Event-Based Feedback Control for Deadlock Avoidance in Flexible Production Systems

Maria Pia Fanti, *Member, IEEE,* Bruno Maione, *Member, IEEE,* Saverio Mascolo, *Member, IEEE,* and Biagio Turchiano, *Member, IEEE*

***Abstract*— Modern production facilities (i.e., flexible manufacturing systems) exhibit a high degree of resource sharing, a situation in which deadlocks (circular waits) can arise. Using digraph theoretic concepts we derive necessary and sufficient conditions for a deadlock occurrence and rigorously characterize highly undesirable situations (second level deadlocks), which inevitably evolve to circular waits in the next future. We assume that the system dynamics is described by a discrete event dynamical model, whose state provides the information on the current interactions job-resources. This theoretic material allows us to introduce some control laws (named restriction policies) which use the state knowledge to avoid deadlocks by inhibiting or by enabling some transitions. The restriction policies involve small on-line computation costs, so they are suitable for real-time implementation. For a meaningful class of systems one of these policies is the least restrictive deadlock-free policy one can find, namely it inhibits resource allocation only if leads directly to a deadlock. Finally, the paper discusses the computational complexity of all the proposed restriction policies and shows some examples to compare their performances.**

***Index Terms*— Deadlock avoidance, discrete event dynamical systems, FMS control.**

I. Introduction

THE RECENT researches in automated production facilities, including flexible manufacturing systems (FMS's), have led to significant improvements of productivity in manufacturing industries. The structure of the traditional manufacturing systems, designed to satisfy a demand for high volume and low variety products, can no longer cope with the rapid technological change and with the new market trend toward an ever increasing demand for variety. On the contrary FMS's, by integrating computer systems, hardware and information flows, provide manufacturers with flexibility and efficiency. A FMS consists of a set of workstations, each one capable of processing parts of different kind simultaneously. Parts (jobs, pieces) enter the system and follow different routes through the workstations, receiving service according to prescribed sequences of operations. The flexibility of FMS's relies on a programmable transportation system connecting the workstations and on a sophisticated computer control supervising process activities, transportation operations, information flow, etc.

Automated production systems (and FMS's in particular) exhibit a high degree of resource sharing. Since the parts advancing through the system compete for a finite number of resources, a deadlock (or "deadly embrace") may occur. Deadlock is a highly unfavorable situation in which the parts in a set request access to resources held by other parts in the same set. A deadlock implicating a set of parts can propagate to other parts, eventually crippling the whole system that remains blocked indefinitely. Only aborting one or more parts involved in the deadlock and granting the released resources to other implicated parts can resolve this situation.

Even if deadlock occurs in many different contexts, the problem has attracted considerable attention in computer science, since deadlocks lead to critical situations in multitasking environments. Some of these approaches use graph models to represent interactions between jobs and resources. Among the relevant works and tutorial papers we quote [2], [7], [9], [11], [14], and [15]. In particular, Coffman *et al.* [2] give four conditions for a deadlock to occur (mutual exclusion, wait-for condition, no-preemption and circular wait).

Deadlock prevention and deadlock avoidance approaches do not allow the system to enter a deadly embrace state. In deadlock prevention, state transitions follow prescribed rules ensuring that at least one of the four necessary conditions fails to hold. On the other hand, deadlock avoidance approaches examine the current system state to elude a deadlock occurrence in the next future. In the context of automated manufacturing, the interest for these problems is relatively recent. However the strategies for deadlock prevention and deadlock avoidance in manufacturing systems differ from those in computers in the fact that the system configuration and the way the system operates play a fundamental role. Moreover in the FMS's the first three conditions given by Coffman *et al.* [2] are always present. Indeed pieces use resources (buffer spaces, machines, carts, etc.) in exclusive mode, jobs hold onto resources while waiting for the next resource of their operation sequence and resources cannot be forcibly removed from the pieces utilizing them until the operation completion. Hence deadlock is excluded only if the fourth condition fails to hold.

The simplest way of preventing deadlock in automated manufacturing system is to outlaw concurrency right at design stage, by defining a layout that does not allow circular waits among jobs concurring for resources. In other words, the system layout must allow all the parts to flow in the same direction

Manuscript received January 15, 1994; revised June 26, 1995. This work was supported in part by 40% MURST funds and by the Italian National Council under Contract C.N.R. 93.00917.PF67 of "Progetto Finalizzato Robotica." This paper was recommended for publication by Associate Editor M. Silva and Editor A. Desrochers upon evaluation of the reviewers' comments.

The authors are with the Department of Electrical and Electronic Engineering, Polytechnic of Bari, Bari 70125, Italy.

