Deadlock Avoidance for Sequential Resource Allocation Systems:

Hard and Easy Cases

Mark Lawley, Assistant Professor*
School of Industrial Engineering, Purdue University
1287 Grissom Hall, West Lafayette, IN 47907-1287
e-mail malawley@ecn.purdue.edu, Phone (765) 494-5415
FAX (765) 494-1299

Spyros Reveliotis, Assistant Professor
School of Industrial and Systems Engineering
Georgia Institute of Technology
765 Ferst Drive, Atlanta, GA 30332-0205
Phone (404) 894-6608, FAX (404) 894-2301

* Corresponding Author

Abstract. Deadlock is a major problem for systems that allocate resources in real-time. The key issue in deadlock avoidance is whether or not a given resource allocation state is safe, that is, whether or not there exists a sequence of resource allocations that completes all processes. Although safety is established as NP-complete for certain broad resource allocation classes, newly emerging resource allocation scenarios often exhibit unique features not considered in previous work. In these cases, establishing the underlying complexity of the safety problem is essential for developing the best deadlock avoidance approach. This work investigates the complexity of safe resource allocation for a class of systems relevant in automated manufacturing. For this class, the resource needs of each process are expressed as a well defined sequence. Each request is for a single unit of a single resource and is accompanied by a promise to release the previously allocated resource. Manufacturing researchers have generally accepted that safety is computationally hard, and numerous sub-optimal deadlock avoidance solutions have been proposed for this class. Recent results, however, indicate that safety is often computationally easy. The objective of this paper is to settle this question by formally establishing the NP-completeness of safety for this class and investigating the boundary between the hard and easy cases. We discuss several special structures that lead to computationally tractable safety characteristics.

Key Words: deadlock avoidance, discrete event systems, flexible manufacturing

1. Introduction

Real-time resource allocation is a fundamental control responsibility in many types of automated systems. A resource allocation system (RAS) consists of a finite set of resources that must be allocated to competing processes. The processes enter the system, request, acquire, use, and release their required resources, and then exit the system. Many types of RASs are prone to deadlock, an insidious halting condition in which there exists a set of processes with every process in the set awaiting the allocation of resources held by other processes in the set. Deadlock is a well known problem in many technological areas such as computer operating systems, distributed databases, and automated manufacturing systems. Well known strategies for handling deadlock are (1) prevention, (2) detection-resolution, and (3) avoidance. Prevention restrains the request structure of processes so that deadlock is impossible. Because it limits process concurrency, prevention tends to be overly restrictive and typically achieves poor resource utilization. Detection-resolution approaches allow deadlock to occur and then concentrate on expedient resolution. This approach achieves the greatest flexibility in resource allocation at the cost of system stoppage and resolution procedures, which may involve aborting processes or the time consuming transport and reshuffling of physical entities. Avoidance uses current state information along with knowledge of process request and release structures to restrain the way resources are allocated so that deadlock never occurs. Avoidance achieves a middle ground in terms of allocation flexibility, being more flexible than prevention but less flexible than detection. It does not incur the cost of system stoppage and resolution and thus is the preferred method when the incremental increase in allocation flexibility does not merit the cost of allowing deadlock to occur. A more complete discussion of fundamental deadlock

concepts can be found in most books on computer operating systems, for example, see Silberschatz and Peterson (1991).

This paper deals exclusively with implementing deadlock avoidance methods in computer-based, real-time control of automated manufacturing systems. As given by Gold (1978), the key issue in deadlock avoidance is whether, for a given resource allocation state, say 'S', there exists a sequence of resource allocations that completes all processes. In other words, is 'S' safe? A deadlock avoidance policy (DAP) that restricts resource allocation based on the safety of the resulting state is maximally permissive and is therefore referred to as 'optimal'. In general, the optimal DAP cannot be implemented in a real-time controller since safety is established as NP-complete for certain broad classes of resource allocation systems.[1] This does not, however, imply that safety is hard in all resource allocation classes. Newly emerging classes often exhibit unique features, not considered in previous work, that can be used to make avoidance more permissive. When dealing with a new allocation class, it is therefore desirable to establish the complexity of the safety problem before preceding with the development of deadlock avoidance approaches.

More specifically, this work investigates the complexity of optimal deadlock avoidance for a class of resource allocation systems that has recently arisen in automated manufacturing, which we refer to as sequential single-unit (SU-RAS). For this class, the resource needs of each process are expressed as a well defined sequence. Each request is for a single unit of a single resource type and is accompanied by a promise to release the previously allocated resource. Thus, a process requests, holds, and releases only one resource at a time. Over the past decade, numerous researchers have proposed sub-optimal deadlock avoidance solutions for the SU-RAS, notably

[1] Real-time controllers do not have the temporal resources to solve computationally hard problems.

Banasak and Roszkowska (1988), Viswanadham, Narahari, and Johnson (1990), Banasak and Krogh (1990), Wysk, Yang, and Joshi (1991), Leung and Sheen (1993), Hsieh and Chang (1994), Fanti, Maione, Mascolo, and Turchiano (1997), and Lawley, Reveliotis, and Ferreira (1997, 1998a, 1998b). Although researchers have generally accepted that safety in the SU-RAS is computationally hard, recent results indicate that safety is often computationally easy (Fanti et al., 1997, Reveliotis, Lawley, and Ferreira 1997, Roszkowska and Jentink 1993, and Xing, Hu, and Chen 1996). The objective of this paper is to settle this question by formally establishing the NP-completeness of safety in the SU-RAS. We then want to investigate the boundary between the hard and easy cases and discuss several SU-RAS sub-classes for which safety is computationally easy.

The paper is organized as follows: section 2 defines the notion of a resource allocation system and briefly reviews earlier work on the complexity of deadlock avoidance. Section 3 formally defines the SU-RAS and establishes that, even for this more restricted class, the safety question remains NP-complete. Section 4 characterizes the boundary between hard and easy cases, while Section 5 discusses several special structures for the SU-RAS that exhibit polynomial safety characteristics. Section 6 provides a summary and conclusion.

2. Literature review

This section reviews relevant literature on the complexity of optimal deadlock avoidance. We begin by noting that a resource allocation system is composed of a set of resource types, $R=\{R_1, R_2, \ldots, R_m\}$ and a set of processes, $P=\{P_1,P_2,\ldots,P_n\}$. Each resource type has an associated 'capacity' indicating the number of identical instances of the resource type that exists in the system. Resources are either reusable or consumable, depending on the situation. Reusable resources are released (deallocated) by the process when it is finishes, whereas consumable

resources are never released. We assume that resource instances are not sharable, i.e., the same resource instance cannot be simultaneously allocated to multiple processes, and that resource instances cannot be preempted from the processes that are using them. A process makes one or more requests for the resources that it needs, and the allocation manager, often referred to as the 'banker', decides which requests to grant. The banker's objective is to allocate resources so that all requests are satisfied and all processes finish; a task that is complicated by overlapping, conflicting resource requirements. More specifically, the computational difficulty of the banker's problem depends on the structure, sequence, and magnitude of resource requests and releases, the capacities of resource types, and the reusability of the resource instances. We now discuss relevant literature on the complexity of the safety problem.

Because of its importance, we provide a brief discussion of the banker's algorithm (Habermann 1969), perhaps the most widely recognized and understood deadlock avoidance algorithm. The algorithm assumes that as each process enters the system, it declares the maximum number of each resource that it might ever require at one time. It further assumes that if a process is simultaneously allocated its stated maximum of each resource, then the process will terminate without additional requests and will release all reusable resources allocated to it. These resources then become available for allocation to other processes. The banker's algorithm avoids deadlock by allowing an allocation only if, in the resulting allocation state, the processes can be ordered so that the maximal resource needs of the i^{th} process, P_i, can be met by pooling available resources with those already held by P_i and those returned by processes $P_1,P_2,...,P_{i-1}$ upon their termination. The order defines a sequence in which all processes in the system can be terminated successfully. Clearly, this approach is conservative since this ordering mechanism limits the complexity of process interaction and because it assumes that each process

simultaneously requires its stated maximum. A process might not require its stated maximum or might not again require resources already requested, allocated, and released. On the other hand, banker's algorithm is polynomial (Holt 1972) and thus suitable for use in real time system controllers. Further, banker's algorithm provides a tool for identifying special structures with polynomial safety characteristics, that is, if a RAS class contains sufficient structure so that every safe state can be ordered by banker's algorithm, then banker's will accept all safe and reject all unsafe states, and thus safety is polynomial. We now briefly review two papers that identify several resource allocation structures for which banker's algorithm decides safety.

Gold (1978) analyzes a RAS in which a process, say P_u, makes a 'termination' request, ρ_{uo}, and a sequence of 'partial' requests $\{\rho_{u1}\ldots\rho_{uv}\}$. Each request is of the form 'give me these resources and I will return these resources'. The termination request is for the maximum number of each resource that the process will need to terminate, while a partial request is for a set of resources that will cause the process to release some other set of resources. The requests are sequenced so that satisfying one request, say ρ_{uj}, also satisfies all requests following ρ_{uj} in the sequence. If a partial request of a process is met, and the process frees some resources, then those resources will eventually have to be returned to the process before it can terminate. In other words, the resources freed upon allocation of a partial request are freed temporarily and must eventually be re-allocated to the process before it can terminate. Thus, the termination request must eventually be met, that is, the process must be allocated its stated maximum requirement of each resource before it can terminate.

