid a \to P \mid P; P \mid P \,\Box\, P \mid P \sqcap P \mid p \mid \mu p.P$$

(where p denotes a process variable), but S contains no free process variables. Further suppose S is divergence free, has every occurrence of $SKIP$ directly or indirectly followed by a ";", and if S contains a construct of the form $P \,\Box\, Q$ or $P \sqcap Q$ then

$$x \in initials(P) \cap initials(Q) \;\Rightarrow\; P \underline{\text{ after }} \langle x\rangle = Q \underline{\text{ after }} \langle x\rangle.$$

Then the following properties hold:

$$\begin{aligned}
\delta^*(SKIP) &= SKIP \\
\delta^*(a \rightarrow P) &= a \rightarrow \delta^*(P) \\
\delta^*(P;Q) &= \delta^*(P);\delta^*(Q) \\
\delta^*(P \,\Box\, Q) &= \delta^*(P) \sqcap \delta^*(Q) \\
\delta^*(P \sqcap Q) &= \delta^*(P) \,\Box\, \delta^*(Q) \\
\delta^*(\mu p.P) &= \delta^*([(\mu p.P)\backslash p]P)
\end{aligned}$$

Proof. This is straightforward except perhaps for the cases when S is of the form $P \,\Box\, Q$ or $P \sqcap Q$. In these cases we assert that if $(s,X) \in \mathcal{F}[\![\delta^*(P)]\!]$ and $s \neq \langle\rangle$ then $(s,X) \in \mathcal{F}[\![\delta(Q)]\!]$. So assume for the sake of contradiction that $(s,X) \in \mathcal{F}[\![\delta^*(P)]\!] - \mathcal{F}[\![\delta(Q)]\!]$ where $s = \langle a \rangle t$. Then we see that $(s, \alpha S - X) \in \mathcal{F}[\![Q]\!]$ by using Lemma 5.3.3 (ii) and that $s \in traces(P)$ by Lemma 5.4.1 (iii). Hence $a \in initials(P) \cap initials(Q)$ which shows that P $\underline{\text{after}}$ $\langle a \rangle = Q$ $\underline{\text{after}}$ $\langle a \rangle$ and thus $(s, \alpha S - X) \in \mathcal{F}[\![P]\!]$. But this implies $\delta^*(P) \parallel P$ is not deadlock free, contradicting Lemma 5.4.1 (ii). The equalities concerning S then readily follow from Lemma 5.4.5 and the fact just proved. $\Box$

It might interest the reader to calculate $\delta^*(BUFF\text{-}S)$ where $BUFF\text{-}S$ is as defined in Example 5.3.1.

It is also of interest to note that the theory of δ^* seems to tie in quite closely with the theory in [Brin], although we have not yet investigated this formally. $\Box$

The example below identifies a class of processes (specifications) which are particularly easy to handle via checking pairs.

Example 5.4.4 Consider the following definition of a type of process.

Definition 5.4.2 A process P is said to be *simple* if

$$(s,X) \in \mathcal{F}[\![P]\!] \;\Leftrightarrow\; s \in traces(P) \;\wedge\; initials(P \;\underline{\text{after}}\; s) \not\subseteq X.$$

Thus a simple process is one which can refuse any proper subset of its initials after a trace. It is apparent that a simple process is also deadlock free. The following result shows the form that many simple process take.

Lemma 5.4.8 Let P, Q be simple and $\mu x.F(x)$ be divergence free, where F is a continuous operator such that $F(P)$ is simple whenever P is simple. Then $a \rightarrow P$, $P \sqcap Q$ and $\mu x.F(x)$ are all simple.

Proof. Follows from the definitions of the operators, and for $\mu x.F(x)$, by a similar argument to that in the proof of Lemma 5.3.6. □

For the purposes of the next theorem, recall the definition of Q_{RUN} given in the proof of Lemma 5.4.2 (vi).

Theorem 5.4.9 $((-)_{RUN}, traces)$ is a checking pair for simple processes.

Proof. If S is a simple process we know that it is deadlock free and hence that $\delta^*(S)$ exists. Using Lemma 5.4.5 we deduce that $\delta^*(S) = S_{RUN}$ and thus Theorem 5.4.4 completes the proof. □

The practical importance of simple processes rests on the following fact.

Lemma 5.4.10 Suppose S and P are processes such that S is simple and $traces(P) \subseteq traces(S)$. Then $S_{RUN} \parallel P$ is deadlock free if and only if P is deadlock free.