Publisher Item Identifier S 1042-296X(97)01391-8.

or, in any case, jobs in process must be restricted to types following a unidirectional route. On the other hand, batching the jobs to be produced according to their flow direction and processing one batch at a time are obvious planning strategies for deadlock prevention. However, these approaches unnecessarily restrict the producible part-mix, reduce the machine utilization and undermine the flexibility of the production system, by removing the possibility of exploiting alternative routing strategies. In conclusion, the higher the requested flexibility the greater is the necessity of defining deadlock-free scheduling policies. The work of Viswanadham *et al* [18], of Banaszak and Krogh [1], Wysk *et al.* [16] and, more recently, of Hsieh and Chang [8], and Ezpeleta *et al.* [3] are remarkable contributions in this direction. In particular, using Petri net (PN) model Viswanadham *et al.* [17], [18] propose some techniques for deadlock prevention and avoidance in FMS's. The authors define a deadlock prevention policy using an exhaustive path analysis of the reachability graph of the PN. However, this approach is feasible for reasonably small systems only. The authors also introduce a deadlock avoidance policy to allocate resources in real-time and use a PN model to look ahead into the future system evolution on a finite horizon. However, this avoidance technique does not prevent the system from deadlock, but it makes infrequent this event. Therefore, if a deadlock arises, suitable recovery strategies are necessary.

Banaszak and Krogh [1] propose a PN named production Petri net (PPN) to model concurrent job flow and dynamic resource allocation in a FMS. They also introduce a deadlock avoidance technique based on the notion of restriction policy for allocating resources. This approach inhibits some transitions of the PPN according to a feedback law, which takes into account the current marking. The main quality of the method is its low complexity that makes it a good candidate for real-time control applications. However, it may impose unnecessary constraints on resource allocation, resulting in reduced performance measures.

To overcome the drawbacks of the approaches proposed by Banaszak and Krogh [1] and by Viswanadham *et al.* [18], Hsieh and Chang [8] combine the ideas of these methods and formulate a deadlock avoidance controller (DAC) based on a PN model, named controlled Petri net. They consider a class of FMS's where a manufacturing operation may require multiple resources. DAC allows transitions involving resources allocation only if they enjoy a sufficient condition to avoid deadlock (Sufficient Validity Test). However, the advantages of this method are paid by an increased complexity in comparison with the approach proposed by Banaszak and Krogh. Namely, executing the Sufficient Validity Test may require a true simulation of the job flow in the system, at each transition.

Recently, Ezpeleta *et al.* [3] formulate a policy that prevents deadlock by keeping live the PN modeling the production system. The authors use an interesting approach to derive the control policy, based on the properties of structural elements of the PN, named siphons.

On the basis of a graph-theoretic approach, Wysk *et al.* [16] derive a model for deadlock detection and avoidance in manufacturing systems. They present an algorithm for deadlock detection, to be used in conjunction with deadlock resolution approaches. Using the same model, Kumaran *et al.* [10] propose an approach to deadlock avoidance. The method is a complete look ahead procedure, performed before the movement of any part between machines and before the arrival of a new part into the system.

In this paper we define deadlock avoidance policies, suitable for real-time implementation. Such policies are simple and require no look ahead or a look ahead of just one step. For tracking the job flow through the system, we refer to a discrete event dynamical system (DEDS) model [4], [5], [19] even if we do not treat it in detail. We only need to assume that, at any instant, the system state describes the jobs in process, the resources that parts currently hold and will acquire before the processing completion and, finally, the sequence of all the resources necessary to process each job. As time moves on, the state changes at any event occurrence. As events relevant for the deadlock analysis, we consider those corresponding to new jobs entering the system and to jobs progressing from one resource to another one or leaving the system. On this basis we define feedback and on-line control rules, which use the knowledge of the current system's state to inhibit (or enable) the occurrence of the above mentioned events. In particular, such rules (restriction policies) allow us to define classes of deadlock-free scheduling policies involving on-line algorithms with low computational complexity. In dealing with this problem we refer to the digraph theory to introduce two digraphs, the former describing the whole sequence of resources that each job needs to orderly acquire and the latter representing the current system's state and future resource requirements. In particular, the Working Procedure Digraph defines once for all the potential interactions among the whole production mix and the sequences of resources required by each part in the mix. On the other hand, for any system's state the Transition Digraph describes the current interactions by indicating both the resources currently held by jobs in process and the resources requested by the same parts in the next future. Using such digraphs allows us to derive necessary and sufficient conditions for deadlock occurrence and for the identification of critical situations (named Second Level Deadlocks or SLD, for brevity), which are not circular waits, even if they necessarily evolve to deadlocks in the next future. Such conditions provide the theoretic framework allowing the formulation of restriction policies for deadlock avoidance.