After describing this model, Gold states and proves the conditions under which safety is NP-complete and polynomial for this resource allocation model. For systems with no partial requests, major results are: if each termination request is either a 'producer' (increases the

availability of every resource type) or ‘consumer’ (decreases the availability of every resource type), then safety is decided by banker’s algorithm and is therefore polynomial; on the other hand, if some termination requests are ‘mixed’ (neither producers nor consumers), then safety is NP-complete. For systems allowing partial requests, major results are: if resources are reusable, then safety is NP-complete; but if every request is either a ‘producer’ or ‘consumer’ and the profitability vectors of a process’ request sequence can be fully ordered,[2] then safety is decided by banker’s algorithm and is therefore polynomial.

Araki, Sugiyama, and Kasami (1977) address the complexity of deadlock avoidance for systems with sequential processes and reusable resources. The resource needs of each process are expressed as sequences of allocation and deallocation macros. An allocation macro represents the allocation of a set of requested resources to the process, while a deallocation macro represents the release of a set of resources by the process. Each resource type has a number of identical instances, and each process is in one of two states, either executing or awaiting resource allocation. The system state consists of the set of states of all processes along with a listing of unallocated resources.

The authors identify four significant restrictions on the process flows: (1) single unit, (2) single parameter, (3) straight line, and (4) nested. The single unit restriction implies that there is only one unit of each resource type in the system (note that this differs from our usage). Single parameter implies that only one resource may be specified by a macro, that is, only one resource may be requested or released at a time (note that a process is still able to accumulate resources through a sequence of requests with no releases). Straight line implies that there are no

[2] The profitability vector of a request is the resources returned less the resources requested.

conditional branches in the process flows, and finally, nested structure implies that any pair of resource scopes are either mutually disjoint or one scope subsumes the other in the process flow.

After describing the model and presenting these definitions, the paper states and proves conditions under which safety for this resource allocation model is NP-complete and polynomially computable. Major results are that under a nested structure with single parameter macros, safety is polynomial, whereas, if no nested structure is present, safety is NP-complete.

We draw two important distinctions between the work discussed above and that presented in the following sections. First, the NP-completeness proofs presented in the above literature rely on the fact that processes accumulate resources, that is, the resource allocation and use is conjunctive. This is not true for the resource allocation system considered in this paper, and thus, the SU-RAS represents a new subclass. Second, the polynomial subcases presented in the above literature have sufficient structure so that banker's algorithm determines safety. The additional structure of these cases causes the resource availability to increase monotonically as the search for a safe allocation sequence progresses. This monotonicity eliminates the need for backtracking and fosters a fundamental reduction in complexity. In contrast, banker's algorithm does not determine safety in the SU-RAS (Lawley et al., 1998a). The method developed in this paper to demonstrate the polynomial complexity for certain special cases of the SU-RAS relies on the fact that deadlock detection is polynomial while safety is NP-complete. These two results together imply the existence of unsafe states that do not exhibit deadlock. Thus, if some special structure eliminates the possibility of deadlock-free unsafe states, then safety must be polynomial. We discuss this point further in section 4 after establishing the NP-completeness of SU-RAS safety in section 3.

3. Complexity of safety in the SU-RAS

This section establishes the NP-completeness of the safety problem for the SU-RAS. We begin by giving a formulation of the safety question for the SU-RAS, call it SU-SAFE, and then present a polynomial reduction from the well known 3-SAT problem (Garey and Johnson 1978) to SU-SAFE.

SU-SAFE

Let R be a finite set of reusable resources and P be a set of processes. Each $R_t \in R$ can be allocated to only one process at a time, and each $P_u \in P$ requires a sequence of resources $\langle R_{u(1)}, R_{u(2)}, \ldots, R_{u(v)} \rangle$. P_u releases $R_{u(k)}$ upon being allocated $R_{u(k+1)}$. Thus, the sequence of allocations/deallocations for P_u is $\langle A(R_{u(1)})\ A(R_{u(2)})\ D(R_{u(1)})\ A(R_{u(3)})\ D(R_{u(2)}) \ldots A(R_{u(v)})\ D(R_{u(v-1)})\ D(R_{u(v)}) \rangle$ (where A denotes 'allocation' and D denotes 'deallocation'). The safety question is as follows: Given a set of resources, a set of partially completed processes, and their corresponding resource sequences, is there a sequence of resource allocations that completes every process and deallocates all resources?

Theorem 1 SU-SAFE is NP-complete.

Proof: SU-SAFE belongs to NP since we can verify a candidate sequence in polynomial time. To see this, let R_{max} represent the maximum length resource sequence required by processes in P. To determine whether a given sequence of resource allocations is safe or unsafe, we need only simulate the sequence and see if it completes all processes. This is $O(|R_{max}| x |P|)$, and thus, SU-SAFE has the property of polynomial verifiability, which indicates the problem belongs to NP.

We now provide a polynomial reduction from 3-SAT to SU-SAFE. We will use a variety of indexed symbols such as A, B, C, D, Y, etc., to represent resources in our RAS. Further, we will use the following formulation of 3-SAT provided by Araki et al., (1977):

Three Satisfiability (3-SAT)

Let $\chi=\{X_1, \underline{X}_1, X_2, \underline{X}_2, \ldots, X_m, \underline{X}_m\}$ be a set of literals and $C=C_1 \wedge C_2 \wedge \ldots \wedge C_n$ be a conjunction of clauses of the form $C_j=\langle Y_{j1} \vee Y_{j2} \vee Y_{j3} \rangle$ where $Y_{jk} \in \chi$, that is each Y_{jk} is one of the literals belonging to χ. Let $K \subseteq \chi$ such that (1) either $X_i \in K$ or $\underline{X}_i \in K$ but not both. Then C_j is 'true' if and only if $K \cap C_j \neq \varnothing$. The satisfiability question is as follows: Given χ and C, does there exist $K \subseteq \chi$ satisfying condition (1) such that $K \cap C_j \neq \varnothing \ \forall j=1 \ldots n$?

For each clause, $C_j \in C$, define seven processes P_{j1} to P_{j7}, which require the following resource sequences (refer to figure 1):

for j=1…n

$P_{j1} \langle C_{j1}, C_{j4}, C_{j5}, C_{j6}, C_{j8}, Y_{j1} \rangle$ $P_{j2} \langle C_{j2}, C_{j4}, C_{j5}, C_{j6}, C_{j8}, Y_{j2} \rangle$ $P_{j3} \langle C_{j3}, C_{j4}, C_{j5}, C_{j6}, C_{j8}, Y_{j3} \rangle$

$P_{j4} \langle C_{j7}, C_{j6}, C_{j8}, C_{j1}, C_{j2}, C_{j3} \rangle$ $P_{j5} \langle C_{j6}, C_{j8}, C_{j1}, C_{j2}, C_{j3} \rangle$ $P_{j6} \langle C_{j8}, C_{j1}, C_{j2}, C_{j3} \rangle$

$P_{j7} \langle A_j, C_{j8}, C_{j6}, C_{j7} \rangle$

Note that for each clause, C_j, we define twelve resources, $\{ C_{j1}, C_{j2}, C_{j3}, C_{j4}, C_{j5}, C_{j6}, C_{j7}, C_{j8}, Y_{j1}, Y_{j2}, Y_{j3}, A_j \}$. As noted in the definition of 3-SAT, $Y_{jk} \in \chi$, that is, Y_{jk} represents some X or $\underline{X}$ in χ. Next, for each pair of 3-SAT literals, X_i and $\underline{X}_i$, define a pair of processes, P_{i1} and P_{i2}, with the following resource sequences:

for i=1…m

$P_{i1} \langle X_i, B_i, D_1, D_2 \rangle$ $P_{i2} \langle \underline{X}_i, B_i, D_1, D_2 \rangle$

Note that for each i, we define resources $\{ X_i, \underline{X}_i, B_i \}$. We also define the two 'global' resources $\{D_1, D_2\}$. Next, define two processes, P_{01} and P_{02}, with resource sequences:

$P_{01} \langle D_1, A_1, A_2, A_3, \ldots, A_{n-1}, A_n \rangle$ $P_{02} \langle D_2, D_1, A_n, A_{n-1}, A_{n-2}, \ldots, A_2, A_1 \rangle$

Finally, advance every process to the state shown in figure 1. Note that the number of processes defined is 2m+7n+2, and the number of resources is 3m+10n+2 (m and n are given in the definition of 3-SAT), and thus, the reduction is polynomial.