Proof. If $traces(P) \subseteq traces(S)$ and $S_{RUN} \parallel P$ is deadlock free then we see $S \sqsubseteq P$, and thus the deadlock freedom of P follows from that of S. Now say $traces(P) \subseteq traces(S)$, P is deadlock free and assume, for the sake of contradiction, that $S_{RUN} \parallel P$ is not deadlock free. Then there exists a failure $(s, X) \in \mathcal{F}[\![P]\!]$ such that $(s, \alpha S - X) \in \mathcal{F}[\![S_{RUN}]\!]$. From the definition of S_{RUN} we see $initials(S_{RUN} \underline{\text{ after }} s) \subseteq X$. As $traces(S_{RUN}) = traces(S)$ and $traces(P) \subseteq traces(S)$, we now deduce that

$$initials(P \underline{\text{ after }} s) \subseteq initials(S_{RUN} \underline{\text{ after }} s).$$

But then $initials(P \underline{\text{ after }} s) \subseteq X$ contradicting the deadlock freedom of P. □

Thus when checking a process to a simple process specification, having established the partial correctness condition, we then only need to establish the deadlock freedom of the process being checked. □

The proof technique that corresponds to the Checking theorem but that is concerned with checking pairs is now stated as follows.

Theorem 5.4.11 (Checking Pair Theorem) Let $(\mathcal{C}, \mathcal{R})$ be a checking pair for $\mathcal{S}$ and suppose $S \in \mathcal{S}$ and $V = \langle P_1, \ldots, P_n \rangle$ is some network. Further suppose that

- the process $PAR(V)\backslash(\alpha V - \alpha S)$ is divergence free,

- $traces(PAR(V)\backslash(\alpha V - \alpha S)) \subseteq \mathcal{R}(S)$, and
- the network $\langle \mathcal{C}(S), P_1, \ldots, P_n \rangle$ is deadlock free.

Then V implements the specification S.

Proof. Similar to that for Theorem 5.3.8. □

Again we give an example that illustrates its use. It is based very loosely on an example in [H2].

Example 5.4.5 Define processes

$$BUTTON = press \rightarrow \left(\begin{array}{l} (l.press \rightarrow SKIP \parallel d.press \rightarrow SKIP) \\ \sqcap \\ l.press \rightarrow SKIP \end{array} \right); \\ release \rightarrow d.release \rightarrow BUTTON$$

$$\alpha BUTTON = \{press, d.press, l.press, release, d.release, l.release\}$$

$$LIGHT = (l.press \rightarrow SKIP \parallel l.close \rightarrow SKIP); \\ on \rightarrow (l.open \rightarrow SKIP \,\Box\, d.on \rightarrow l.open \rightarrow SKIP); \\ off \rightarrow LIGHT$$

$$\alpha LIGHT = \{on, off, l.press, l.open, l.close, d.on, l.release\}$$

$$DOOR = l.close \rightarrow \left(\begin{array}{l} \Box \left(\begin{array}{l} d.press \rightarrow SKIP \\ d.on \rightarrow SKIP \end{array} \right); open \rightarrow DOOR' \\ d.release \rightarrow open \rightarrow l.open \rightarrow SKIP \end{array} \right); \\ close \rightarrow DOOR$$

$$\alpha DOOR = \{open, close, l.open, l.close, d.press, d.release, d.on\}$$

where

$$DOOR' = (l.open \rightarrow SKIP) \parallel \left(\begin{array}{l} \Box \; d.release \rightarrow SKIP \\ \quad\; d.press \rightarrow d.release \rightarrow SKIP \end{array} \right)$$

and consider the network

$$LIFT = \langle BUTTON, LIGHT, DOOR \rangle \quad .$$

The network represents part of a lift system with faulty components. The button of the lift can be pressed or released, in that order, and when either of these occur signals are sent to the light and door to that effect. However, as the button is faulty, the signal that the button has been pressed is sometimes not transmitted to the door and the light is never informed about the release of the button. The light can come on or off and is initially switched off. It awaits the receipt of signals telling it that the door has closed and the button has been pressed, in any order, before coming on. It may then inform the door that it is switched on, but is always willing to accept the news that the door has opened after which it turns itself off. The behaviour of the door is a little more complicated. It can, of course, open and close, and is closed initially. It firstly signals the light of its closure and then waits for messages from the other components. On receiving any such message, it opens and is now obliged to tell the light of this occurrence before closing. If the message received earlier was not one of the button being released, it ensures that as well as informing the light of its opening, that it itself is informed of the buttons release prior to shutting itself.