Now, suppose C is satisfiable. Then there exists K such that $K \cap C_j \neq \varnothing$ for j=1…n. For each i=1…m, if $X_i \in K$, advance P_{i1} to B_i, otherwise advance P_{i2} to B_i. (Note that by definition of 3-SAT either X_i or $\underline{X}_i$ will be in K, but not both.) Figure 2 gives the resultant state. Thus, for every j=1…n, at least one resource in the set $\{Y_{j1}, Y_{j2}, Y_{j3}\}$ is free. For each j=1…n, we now show that the processes $\{P_{j1}, P_{j2}, P_{j3}, P_{j4}, P_{j5}, P_{j6}, P_{j7}\}$ can be advanced to a point where A_j is released without incurring unsafeness. We consider three cases:

Case 1 $Y_{j1} \hat{I} K \cap C_j$

If Y_{j1} is free, advance processes $\{P_{j1}, P_{j2}, P_{j3}, P_{j4}, P_{j5}, P_{j6}, P_{j7}\}$ as follows: P_{j1} to C_{j4} to C_{j5}; P_{j2} to C_{j4}; P_{j6} to C_{j1} to C_{j2}; P_{j5} to C_{j8} to C_{j1}; P_{j1} to C_{j6} to C_{j8} to Y_{j1} and out of the system; P_{j2} to C_{j5}; P_{j3} to C_{j4}; P_{j6} to C_{j3} and out of the system; P_{j5} to C_{j2} to C_{j3} and out of the system; P_{j4} to C_{j6} to C_{j8} to C_{j1} to C_{j2} to C_{j3} and out of the system; P_{j7} to C_{j8} to C_{j6} to C_{j7} and out of the system. Figure 3(a) shows the resulting state. Note that every P_j has finished except P_{j2} and P_{j3}, and that A_j is free.

Case 2 $Y_{j2} \hat{I} K \cap C_j$

If Y_{j2} is free, advance processes $\{P_{j1}, P_{j2}, P_{j3}, P_{j4}, P_{j5}, P_{j6}, P_{j7}\}$ as follows: P_{j2} to C_{j4} to C_{j5}; P_{j1} to C_{j4}; P_{j6} to C_{j1} to C_{j2}; P_{j5} to C_{j8} to C_{j1}; P_{j2} to C_{j6} to C_{j8} to Y_{j2} and out of the system; P_{j1} to C_{j5}; P_{j3} to C_{j4}; P_{j6} to C_{j3} and out of the system; P_{j5} to C_{j2} to C_{j3} and out of the system; P_{j4} to C_{j6} to C_{j8} to C_{j1} to C_{j2} to C_{j3} and out of the system; P_{j7} to C_{j8} to C_{j6} to C_{j7} and out of the system. Figure 3(b) shows the resulting state. Note that every P_j has finished except P_{j1} and P_{j3}, and that A_j is free.

Case 3 $Y_{j3} \hat{I} K\Ç C_j$

If Y_{j3} is free, advance processes $\{P_{j1}, P_{j2}, P_{j3}, P_{j4}, P_{j5}, P_{j6}, P_{j7}\}$ as follows: P_{j3} to C_{j4} to C_{j5}; P_{j1} to C_{j4}; P_{j6} to C_{j1}; P_{j5} to C_{j8}; P_{j3} to C_{j6}; P_{j1} to C_{j5}; P_{j2} to C_{j4}; P_{j6} to C_{j2} to C_{j3} and out of the system; P_{j5} to C_{j1} to C_{j2} to C_{j3} and out of the system; P_{j3} to C_{j8} to Y_{j3} and out of the system; P_{j4} to C_{j6} to C_{j8} to C_{j1} to C_{j2} to C_{j3} and out of the system; P_{j7} to C_{j8} to C_{j6} to C_{j7} and out of the system. Figure 3(c) shows the resulting state. Note that every P_j has finished except P_{j1} and P_{j2}, and that A_j is free.

Thus, for j=1…n, it is possible to complete all processes in the set $\{P_{j4}, P_{j5}, P_{j6}, P_{j7}\}$, thus releasing resource A_j. Furthermore, it is possible to complete at least one of $\{P_{j1}, P_{j2}, P_{j3}\}$, with the remaining processes being in one of the states of figure 3.

Since A_j is now free for j=1…n, advance P_{01} to $A_1,A_2,\ldots,A_n$ and out of the system. Next, advance P_{02} to $D_1,A_n,A_{n-1},\ldots,A_1$ and out of the system. Now, since D_1 and D_2 are free, each P_{i1} and P_{i2} can be advanced to D_1, D_2, and out of the system for i=1…m, thus releasing all X_i and $\underset{\sim}{X}_i$ resources. Thus, every resource in the set $\{Y_{j1},Y_{j2}, Y_{j3}\}$ is free for j=1…n. For j=1…n, advance all remaining P_{jk}'s on C_{j5} to C_{j6},C_{j8},Y_{jk}, and out of the system. Then advance all remaining P_{jk}'s on C_{j4} to C_{j5}, C_{j6}, C_{j8}, Y_{jk}, and out of the system. Since all processes are completed and all resources are deallocated, the state of figure 1 is safe. Thus, satisfiable implies safe.

Now suppose C is not satisfiable. (We show the contrapositive of safe implies satisfiable, that is, not satisfiable implies not safe.) If C is not satisfiable, then for every $K \subseteq \chi$, there exists C_j such that $K \cap C_j = \varnothing$. Without loss of generality, consider a given K and i=1…m. If $X_i \in K$, advance P_{i1} to B_i, otherwise advance P_{i2} to B_i. Starting from the state of figure 2, we show that if P_{j7} releases A_j before a Y_{jk} is available, then deadlock results. Note that if P_{j7} advances to C_{j8} (and releases A_j) before P_{j4} advances to C_{j1}, then deadlock involving P_{j7} and P_{j4} is inevitable. Thus, we must advance processes so that P_{j4} can gain C_{j1}. Starting from the state given in figure

2 and without loss of generality, advance the processes as follows: P_{j1} to C_{j4} to C_{j5}; P_{j2} to C_{j4}; P_{j6} to C_{j1} to C_{j2}; P_{j5} to C_{j8} to C_{j1}; P_{j4} to C_{j6} to C_{j8}; P_{j1} to C_{j6}; P_{j2} to C_{j5}; P_{j3} to C_{j4}; P_{j6} to C_{j3} and out of the system; P_{j5} to C_{j2} to C_{j3} and out of the system; P_{j4} to C_{j1} to C_{j2} to C_{j3} and out of the system. These advancements yield the state of figure 4. Note that P_{j1} cannot proceed beyond C_{j8} since none of the Y_{jk}'s are available. At this point, if P_{j7} advances to C_{j8} and releases A_j, then it deadlocks with P_{j1}. Thus, if $K \cap C_j = \varnothing$, A_j must not be released, otherwise P_{j7} becomes involved in deadlock.

Thus, in the state of figure 2, there exists at least one set of processes $\{ P_{j1}, P_{j2}, P_{j3}, P_{j7} \}$ that cannot be completed until additional Y_{jk}'s are released. Furthermore, the corresponding A_j will not be released. To release additional Y_{jk}'s, we must free some B_i resources by advancing P_{01}. If $K \cap C_1 = \varnothing$, then A_1 is not available, P_{01} cannot be advanced, no other Y_{jk}'s can be released, at least one set of processes of the form $\{ P_{j1}, P_{j2}, P_{j3}, P_{j7} \}$ cannot be completed, and the system is unsafe. If A_1 is available, advance P_{01} to A_1. We must now advance P_{02} to D_1, for if we advance any P_i from B_i to D_1, it deadlocks with P_{02}. Our only next choice is to advance P_{02} to A_n, thus freeing D_1. If $K \cap C_n = \varnothing$, then A_n is not available, P_{02} cannot be advanced, no other Y_{jk}'s can be released, at least one set of processes of the form $\{ P_{j1}, P_{j2}, P_{j3}, P_{j7} \}$ cannot be completed, and the system is unsafe. If A_n is available, advance P_{02} to A_n. D_1 and D_2 are now free, so all P_i processes can finish, one at a time, and all X_i and $\underline{X}_i$ resources can be released. Thus, for j=1…n, all process sets $\{P_{j1}, P_{j2}, P_{j3}, P_{j4}, P_{j5}, P_{j6}, P_{j7} \}$ can be completed, and all A_j resources can be released. Only P_{01} and P_{02} remain to be completed (see figure 5). Unfortunately, P_{01} and P_{02} are headed for inevitable deadlock. To see this, note that P_{01} holds A_1 and requires the resource sequence $\langle A_2, A_3, \ldots, A_{n-1}, A_n \rangle$, while P_{02} holds A_n and requires the resource sequence $\langle A_{n-1}, A_{n-2}, \ldots, A_2, A_1 \rangle$. Thus, it is impossible to finish these two processes, and the system is unsafe.

Although Theorem 1 establishes the intractability of safety for the SU-RAS, other recent results show that the problem of deciding whether an allocation state is deadlock or deadlock-free is polynomial for the SU-RAS (see Reveliotis et al., 1997). Taken together, these results imply that the intractability of SU-SAFE is related to the existence of allocation states that are both deadlock-free and unsafe, as we saw in figure 5. The implications of this result are discussed more fully in the following section.

4. On the boundary between hard and easy cases

This section discusses the complexity boundary between classes of SU-RAS systems that exhibit tractable safety and those for which safety is intractable. We consider a subclass to be a subset of SU-RAS systems with some common feature that is not artificially constraining in measures of system size. For example, the set of SU-RAS systems with acyclic resource sequences forms a subclass, whereas the set of SU-RAS systems with three resource types does not.