The network is far from an ideal model of a lift but serves rather as an example of a distributed system whose behaviour is not obviously apparent. We now prove that it does have some desirable properties, properties which we shall express in the form of specification processes. We shall divide these specifications into three classes and present the classes in the order decided by the degree of difficulty of their proof, with the easiest first.

$$\begin{array}{lcllcl} A_1 & = & press \rightarrow release \rightarrow A_1 & \alpha A_1 & = & \{press, release\} \\ A_2 & = & on \rightarrow off \rightarrow A_2 & \alpha A_2 & = & \{on, off\} \\ A_3 & = & open \rightarrow close \rightarrow A_3 & \alpha A_3 & = & \{open, close\} \end{array}$$

We observe from Lemma 5.4.8 that all the A_i are simple and it is easy to show that $traces(PAR(LIFT)\backslash(\alpha LIFT - \alpha A_i)) \subseteq traces(A_i)$, as only one process in the network need ever be examined for each particular A_i. The divergence freedom of $PAR(LIFT)\backslash(\alpha LIFT - \alpha A_i)$ follows by making the process containing the events in αA_i maximal in the order required for the use of Theorem 5.2.3. The Checking Pairs Theorem together with Theorem 5.4.9 and Lemma 5.4.10 then indicate that all that remains to be proved is the deadlock freedom of the network $LIFT$. That this must

be proved is intuitively obvious, as the specifications above assert that the individual components in the lift do not "jam" when connected together. The network is clearly triple-disjoint and busy, while its communication graph is fully connected and hence contains a cycle. The subnetwork $\langle BUTTON, LIGHT\rangle$ is conflict free due to the fact that the intersection of the alphabets of the two processes contains only one event. The other two subnetworks containing just pairs of processes require closer analysis, but these too are conflict free. Thus the whole network is conflict free. Now define functions

$$\begin{aligned} f_{BUTTON} &: \mathcal{F}[\![BUTTON]\!] \to \mathbf{Q} \\ f_{LIGHT} &: \mathcal{F}[\![LIGHT]\!] \to \mathbf{Q} \\ f_{DOOR} &: \mathcal{F}[\![DOOR]\!] \to \mathbf{Q} \end{aligned}$$

by putting

$$\begin{aligned} f_{BUTTON}((s,X)) &= 2(\,|s{\restriction}\{press, release, d.release\}| + |s{\restriction}\{d.release\}|\,) \\ f_{LIGHT}((s,X)) &= 2(\,|s{\restriction}\{on, l.open, off\}| + |s{\restriction}\{off\}|\,) + 1 \\ f_{DOOR}((s,X)) &= 2(\,|s{\restriction}\{l.close, open, close\}| + |s{\restriction}\{close\}|\,) \end{aligned}$$

where $\mathbf{N}$ has its usual order $>$. Note that the events mentioned in the definition of each function occur once on every cycle of the corresponding process and that the function increases along with the number of these events completed. Let σ be a state of the subnetwork $\langle BUTTON, LIGHT\rangle$ with s, t being the traces associated with the restriction of this state to $BUTTON$, $LIGHT$ respectively. First of all say $BUTTON$ has an ungranted request to $LIGHT$ in the state σ. As the only event shared by the two processes is $l.press$, $BUTTON$ must be willing to communicate this event and thus the value of its function is $8|s{\restriction}\{l.press\}| + 2$. The maximum possible value of the function for $LIGHT$ in such a state occurs when it is willing to do the event $l.open$, its value being $8|t{\restriction}\{l.press\}| - 5$. But $s{\restriction}\{l.press\} = t{\restriction}\{l.press\}$ so the former value is greater than the latter. Now say $LIGHT$ has an ungranted request to $BUTTON$. The value of the function for $LIGHT$ is then $8|t{\restriction}\{l.press\}| + 1$ and the maximum possible value of the function for $BUTTON$ is $8|s{\restriction}\{l.press\}| - 4$, this being the case when $BUTTON$ is ready to communicate the event $d.release$. Again the former value is greater than the latter. Employing similar arguments for the other cases, we may conclude that the network $LIFT$ is deadlock free by the Selected Edge Rule

with selected set of edges

$$\{\langle BUTTON, LIGHT\rangle, \langle LIGHT, BUTTON\rangle\} \quad .$$

We now proceed to another class of specifications. These processes are similar to the specifications above and hence the proofs required may be obtained in the same way.

$$\begin{array}{lcllcl} B_1 & = & press \rightarrow open \rightarrow B_1 & \alpha B_1 & = & \{press, open\} \\ B_2 & = & press \rightarrow on \rightarrow B_2 & \alpha B_2 & = & \{press, on\} \\ B_3 & = & release \rightarrow close \rightarrow B_3 & \alpha B_3 & = & \{release, close\} \\ B_4 & = & open \rightarrow off \rightarrow B_4 & \alpha B_4 & = & \{open, off\} \end{array}$$

Observe that this time more than one process in the network has to be examined to prove $traces(PAR(LIFT)\backslash(\alpha LIFT - \alpha B_i)) \subseteq traces(B_i)$.

Our final class of specifications is as follows.

$$C_1 = \left(\sqcap \begin{array}{l} (on \rightarrow SKIP \parallel open \rightarrow SKIP) \\ on \rightarrow open \rightarrow SKIP \end{array} \right); C_1$$

$$\alpha C_1 = \{on, open\}$$

$$C_2 = (off \rightarrow SKIP \parallel close \rightarrow SKIP); C_2$$

$$\alpha C_2 = \{off, close\}$$

The first specification states that either the switching on of the light and the opening of the door happen in parallel or the former precedes the latter. The second states that the switching off of the light and the closing of the door happen in parallel. Proving that $PAR(LIFT)\backslash(\alpha LIFT - \alpha C_i)$ is divergence free and that $traces(PAR(LIFT)\backslash(\alpha LIFT - \alpha C_i)) \subseteq traces(C_i)$ holds is again straightforward. Now using Lemma 5.4.7 we see that the $\delta^*(C_i)$ take the following form:

$$\delta^*(C_1) = \left(\sqcap \begin{array}{l} on \rightarrow open \rightarrow \delta^*(C_1) \\ open \rightarrow on \rightarrow \delta^*(C_1) \end{array} \right) \Box (on \rightarrow open \rightarrow \delta^*(C_1))$$

$$\delta^*(C_2) = (off \rightarrow close \rightarrow \delta^*(C_2)) \sqcap (close \rightarrow off \rightarrow \delta^*(C_2))$$

So it remains to prove the deadlock freedom of the networks obtained from the network $LIFT$ by adding the relevant checking process. The networks

$\langle \delta^*(C_i), BUTTON, LIGHT, DOOR \rangle$ are again conflict free but the commu nication graphs now contain more cycles. Define functions

$$\begin{aligned} f_{C_1} &: \mathcal{F}[\![\delta^*(C_1)]\!] \to \mathbf{Q} \\ f_{C_2} &: \mathcal{F}[\![\delta^*(C_2)]\!] \to \mathbf{Q} \end{aligned}$$

as follows:

$$f_{C_1}((s,X)) = 8|s \setminus \{open\}| + \left\{ \begin{array}{ll} \frac{3}{2} & \text{if } open \in X \\ \frac{5}{2} & \text{otherwise} \end{array} \right\}$$

$$f_{C_2}((s,X)) = 8|s \setminus \{off\}| + \left\{ \begin{array}{ll} \frac{7}{2} & \text{if } close \in X \\ \frac{9}{2} & \text{otherwise} \end{array} \right\}$$

Then using the Selected Edge Rule twice with the appropriate function above, the functions used earlier and the set of selected edges

$$\{\langle BUTTON, LIGHT\rangle, \langle LIGHT, BUTTON\rangle, \langle \delta^*(C_i), LIGHT\rangle, \langle LIGHT, \delta^*(C_i)\rangle\}$$

we may deduce that the augmented networks are deadlock free. Note that the variant functions for *LIFT* do need to be verified again in the context of these two new networks, as some events that were not in the vocabulary of *LIFT* are events in the vocabularies of the extended networks. Thus we conclude by the Checking Pairs Theorem that *LIFT* does implement the specifications C_1 and C_2. □

It is sometimes useful to show that a certain specification is not met by a network. Assuming the network meets the first two criteria in the Checking Pairs Theorem, we may then instead of proving the deadlock freedom of the new network consisting of the checking process and original network, attempt to detect a deadlock state in the new network by using for instance Theorem 3.1.1.

Conclusion

The major aim of this work has been to show how mainly local analysis can make proving deadlock freedom both clear and tractable. We believe that the methods we have described enable us to do this for a large proportion of real systems and also that the insight into deadlock produced by our investigations should be useful for producing new networks which are deadlock free by design. In our experience, the methods for proving deadlock freedom that have been of particular use are the Selected Edge Rule and the Selected Path Rule, the first because of its simplifying nature and the second due to its generality. An indication of this latter fact being that the global version of the Selected Path Rule is exactly as powerful as the proof rule in [CM1].