Because detecting deadlock in the SU-RAS is a polynomial computation (Reveliotis et al., 1997), safety is computationally easy for those SU-RAS subclasses that exhibit no reachable[3] deadlock-free unsafe states. For these systems, every allocation state encountered under normal system operation is either safe or deadlock, and thus single step look-ahead for deadlock is the optimal deadlock avoidance policy. These observations lead to the following proposition:

Proposition 2 If SU-SAFE is intractable for a given class of SU-RAS, then that class exhibits deadlock-free unsafe states.

Five papers have appeared in the recent literature that discuss the existence of deadlock-free unsafe states (Fanti et al., 1997, 1998, Reveliotis et al., 1997, Roszkowska et .al, 1993, and Xing

[3] A state is 'reachable' if, starting from the state in which no resources are allocated, there exists a sequence of resource allocations and deallocations that takes the system to that state.

et al., 1996). The strongest conditions are given by Fanti et al., (1997,1998). The proposed procedure is based on the enumeration and analysis of cycles in a working procedure digraph, D_w. This is a directed graph with each resource serving as a node and a directed edge (R_i,R_j) being present if R_j immediately follows R_i in the resource sequence of some process. After constructing D_w for a given set of processes, the authors enumerate all cycles contained in D_w and establish a new digraph, D_w^2, with the vertices being the cycles of D_w. (A cycle of D_w is denoted with the symbol γ.) An edge (γ_i, γ_j) is present in D_w^2 if (1) the two cycles intersect at exactly one resource, say R_v, and (2) some process requires a sequence of resources R_u,R_v,R_w where $R_u \in \gamma_i$, $R_v \in \gamma_i \cap \gamma_j$, and $R_w \in \gamma_j$. Thus, in some system state, a process holds R_u in cycle γ_i and requests R_v in the intersection of γ_i and γ_j. Upon being allocated R_v, it releases R_u, enters cycle γ_j, and requests $R_w \in \gamma_j$. If this allocation of R_v completes the cycle, γ_j, then deadlock results. The basic structure of a deadlock-free unsafe state one step from deadlock is characterized as a critical cycle of D_w^2. A cycle of D_w^2, $\gamma_i^2 = \{\gamma_u, \gamma_{u+1}, \ldots, \gamma_v\}$, is critical if (1) $\text{card}\{ \gamma_u \cap \gamma_{u+1} \cap \ldots \cap \gamma_v \} = 1$, and (2) every pair of cycles in γ_i^2 intersects at exactly one resource, say R_u. This situation is illustrated in figure 6. Fanti et al., (1998) discusses the effect of multi-capacity resources and adds a third condition, namely (3) R_u must be single capacity. This implies that if every resource in the intersection of a set of cycles satisfying (1) and (2) is multi-capacity, then deadlock-free unsafe states will not exist. For example, in figure 6, if R_7 is multi-capacity, then the system will exhibit no deadlock-free unsafe state.

We emphasize that a critical cycle in D_w^2 is a necessary condition for the existence of deadlock-free unsafe states in the SU-RAS operational state space. The condition is not sufficient because the deadlock-free unsafe states implied by the presence of a critical cycle might not be reachable under normal system operation. Figure 7 provides a simple example. The graphs D_w

and D_w^2 for the SU-RAS of figure 7(a) are depicted in figure 7(c). Notice that D_w^2 contains a critical cycle $\{C_2,C_3\}$ that corresponds to the deadlock-free unsafe state of figure 7(b). The reader should be able to verify that this state is unreachable when the SU-RAS of figure 7(a) starts from the empty state and operates according to the assumptions stated in section 3.

Finally, the test described above requires exponential computation since it enumerates all the cycles in D_w and D_w^2. For smaller systems, this increased complexity might not be a very critical issue, particularly if addressed in an off-line mode. However, for larger more dynamic systems, we believe that a series of easily testable conditions is more practical in application. The following section provides a set of such conditions that can function as useful guidelines for the manufacturing system designer.

5. SU-RAS structures exhibiting polynomial safety

The objective of this section is to gather and categorize existing results from the field literature regarding polynomially computable conditions for the non-existence of deadlock-free unsafe states. Furthermore, it introduces two new conditions that do not appear elsewhere. We classify these polynomial structures into three different categories of features leading to reduced complexity: (1) resource capacity, RC, (2) sequence restrictions, SR, and (3) central buffering, CB.

To support our discussion, we make use of the resource allocation graph (RAG), a common structure in the deadlock literature. It is defined as follows: $RAG=\{R\cup P, A_r\cup A_a\}$ where R is the set of resource types, P is the set of processes, $A_r = \{(P_i,R_u) : P_i$ is requesting an instance of $R_u\}$, and $A_a = \{(R_v,P_k) :$ an instance of R_v is allocated to $P_k\}$. Recall that the SU-RAS uses reuseable resource types, each type having an associated 'capacity' indicating the number of identical instances of the resource type that exists in the system. Thus, the maximum out-degree of $R_v\in R$

is the capacity of R_v, C_v. Define the reachable set of vertices of $v \in \{R \cup P\}$ to be $R^c(v) = \{u : u \in \{R \cup P\}$ and RAG contains a directed path from v to u$\}$. Then, a knot in RAG is a set of vertices, $K \subseteq \{R \cup P\}$, such that $\forall\ v \in K$, $R^c(v) = K$. A knot can be thought of as a strongly connected component with no emerging arcs. It is easily established that the SU-RAS is deadlocked if and only if the associated RAG contains a capacitated[4] knot (Reveliotis et al., 1997).

Resource Capacity

The existence of deadlock-free unsafe states in the SU-RAS is closely related to the existence of resource types that have a single instance only. RC1 provides a capacity based sufficient condition for the non-existence of deadlock-free unsafe states. This result is given by Xing et al., (1996) and Reveliotis et al., (1997). It is also implied by the conditions developed in Fanti, Maione, and Turchiano (1998).

RC1. In the SU-RAS, if every resource type has capacity exceeding one, then deadlock-free unsafe states do not exist.

RC1 guarantees that if a SU-RAS has no single capacity resources, optimal deadlock avoidance is achieved through single step look-ahead for capacitated knots in RAG. It has significant impact for those SU-RAS systems where equipping each resource type with at least two units of capacity is economically feasible.

Sequence Restrictions

Deadlock-free unsafe states require complex interactions between process sequences. This section examines sequence restrictions that impose sufficient limitation on sequence interaction

[4] By capacitated, we mean that every resource in the knot is allocated to capacity.

so that deadlock-free unsafe states do not arise. The first of these restrictions, referred to as SR1, uses the notions of immediate predecessors and successors of a resource, defined as follows:

$pred_j$ ={R_u : R_u immediately precedes R_j in some resource sequence}, in words, some process holds R_u and requests the allocation R_j.

$succ_j$ = {R_v : R_v immediately follows R_j in some resource sequence}, in words, some process holds R_j and requests the allocation R_v.

This restriction was initially observed by Fanti et al., (1997) as a consequence of their characterization of deadlock-free unsafe states one step from deadlock. Here we provide an alternative proof for this result that is based on contradiction arising from the assumption of the existence of deadlock-free unsafe states one step away from deadlock in the RAS state space. The basic logic of this approach was first developed in Reveliotis et al., (1997) for proving result RC1, stated above, and it constitutes a generic scheme for establishing the results of this section.

SR1. In the SU-RAS, if for every $R_j \in R$, either $pred_j$ or $succ_j$ is a singleton, then deadlock-free unsafe states do not exist.

Proof: Assume that all terminal parts are completed and removed from the system. Further, suppose that the SU-RAS is in a deadlock-free unsafe state, s_o, one step away from deadlock and that for every $R_j \in R$, either $pred_j$ or $succ_j$ is a singleton. Thus, if any available resource is allocated, deadlock results. Let π_u represent the set of processes requesting available resource type R_u in s_o, and let ρ_u represent the set of resources these processes hold in s_o. Let s_i be the state that results if R_u is allocated to $P_i \in \pi_u$. We have two cases:

Case 1, $|pred_u| = 1$: Suppose $pred_u=\{R_v\}$. This implies $\rho_u = \{R_v\}$, and thus every $P_i \in \pi_u$ holds an instance of R_v. Because (1) every process requesting R_u must be holding R_v, and (2) allocating R_u to $P_i \in \pi_u$ in state s_o must result in deadlock state s_i, we must have $R_v \in R^c(R_u)$ in s_i. Clearly,

however, R_v cannot be allocated to capacity in s_i, since P_i releases an instance of R_v upon receiving R_u. Thus, $R^c(R_u)$ has available capacity at R_v and no capacitated knot exists.