We have also shown that the problem of proving total correctness may be translated to one of proving deadlock freedom. This result, though interesting in itself, can provide us with a way to carry out a structured proof of total correctness via the use of the techniques developed above.

It should be fairly straightforward to transfer the results here to other models of concurrency such as CCS [M], and particularly to the extension given in [AH] which has an explicit representation of deadlock.

We now turn our attention to some directions that future research could perhaps take.

It does not seem clear how to extend our deadlock freedom rules which involve only local checking without making them unattractive to use. However, it would be of interest to see if there do exist any generalisations which keep within the spirit of the rules but are applicable to a wider class of hereditarily deadlock free networks.

It is a natural question to ask how our results have to be amended to cope with infinite networks of processes. One approach in this case would be to only allow well-founded partial orders in the preconditions of our Rules. This would allow us to deal with a network such as the infinite version of the one in Example 2.3.2. Another question that arises is how to deal with

the deadlock freedom of dynamic networks, which are networks in which processes can be created or killed. Work in this area is being carried out in [BR4]. Although the number of processes in such a network is not fixed over the evolution of the system, at any single instant the state of the network can be considered as corresponding to a state of a network with a fixed number of processes. Because of this, we envisage that it will be possible for our Rules to be adapted to such a class of networks.

Our concept of a deadlock transformation is a very general one and it would be useful to find more specific examples of such. In addition, it would be interesting to categorise this within some framework of deadlock preserving morphisms (see [He] for one obvious notion of such on transition systems). One particular aim of this would be to understand more clearly transformations that are less obvious, such as in Example 4.3.2.

Because of a technical difficulty, we were forced in Chapter 5 to assume that all processes had a finite alphabet. It should however be possible, using models such as in [R2] and [R3] that deal with unbounded non-determinism, to surmount this. For the use of our deadlock freedom techniques, this would entail a change in our definition of a state. Instead of maximal refusal sets (whose existence would now no longer be guaranteed) we would have to use some notion of a covering set.

Finally, it might be of practical benefit to develop checking pairs for specific classes of networks, which may in turn enable us to obtain convenient total correctness proof rules for these special classes.

Appendix

The Failures Model

We briefly recall here the material relevant to us from the *failures model* for CSP of [BR1]. We only present the semantics of the operators we have used and as for the laws that they obey, we refer the reader either to the reference above or [H1]. In addition, we give a semantics for the "interrupt" operator $\triangle$ of [H1].

In what follows, given a fixed set Σ, the set of finite sequences over Σ will be denoted by Σ^*, the powerset of Σ will be denoted by $\mathcal{P}(\Sigma)$ and the set of all finite subsets of Σ by $p(\Sigma)$. We shall use a,c to range over the set Σ, while s,t,u will range over Σ^* and X,Y,Z,A,B will range over $\mathcal{P}(\Sigma)$.

A process is regarded as an agent which communicates with its environment by performing *events* (or actions) from an *alphabet* Σ. In the model, a process is then represented by a pair $\langle F, D\rangle$. Here F is a set of *failures* or pairs (s, X), s being a *trace* (a finite sequence of communications from Σ) and X being a *refusal set* (a set of communications whose elements are drawn from Σ). We have that $(s, X) \in F$ if the process can carry out the events 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.

Formally, a pair $\langle F, D\rangle$ with

$$\begin{array}{lll} F & \subseteq & \Sigma^* \times \mathcal{P}(\Sigma) \\ D & \subseteq & \Sigma^* \end{array}$$

has to satisfy the following conditions to represent a process. For the failure

set F we must have

$$
\begin{array}{ll}
(\langle\rangle, \emptyset) \in F & \text{(F1)} \\
(st, \emptyset) \in F \;\Rightarrow\; (s, \emptyset) \in F & \text{(F2)} \\
(s, X) \in F \wedge Y \subseteq X \;\Rightarrow\; (s, Y) \in F & \text{(F3)} \\
(s, X) \in F \wedge (\forall c : Y.\, (s\langle c\rangle, \emptyset) \notin F) \;\Rightarrow\; (s, X \cup Y) \in F & \text{(F4)} \\
(\forall Y : p(X).\, (s, Y) \in F) \;\Rightarrow\; (s, X) \in F & \text{(F5)}
\end{array}
$$

while the condition on the divergence set D and the condition relating F and D take the form

$$
\begin{array}{ll}
s \in D \;\Rightarrow\; st \in D & \text{(D1)} \\
s \in D \;\Rightarrow\; (st, X) \in F & \text{(D2)} \quad .
\end{array}
$$

We shall denote the set of all such valid pairs $\langle F, D\rangle$ by N. For any set of failures F, we may define

$$
\begin{aligned}
traces(F) &= \{s \mid \exists X.\, (s, X) \in F\} \\
initials(F) &= \{c \mid \langle c\rangle \in traces(F)\} \\
refusals(F) &= \{X \mid (\langle\rangle, X) \in F\} \\
F \underline{\text{ after }} s &= \{(t, X) \mid (st, X) \in F\} \quad .
\end{aligned}
$$

There is a natural partial order on the set N given by

$$\langle F_1, D_1\rangle \sqsubseteq \langle F_2, D_2\rangle \;\Leftrightarrow\; F_2 \subseteq F_1 \wedge D_2 \subseteq D1$$

where $\sqsubseteq$ is termed the *non-determinism order*. With this order, N then becomes a complete semi-lattice.

To give a semantics to CSP, a mapping

$$\mathcal{N} : CSP \to N$$

is defined, where it is convenient to factor this function into two component functions,

$$\mathcal{F} : CSP \to \mathcal{P}(\Sigma^* \times \mathcal{P}(\Sigma))$$

dealing with the failure sets and

$$\mathcal{D} : CSP \to \mathcal{P}(\Sigma^*)$$

dealing with the set of divergences. Notation is often abused, by for instance writing $traces(P)$ rather than the more pedantic $traces(\mathcal{F}[\![P]\!])$.

We shall first only consider the standard syntax we have utilised (with the omission of recursion), as given by

$$P ::= STOP \mid SKIP \mid a \rightarrow P \mid P \,\Box\, P \mid P \sqcap P \mid P; P \mid P \backslash a \mid P_A \|_B P \quad .$$

Then

$$\begin{aligned}
\mathcal{D}[\![STOP]\!] &= \emptyset \\
\mathcal{D}[\![SKIP]\!] &= \emptyset \\
\mathcal{D}[\![a \rightarrow P]\!] &= \{\langle a \rangle s \mid s \in \mathcal{D}[\![P]\!]\} \\
\mathcal{D}[\![P \,\Box\, Q]\!] &= \mathcal{D}[\![P]\!] \cup \mathcal{D}[\![Q]\!] \\
\mathcal{D}[\![P \sqcap Q]\!] &= \mathcal{D}[\![P]\!] \cup \mathcal{D}[\![Q]\!] \\
\mathcal{D}[\![P; Q]\!] &= \{st \mid s \text{ is tick-free} \wedge s\langle\surd\rangle \in traces(P) \wedge t \in \mathcal{D}[\![Q]\!]\} \\
&\quad \cup \mathcal{D}[\![P]\!] \\
\mathcal{D}[\![P \backslash a]\!] &= \{(s \backslash a)t \mid \forall n : \mathbf{N}.\, s\langle a \rangle^n \in traces(P)\} \\
&\quad \cup \{(s \backslash a)t \mid s \in \mathcal{D}[\![P]\!]\} \\
\mathcal{D}[\![P_A \|_B Q]\!] &= \{st \mid s \in (A \cup B)^* \wedge \\
&\qquad ((s \restriction A \in \mathcal{D}[\![P]\!] \wedge s \restriction B \in traces(Q)) \vee \\
&\qquad\quad (s \restriction A \in traces(P) \wedge s \restriction B \in \mathcal{D}[\![Q]\!]))\}
\end{aligned}$$