Case 2, $|\,succ_u\,| = 1$: Suppose $succ_u=\{R_w\}$. Because allocating R_u to any $P_i \in \pi_u$ in state s_o must result in deadlock state s_i, we conclude (1) R_w is allocated to capacity in s_o, (2) $R_u \in R^c(R_w)$ in s_i, and (3) $\rho_u \cap R^c(R_w)=\varnothing$ in s_o (for if $\rho_u \cap R^c(R_w)\neq\varnothing$ in s_o, $\exists\ P_i \in \pi_u$ in state s_o that releases $R_v \in \rho_u \cap R^c(R_w)$ upon being allocated R_u, and thus $R^c(R_w)$ has available capacity at R_v in s_i). To see the contradiction between (2) and (3), note that allocating R_u to $P_i \in \pi_u$ in state s_o deletes allocation arc (R_v,P_i), converts request arc (P_i,R_u) to allocation arc (R_u,P_i), and adds request arc (P_i,R_w). None of these affect $R^c(R_w)$ if $\rho_u \cap R^c(R_w)=\varnothing$, and thus no deadlock-free unsafe state exists. ÿ

Kumar and Ferreira (1998) combine RC1 and SR1 to yield the following result (which is established using a proof argument similar to that employed in the proof of SR1): For the SU-RAS, if for every $R_j \in R$, either $C_j>1$, $|pred_j\,|=1$, or $|succ_j|=1$, then deadlock-free unsafe states do not exist. The most common application of SR1 is in systems where every machine is equipped with input and output buffers. Parts coming to the machine enter the input buffer, then proceed to a processing location, and when finished, enter the output buffer where they await transport to the next required machine's input buffer (see Roszkowska et al., 1993 for a detailed analysis).

We next consider sequence restrictions imposed in reentrant flowline systems. Reentrant systems are important in semi-conductor manufacturing where certain processing sequences need to be repeated several times, see for example Narahari and Khan (1996). Although reentrant flowline deadlock has received some research attention (Lewis, Gurel, Bogdan, Doganalp, and Pastravanu, 1998), the following is, to the best of our knowledge, a new result.

Let the following restriction be referred to as the 'reentrant restriction': The resources can be ordered $\langle R_1,R_2,\ldots,R_m\rangle$ such that for every $R_j \in R$, $succ_j = \{R_{(j+1)}\} \cup \{R_v : j > v\}$. In words, the

resources can be ordered so that any process holding R_j requests either $R_{(j+1)}$ or R_v with $v<j$ for its next operation. We will say that process, P_i, requires a 'right' move if it holds R_j and requests $R_{(j+1)}$, and that P_i requires a 'left' move if it holds R_j and requests R_v such that $v<j$.

SR2. Given an SU-RAS with the reentrant restriction, if every left move is followed by at least one right move, then deadlock-free unsafe states do not exist.

Proof: Make the usual assumptions that that all terminal parts are completed and removed from the system, and that the SU-RAS is in a deadlock-free unsafe state, s_o, one step away from deadlock. Further, suppose that reentrant restrictions apply and that every left move is followed by at least one right move. If any available resource is allocated, deadlock results. Let π_u represent the set of processes requesting available resource type R_u in s_o, and let ρ_u represent the set of resources these processes hold in s_o. Let s_i be the state that results if R_u is allocated to $P_i \in \pi_u$. By the reentrant restriction, ρ_u and π_u can each be partitioned into two sets, $\rho_u = \rho_u' \cup \rho_u''$, where $\rho_u' = \{R_{(u-1)}\}$ and $\rho_u'' = \{R_v : R_v \in \rho_u \text{ and } v>u\}$, and $\pi_u = \pi_u' \cup \pi_u''$, where $\pi_u' = \{P_k : P_k \text{ holds } R_{(u-1)}\}$ and $\pi_u'' = \{P_i : P_i \text{ holds } R_v \in \rho_u''\}$. We now examine three cases.

Case 1: Suppose $\rho_u' = \varnothing$ and $\rho_u'' \neq \varnothing$ in s_o. As established in the proof of RC1, $R_u \notin R^c(R_u)$ in s_o, and $R_u \in R^c(R_u)$ in s_i. Let $R_v = \min\{ R_w : R_w \in \rho_u''\}$ and suppose that R_v is held by $P_i \in \pi_u''$. Allocating R_u to P_i causes the following changes to the RAG of s_o: (i) converting the request arc (P_i,R_u) to the allocation arc (R_u,P_i), (ii) deleting the allocation arc (R_v,P_i), and (iii) adding a new request arc for P_i, $(P_i, R_{(u+1)})$, since P_i cannot make two consecutive left moves. Adding the arcs (R_u,P_i), $(P_i,R_{(u+1)})$ can result in $R_u \in R^c(R_u)$ with $R^c(R_u)$ being a capacitated knot only if there is a pre-existing path in s_o from $R_{(u+1)}$ to R_u that does not include R_v (for if the path contains R_v, R_v will be reachable from R_u in state s_i, implying that $R^c(R_u)$ is not a capacitated knot). This implies

$\{R_{(u+1)}, \ldots, R_{(v-1)}\} \cap \rho_u'' \neq \varnothing$ (since $\rho_u' = \varnothing$), which contradicts our choice of P_i. Thus, a deadlock-free unsafe state one step from deadlock cannot have $\rho_u' = \varnothing$ and $\rho_u'' \neq \varnothing$.

Case 2: Suppose $\rho_u' \neq \varnothing$ and $\rho_u'' = \varnothing$ in s_o. Since $\rho_u' = \{R_{(u-1)}\}$, $\exists\ P_k \in \pi_u'$ such that P_k holds $R_{(u-1)}$ in s_o. Further, $\rho_u'' = \varnothing$ in s_o implies $\rho_u'' = \varnothing$ in s_i, since allocating R_u to P_k does not cause any additional request for R_u. Thus, any deadlock in s_i involving R_u must also involve $R_{(u-1)}$, since all processes requesting R_u will be holding $R_{(u-1)}$. But $R^c(R_u)$ will have available capacity at $R_{(u-1)}$ in s_i, since P_i releases an instance of $R_{(u-1)}$ upon allocation of R_u. Thus, a deadlock-free unsafe state one step from deadlock cannot have $\rho_u' \neq \varnothing$ and $\rho_u'' = \varnothing$.

Case 3: Suppose that $\rho_u' \neq \varnothing$ and $\rho_u'' \neq \varnothing$ in s_o. Let $P_i \in \pi_u''$ such that P_i holds $R_v = \min\{ R_w : R_w \in \rho_u''\}$ and let $P_k \in \pi_u'$ such that P_k holds $R_{(u-1)}$ in s_o. As noted in case 1, allocating R_u to P_i can result in $R_u \in R^c(R_u)$ with $R^c(R_u)$ being a capacitated knot only if there is a pre-existing path in s_o from $R_{(u+1)}$ to R_u that does not include R_v. This path must include $R_{(u-1)}$, since $\{R_{(u+1)}, \ldots, R_{(v-1)}\} \cap \rho_u'' = \varnothing$. Allocating R_u to P_k can result in $R_u \in R^c(R_u)$ with $R^c(R_u)$ being a capacitated knot only if there is a path in s_i from R_u to R_u that does not include $R_{(u-1)}$ (since $R_{(u-1)} \in R^c(R_u)$ implies $R^c(R_u)$ has available capacity at $R_{(u-1)}$). This implies that the path must include R_v. Clearly, these two conditions are in contradiction. Thus, a deadlock-free unsafe state one step from deadlock cannot have $\rho_u' \neq \varnothing$ and $\rho_u'' \neq \varnothing$.

Since we have enumerated all possibilities for ρ_u' and ρ_u'', we conclude that a deadlock-free unsafe state one step from deadlock does not exist.

As previously stated, SR2 has potential application in the semi-conductor industry where automated reentrant lines are commonly used (e.g., cluster tools). We believe the conditions imposed by SR2 are reasonably met in most reentrant systems.

Central Buffering

In this final section, we develop complexity results for two resource allocation models based on 'central buffering'. In automated manufacturing systems, central buffers are sometimes used to free up capacity on bottleneck machines. Note that parts finished with their currently allocated machine continue to occupy that machine's capacity until they are allocated capacity at their next required machine. This blocking effect can result in very poor system performance, particularly when processing times are highly variable. Under central buffering, such parts typically have the option of moving to and awaiting their next allocation at a centrally located buffer, if the buffer is not full. Properly used, this 'optional' central buffer enhances resource utilization and fosters better system throughput. Alternatively, some systems use a centralized material handler, such as a robot, to move parts from machine to machine. In this case, the material handler is equivalent to a 'requisite' central buffer, a central buffer to which every part must return after every operation.

In this section, we discuss how these central buffer models affect the complexity of deadlock avoidance. Let β represent the central buffer. Under the optional central buffer, the RAG has the following structure. Suppose that P_i holds resource R_v while requesting resource R_u. Then (P_i,R_u) in RAG implies (P_i,β) in RAG, in words, any requesting process also requests the central buffer. Allocating β to P_i results in the deletion of the request arc (P_i,β) and allocation arc (R_v,P_i), the addition of allocation arc (β,P_i), and, since β performs no processing function, the request arc (P_i,R_u) remains unchanged. For the requisite central buffer, (R_v,P_i) in RAG implies that (P_i,β) is the lone request of P_i in RAG. In words, if P_i is allocated a resource other than β, then its next required resource is β and β alone. The first result regarding the effect of SU-RAS central

buffering on the complexity of the optimal DAP appeared in Lawley (1999) and can be stated as follows:

CB1. For the SU-RAS under optional central buffering, deadlock-free unsafe states do not exist.