and

$$\begin{aligned}
\mathcal{F}[\![STOP]\!] &= \{(\langle\rangle, X) \mid X \in \mathcal{P}(\Sigma)\} \\
\mathcal{F}[\![SKIP]\!] &= \{(\langle\rangle, X) \mid \surd \notin X\} \cup \{(\langle\surd\rangle, X) \mid X \in \mathcal{P}(\Sigma)\} \\
\mathcal{F}[\![a \rightarrow P]\!] &= \{(\langle\rangle, X) \mid a \notin X\} \cup \{(\langle a \rangle s, X) \mid (s, X) \in \mathcal{F}[\![P]\!]\} \\
\mathcal{F}[\![P \,\Box\, Q]\!] &= \{(\langle\rangle, X) \mid (\langle\rangle, X) \in \mathcal{F}[\![P]\!] \cap \mathcal{F}[\![Q]\!]\} \\
&\quad \cup \{(s, X) \mid s \neq \langle\rangle \wedge (s, X) \in \mathcal{F}[\![P]\!] \cup \mathcal{F}[\![Q]\!]\} \\
&\quad \cup \{(s, X) \mid s \in \mathcal{D}[\![P \,\Box\, Q]\!]\} \\
\mathcal{F}[\![P \sqcap Q]\!] &= \mathcal{F}[\![P]\!] \cup \mathcal{F}[\![Q]\!] \\
\mathcal{F}[\![P; Q]\!] &= \{(st, X) \mid (s\langle\surd\rangle, \emptyset) \in \mathcal{F}[\![P]\!] \wedge \\
&\qquad\qquad s \text{ tick-free} \wedge (t, X) \in \mathcal{F}[\![Q]\!]\} \\
&\quad \cup \{(s, X) \mid s \text{ tick-free} \wedge (s, X \cup \{\surd\}) \in \mathcal{F}[\![P]\!]\} \\
&\quad \cup \{(s, X) \mid s \in \mathcal{D}[\![P; Q]\!]\} \\
\mathcal{F}[\![P \backslash a]\!] &= \{(s \backslash a, X) \mid (s, X \cup \{a\}) \in \mathcal{F}[\![P]\!]\} \\
&\quad \cup \{(s, X) \mid (s, X) \in \mathcal{D}[\![P \backslash a]\!]\} \\
\mathcal{F}[\![P_A \|_B Q]\!] &= \{(s, X \cup Y \cup Z) \mid s \in (A \cup B)^* \wedge Z \subseteq \overline{A \cup B} \wedge \\
&\qquad\qquad X \subseteq A \wedge (s \restriction A, X) \in \mathcal{F}[\![P]\!] \wedge \\
&\qquad\qquad Y \subseteq B \wedge (s \restriction B, Y) \in \mathcal{F}[\![Q]\!]\} \\
&\quad \cup \{(s, X) \mid s \in \mathcal{D}[\![P_A \|_B Q]\!]\}
\end{aligned}$$

where the special event $\surd$ in Σ is used to denote successful termination.

All of the operations defined above are continuous with respect to the non-determinism ordering, and so we can appeal to the fixed point theorem of Knaster-Tarski to justify the existence of recursively defined processes. Accordingly, we add the to our CSP syntax a form of recursion

$$P ::= \mu p.P \mid p$$

where p ranges over a set of process identifiers.

This completes our summary of the failures model and we conclude the Appendix by presenting a semantics for the $\triangle$ operator.

$$\begin{aligned}
\mathcal{D}[\![P \triangle Q]\!] &= \mathcal{D}[\![P]\!] \cup \{stu \mid s \in traces(P) \wedge t \in \mathcal{D}[\![Q]\!]\} \\
\mathcal{F}[\![P \triangle Q]\!] &= \{(s,X) \mid (s,X) \in \mathcal{F}[\![P]\!] \wedge (\langle\rangle, X) \in \mathcal{F}[\![Q]\!]\} \\
&\quad \cup \{(st,X) \mid t \neq \langle\rangle \wedge (t,X) \in \mathcal{F}[\![Q]\!] \\
&\qquad\qquad s \text{ tick-free} \wedge s \in traces(P)\} \\
&\quad \cup \{(st,X) \mid s \in \mathcal{D}[\![P \triangle Q]\!]\} \quad .
\end{aligned}$$

References

[A1] K.R. Apt, *A Static Analysis of CSP Programs.* In Logics of Programs, Proceedings Springer-Verlag LNCS Vol. 164 (1983)

[A2] K.R. Apt, *Proving Correctness of CSP Programs: A Tutorial.* Technical Report 84-24, Université Paris 7 (1984)

[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)

[AH] L. Aceto and M. Hennessy, *Termination, Deadlock and Divergence.* Report 6/88, University of Sussex (December 1988)

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

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

[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-Verlag (1985)

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

[BR4] S.D. Brookes and A.W. Roscoe, *The Deadlock Freedom of Dynamic Networks.* To appear.

[Brin] E. Brinksma, *On the Existence of Canonical Testers.* Memorandum INF-87-5, University of Twente (1987)

[Broo] S.D. Brookes, *On the Axiomatic Treatment of Concurrency.* In Proceedings of the Pittsburgh Seminar on Concurrency, Springer-Verlag LNCS Vol. 197 (1985)

[C] Zhou Chao Chen, *The Consistency of the Calculus of Total Correctness for Communicating Processes.* Technical Monograph PRG-26, Oxford University Computing Laboratory, Programming Research Group (1982)

[CH] Zhou Chao Chen and C.A.R. Hoare, *Partial Correctness of Communicating Processes and Protocols.* Technical Monograph PRG-20, Oxford University Computing Laboratory, Programming Research Group (1982)

[CM1] K.M. Chandy and J. Misra, *Deadlock Absence Proofs for Networks of Communicating Processes.* Information Processing Letters Vol. 9, No. 4 (November 1979)

[CM2] K.M. Chandy and J. Misra, *Proofs of Networks of Processes.* IEEE Transactions on Software Engineering Vol. SE-7, No. 4 (July 81)

[DS] E.W. Dijkstra and C.S. Scholten, *A Class of Simple Communication Patterns.* Selected Writings on Computing EDW643, Springer-Verlag (1982)

[G] K.D. Günther, *Prevention of Deadlocks in Packet-Switched Data Transport Systems.* IEEE Transactions on Communications Vol. COM-29, No. 4 (April 1981)

[Ge] R. Gerth, *Transition Logic.* Proceedings of the 16th ACM STOC Conference (1983)

[H1] C.A.R. Hoare, *Communicating Sequential Processes.* Prentice-Hall International (1985)

[H2] C.A.R. Hoare, *Communicating Sequential Processes: Exercises and Answers.* Prentice-Hall International (1986)

[H3] C.A.R. Hoare, *A Calculus of Total Correctness for Communicating Processes.* Technical Monograph PRG-23, Oxford University Computing Laboratory, Programming Research Group (1981)

[He] W.H. Hesselink, *Deadlock and Fairness in Morphisms of Process Spaces.* Computing Science Notes CS 8603, Groningen University (1986)

[LG] G.M. Levin and D. Gries, *A Proof Technique for Communicating Sequential Processes.* Acta Informatica 15 (1981)

[LT] K.G. Larsen and B. Thomsen, *A Modal Process Logic.* Proceedings of LICS (1988)

[M] R. Milner, *A Calculus of Communicating Systems.* Springer-Verlag LNCS Vol. 92 (1980)

[MS1] P. M. Merlin and P.J. Schweitzer, *Deadlock Avoidance in Store-and-Forward Networks-I: Store-and-Forward Deadlock.* IEEE Transactions on Communications, Vol. COM-28, No. 3 (March 1980)

[MS2] P. M. Merlin and P.J. Schweitzer, *Deadlock Avoidance in Store-and-Forward Networks-II: Other Deadlock Types.* IEEE Transactions on Communications, Vol. COM-28, No. 3 (March 1980)

[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)

[OH] E.-R. Olderog and C.A.R. Hoare, *Specification-Oriented Semantics for Communicating Processes.* Technical Monograph PRG-37, Oxford University Computing Laboratory, Programming Research Group (1984)

[R1] A.W. Roscoe, *A Mathematical Theory of Communicating Processes.* Oxford University D.Phil. Thesis (1982)

[R2] A.W. Roscoe, *An Alternative Order for the Failures Model.* In "Two Papers on CSP", Technical Monograph PRG-67, Oxford University Computing Laboratory, Programming Research Group (1988)

[R3] A.W. Roscoe, *Unbounded Non-determinism in CSP.* In "Two Papers on CSP", Technical Monograph PRG-67, Oxford University Computing Laboratory, Programming Research Group (1988)

[R4] A.W. Roscoe, *Routing Messages Through Networks: An Exercise in Deadlock Avoidance.* In the Proceedings of the 7th Occam Users

Group Meeting and International Workshop on Parallel Programming of Transputer based Machines (1989)

[RD] A.W. Roscoe and N. Dathi, *The Pursuit of Deadlock Freedom.* Information and Computation, Vol. 75, No. 3 (December 1987)

[S] J. Sifakis, *Deadlocks and Livelocks in Transitions Systems.* In Proceedings of the Mathematical Foundations of Computer Science, Springer-Verlag LNCS, Vol. 88 (1980)

[So] N. Soundararajan, *Correctness Proofs of CSP Programs.* Theoretical Computer Science 24 (1983)

[T] A.S. Tanenbaum, *Network Protocols.* ACM Computing Surveys, Vol. 13, No. 4 (December 1981)

[Y] J. Yantchev, *Routing Messages in a Transputer Network.* In communications between the author and A.W. Roscoe (1988)