Thus, in systems with an optional central buffer, single step look-ahead for deadlock guarantees deadlock-free operation. Note that the capacity of the central buffer is not important and that it need not be treated in any special way, that is, it should be treated just like any other resource. CB2 provides the condition under which safety is polynomial for the requisite central buffer.

CB2. For the SU-RAS with requisite central buffering, if the capacity of the central buffer exceeds one, then deadlock-free unsafe states do not exist.

CB2 is directly implied by the result of Kumar et al., (1998) mentioned earlier, i.e., in the SU-RAS, if for every $R_j \in R$, either $C_j>1$, $|pred_j|=1$, or $|succ_j|=1$, then deadlock-free unsafe states do not exist. Note that for every resource other than the central buffer, we have $pred_j = succ_j = \{\beta\}$, and for the central buffer $C_\beta>1$.

CB1 and CB2 achieve optimal deadlock avoidance by ensuring that a 'swapping' mechanism is always available. While CB1 is most suitable for those systems where transfer times are small compared to processing times, CB2 applies to any system with a single centralized material handler, such as a robot moving parts about in a cell.

6. Conclusion

In this paper, we investigated the computational complexity of the safety question for the SU-RAS, a very important resource allocation model for automated manufacturing systems. After showing safety to be NP-complete, we established the existence of deadlock-free unsafe states as being a necessary (but not sufficient) condition for this intractability. We further established that,

using presently available methods, deciding whether or not a given SU-RAS instance or class exhibits hard safety characteristics is intractable due to the computational cost of establishing the existence of reachable deadlock-free unsafe states. Finally, we reviewed all currently known (polynomially identifiable) SU-RAS classes that exhibit no deadlock-free unsafe states, and thus admit polynomial optimal deadlock avoidance policies. From a practical standpoint, characterizing special structures that exhibit polynomial safety is an important research contribution since these structures serve as explicit guidelines in the system design process.

References

Araki, T., Sugiyama, Y., Kasami, T., and Okui, J., "Complexity of the Deadlock Avoidance Problem," *Proceedings of 2nd IBM Symposium on the Mathematical Foundations of Computer Science*, IBM Japan, Tokyo, pp. 229-252 (1977).

Banaszak, Z. and Roszkowska, E., "Deadlock Avoidance in Pipeline Concurrent Processes," *Podstawy Sterowania (Foundations of Control)*, Vol. 18, pp. 3-17 (1988).

Banaszak, Z. and Krogh, B., "Deadlock Avoidance in Flexible Manufacturing Systems with Concurrently Competing Process Flows," *IEEE Transactions on Robotics and Automation*, Vol. 6, No. 6, pp.724-734 (December 1990).

Fanti, M., Maione, B., Mascolo, S., and Turchiano, B., "Event-Based Feedback Control for Deadlock Avoidance in Flexible Production Systems," *IEEE Transactions on Robotics and Automation*, Vol. 13, No. 2, pp. 347-363 (June 1997).

Fanti, M., Maione, B., and Turchiano, B., "Event Control for Deadlock Avoidance in Production Systems with Multiple Capacity Resources," *Studies in Informatics and Control*, Vol. 7, No. 4, pp. 343-364 (December 1998).

Garey, M. and Johnson, D. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York (1979).

Gold, M., "Deadlock Prediction: Easy and Difficult Cases," *SIAM Journal of Computing*, Vol. 7, No. 3, pp. 320-336 (August 1978).

Haberman, A., "Prevention of System Deadlocks," *Communications of the ACM*, Vol. 12, No. 7, pp.373-377 (July 1969).

Holt, R., "Some Deadlock Properties of Computer Systems," *ACM Computing Surveys*, Vol. 4, No. 3, pp. 179-196 (September 1972).

Hsieh, F. and Chang, S., "Dispatching Driven Deadlock Avoidance Controller Synthesis for Flexible Manufacturing Systems," *IEEE Transactions on Robotics and Automation*, Vol. 10, No. 2, pp. 196-209 (April 1994).

Kumar, P. and Ferreira, P., "Scalable and Maximally Permissive Deadlock Avoidance for FMS," *Proceedings of the 1998 International Conference on Robotics and Automation*, Leuven, Belgium (May 1998).

Lawley, M., Reveliotis, S., and Ferreira, P., "FMS Structural Control and the Neighborhood Policy: Parts 1 and 2," *IIE Transactions*, Vol. 29, No. 10, pp. 877-899 (October 1997).

Lawley, M., Reveliotis, S., and Ferreira, P., "Application and Evaluation of Banker's Algorithm for Deadlock-Free Buffer Space Allocation in Flexible Manufacturing Systems," I*nternational Journal of Flexible Manufacturing Systems*, Vol. 10, No. 1, pp. 73-100 (February 1998a).

Lawley, M., Reveliotis, S., and Ferreira, P., "A Correct and Scalable Deadlock Avoidance Policy for Flexible Manufacturing Systems," *IEEE Transactions on Robotics and Automation*, Vol. 14, No. 5, pp. 796-809 (October 1998b).

Lawley, M., "Deadlock Avoidance for Production Systems with Flexible Routing," *IEEE Transactions on Robotics and Automation*, Vol. 15, No. 3, pp. 1-13 (June 1999).

Leung, Y. and Sheen, G., "Resolving Deadlocks in Flexible Manufacturing Cells," *Journal of Manufacturing Systems*, Vol. 12, No. 4, pp. 291-304 (1993).

Lewis, F., Gurel, A., Bogdan, S., Doganalp, A., and Pastravanu, O., "Analysis of Deadlock and Circular Waits Using a Matrix Model for Flexible Manufacturing Systems," *Automatica*, Vol. 34, No. 9, pp. 1083-1100 (September 1998).

Narahari, Y. and Khan, L., "Modeling Reentrant Manufacturing Systems with Inspection Stations," *Journal of Manufacturing Systems*, Vol.15, No.6, pp. 367-378 (1996).

Reveliotis, S., Lawley, M., and Ferreira, P., "Polynomial Complexity Deadlock Avoidance Policies for Sequential Resource Allocation Systems," *IEEE Transactions on Automatic Control*, Vol. 42, No. 10, pp. 1344-1357 (October 1997).

Roszkowska, E. and Jentink, J., "Minimal Restrictive Deadlock Avoidance in FMSs," *Proceedings of European Control Conference, ECC '93*, Vol. 2, pp. 530-534 (1993).

Silberschatz, A. and Peterson, G., *Operating Systems Concepts*, Addison-Wesley, Reading, MA (1991).

Viswanadham, N., Narahari, Y., and Johnson, T., "Deadlock Prevention and Deadlock Avoidance in Flexible Manufacturing Systems Using Petri Nets," *IEEE Transactions on Robotics and Automation*, Vol. 6, No. 6, pp. 712-723 (December 1990).

Wysk, R., Yang, N., and Joshi, S., "Detection of Deadlocks in Flexible Manufacturing Cells," *IEEE Transactions on Robotics and Automation*, Vol. 7, No. 6, pp. 853-859 (December 1991).

Xing, K., Hu, B., and Chen, H., "Deadlock Avoidance Policy for Petri Net Modeling of Flexible Manufacturing Systems with Shared Resources," *IEEE Transactions on Automatic Control*, Vol. 41, pp. 289-295 (February 1996).

Process Routes

P_{01} $\langle D_1, A_1, A_2, A_3, \ldots, A_{n-1}, A_n \rangle$
P_{02} $\langle D_2, D_1, A_n, A_{n-1}, A_{n-2}, \ldots, A_2, A_1 \rangle$

P_{i1} $\langle X_i, B_i, D_1, D_2 \rangle$
P_{i2} $\langle \underline{X}_i, B_i, D_1, D_2 \rangle$

P_{j1} $\langle C_{j1}, C_{j4}, C_{j5}, C_{j6}, C_{j8}, Y_{j1} \rangle$
P_{j2} $\langle C_{j2}, C_{j4}, C_{j5}, C_{j6}, C_{j8}, Y_{j2} \rangle$
P_{j3} $\langle C_{j3}, C_{j4}, C_{j5}, C_{j6}, C_{j8}, Y_{j3} \rangle$

P_{j4} $\langle C_{j7}, C_{j6}, C_{j8}, C_{j1}, C_{j2}, C_{j3} \rangle$
P_{j5} $\langle C_{j6}, C_{j8}, C_{j1}, C_{j2}, C_{j3} \rangle$
P_{j6} $\langle C_{j8}, C_{j1}, C_{j2}, C_{j3} \rangle$

P_{j7} $\langle A_j, C_{j8}, C_{j6}, C_{j7} \rangle$

Number of Processes: 2m+7n+2
Number of Resources: 3m+10n+2

Note that resources are represented as boxes. A process occupying a resource box is assumed to be holding that resource. An arc indicates the next resource required by the process. Cross-hatched resources are also allocated.

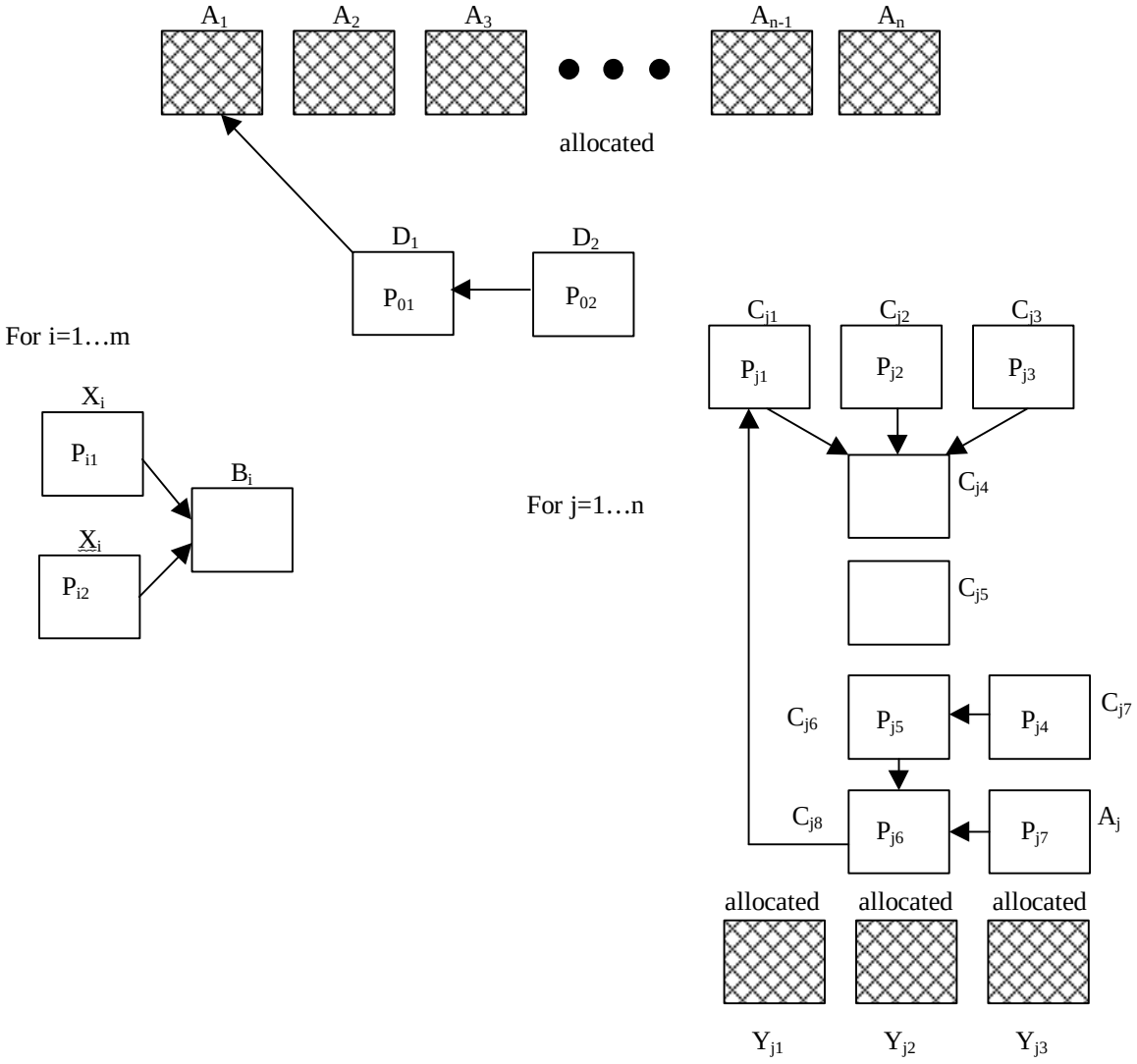


Figure 1. SU-RAS construction with partially completed processes

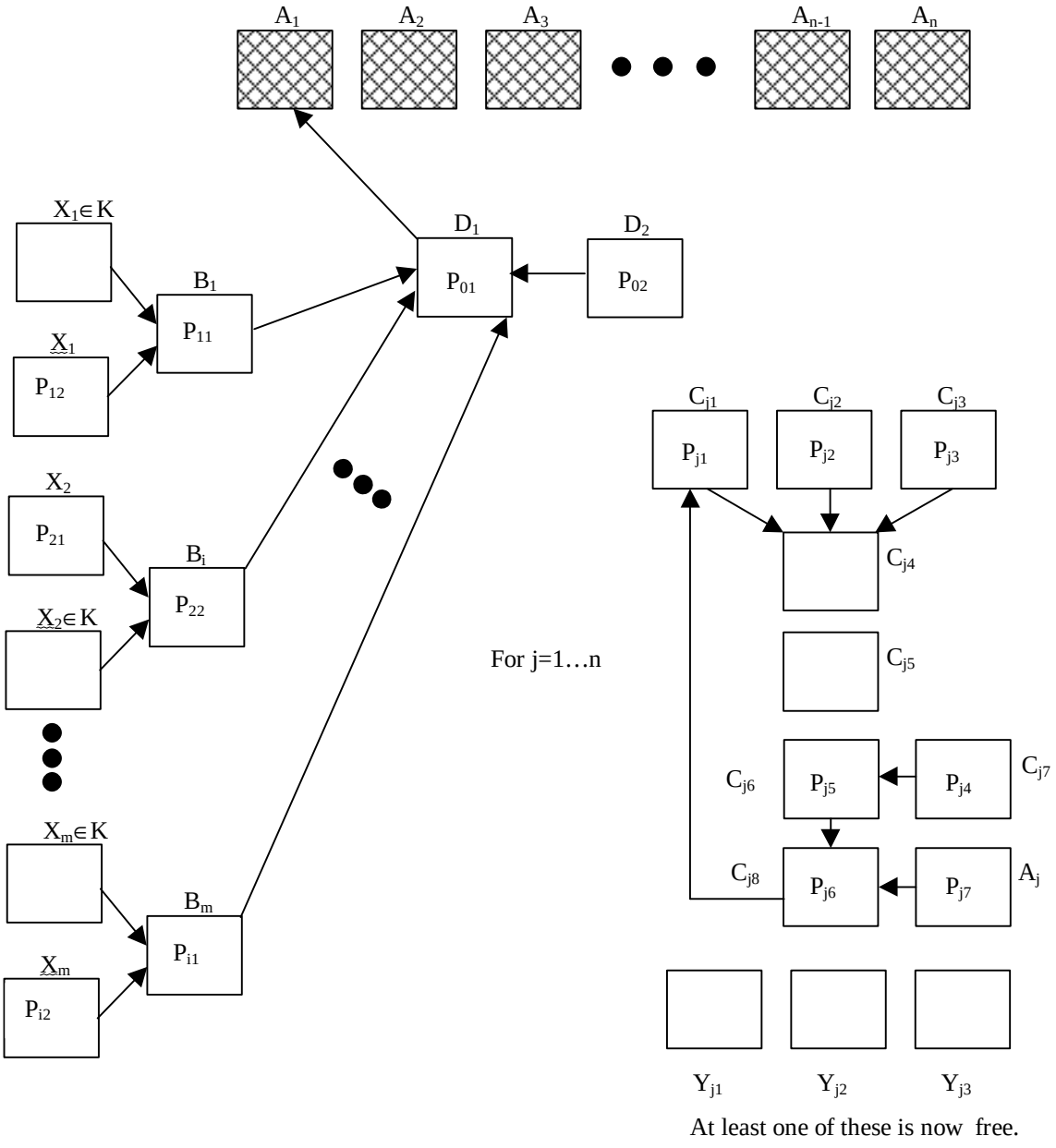
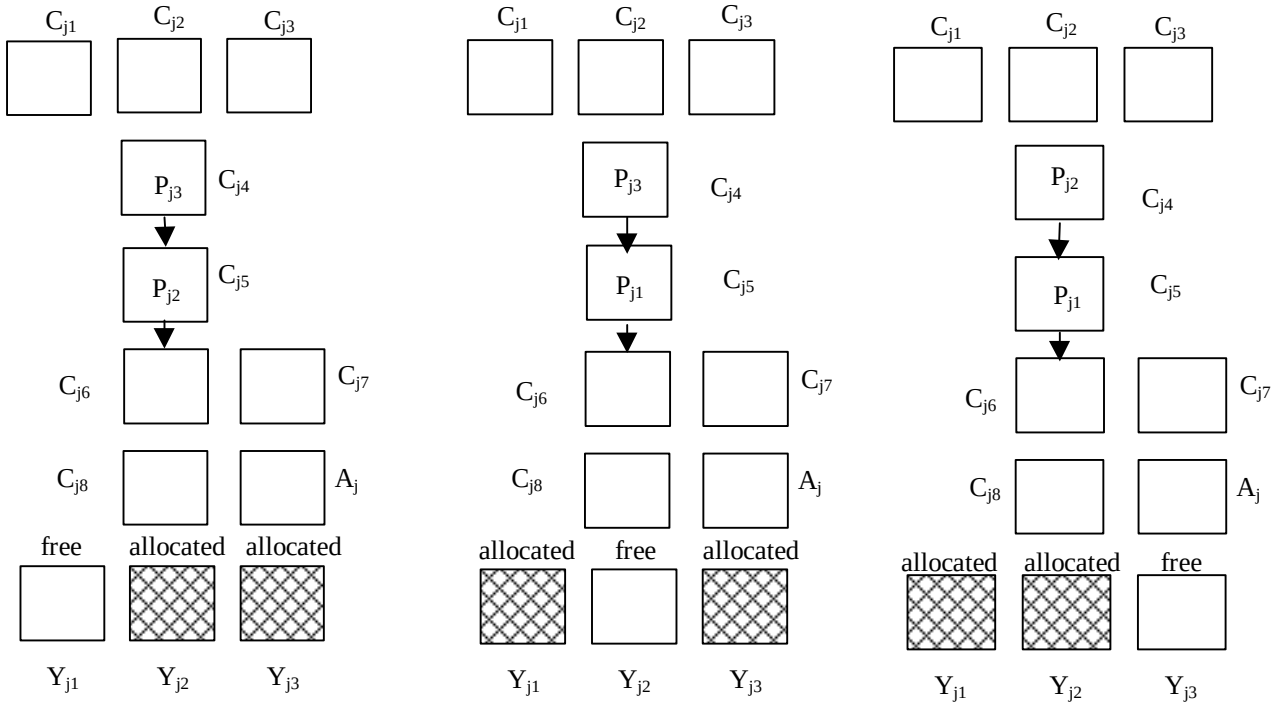


Figure 2. SU-RAS state after release of all resources in K



(a) Case 1 $Y_{j1} \in K \cap C_j$ (b) Case 2 $Y_{j2} \in K \cap C_j$ (c) Case 3 $Y_{j3} \in K \cap C_j$

Figure 3. Freeing A_j without unsafeness

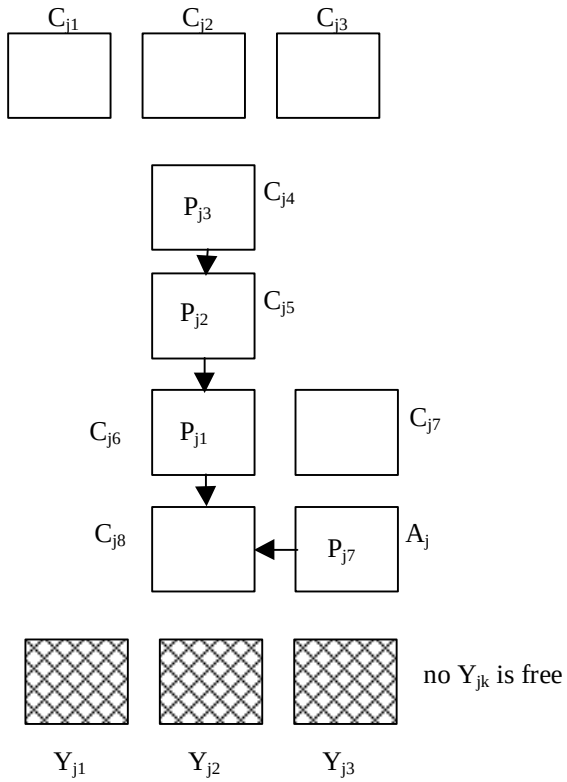


Figure 4. Case where $C_j \cap K = \varnothing$

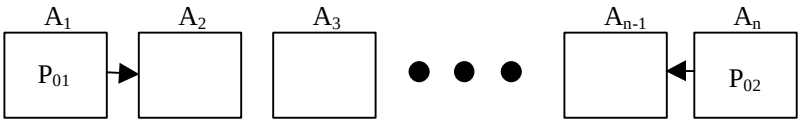
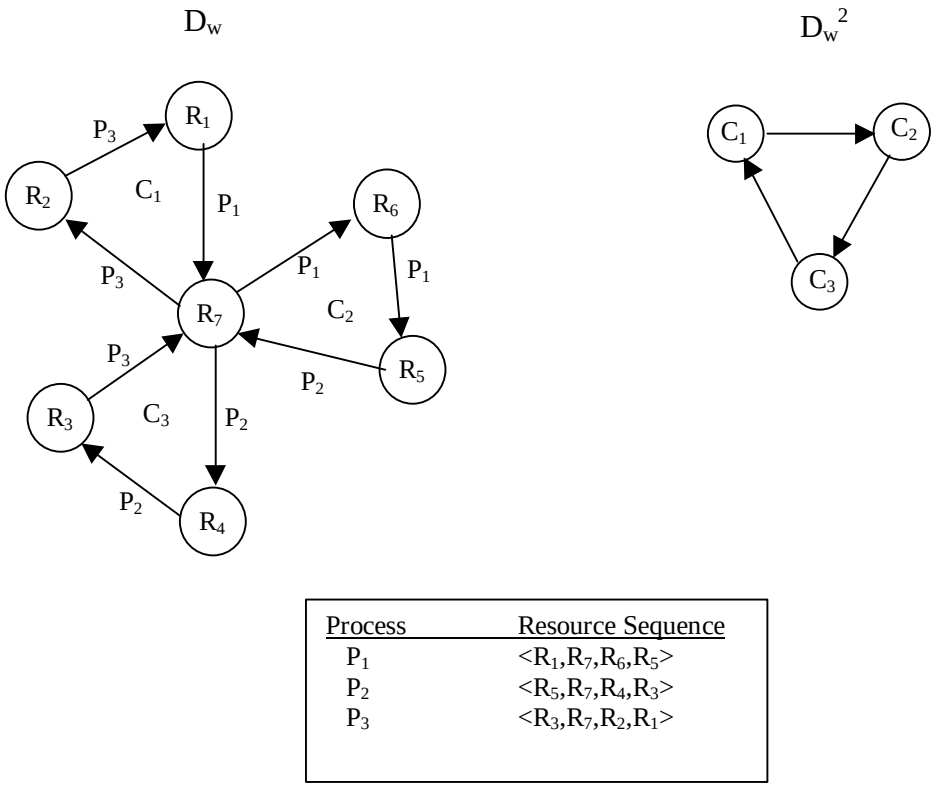
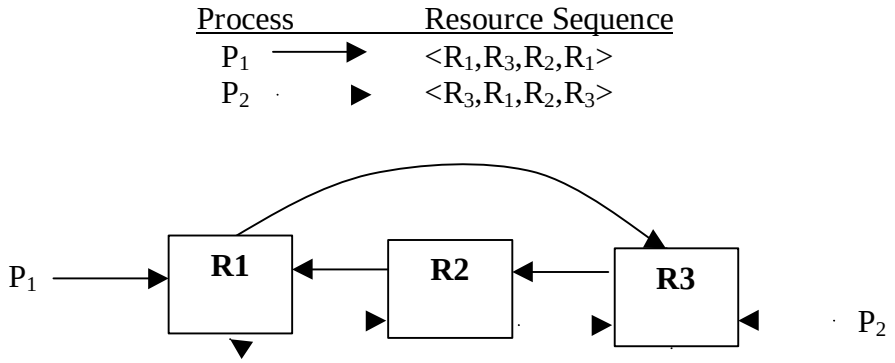


Figure 5. Unsafe state involving P_{01} and P_{02}

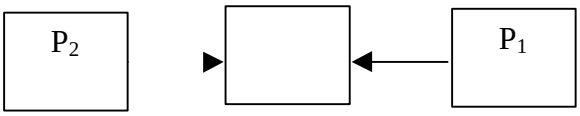


Process	Resource Sequence
P_1	$<R_1,R_7,R_6,R_5>$
P_2	$<R_5,R_7,R_4,R_3>$
P_3	$<R_3,R_7,R_2,R_1>$

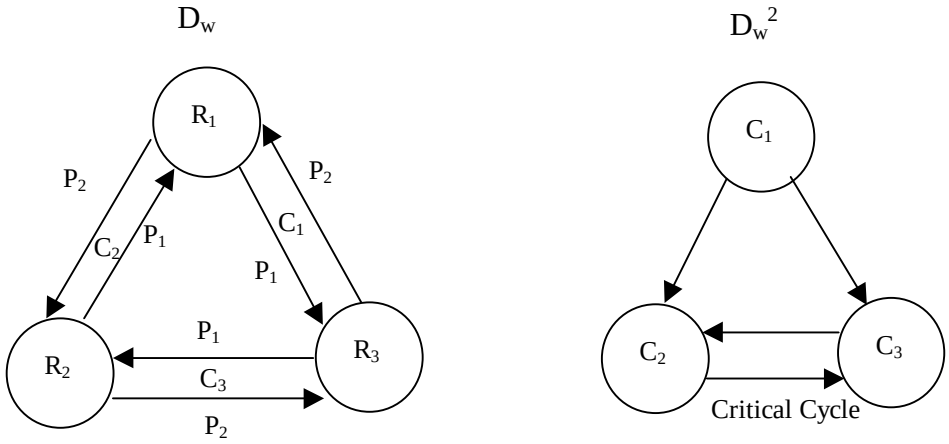
Figure 6. Fanti's first and second level digraphs



(a) SU-RAS



(b) Unreachable deadlock-free unsafe state



(c) Fanti's digraphs

Figure 7. SU-RAS with all deadlock-free unsafe states unreachable