The paper is organized as follows. Section II introduces basic notations and preliminary ideas by defining the Working Procedure Digraph and the Transition Digraph. Section III gives necessary and sufficient conditions for deadlock occurrence, by relating deadlocks to the presence of cycles in the Transition Digraph. Section IV discusses the problem of SLD by developing theoretic principles to identify and characterize such situations. Moreover it enlightens the mechanism of interaction among cycles of the Working Procedure Digraph, which provides the key for understanding the complex phenomenon of a SLD occurrence. Section V introduces the deadlock-free scheduling policies and Section VI discusses their complexity concerning both off-line and on-line computations. Section VII provides some examples and Section VIII gives the conclusions.

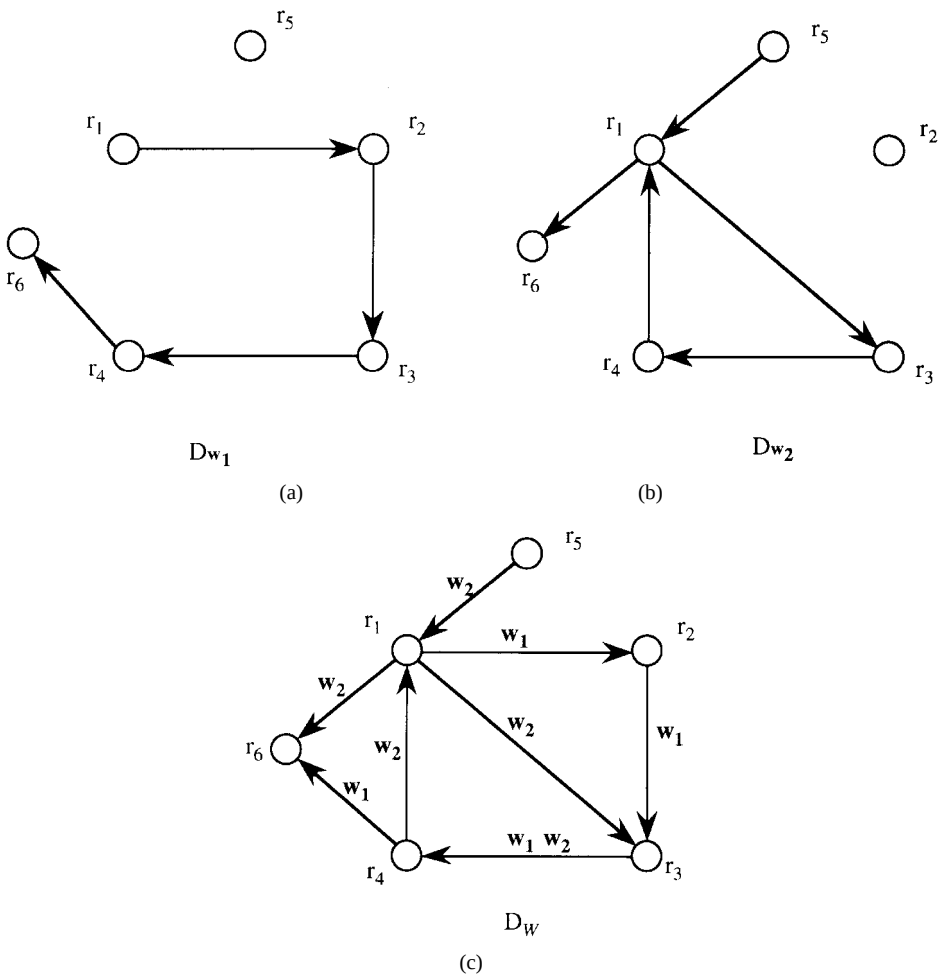

Fig 1. Digraphs $D_{\boldsymbol{w}\mathbf{1}}$, $D_{\boldsymbol{w}\mathbf{2}}$ and D_W for Example 1.

II. Basic Definitions and Notations

Let us consider a production system, consisting of R -1 resources $r_1, r_2, \cdots, r_{\mathrm{R}-1}$, to which the jobs access, as they flow through the system. For our purposes, we use the term "resources" only to refer to the direct means of production, i.e., to the facilities (machines, buffers, carts of transportation systems) which provide service for parts. Each part enters the system and advances through it, holding one resource at a time in exclusive mode. In other words, different jobs can not hold the same resource simultaneously. Flowing through the system, each part follows a specific working procedure, i.e., a strict order of succession in using the resources necessary to provide the scheduled services. So, if J indicates the set of jobs we have to produce, each part $j \in J$ follows a particular working procedure $\boldsymbol{w}$ chosen in the set W of the admissible operation sequences. At the handling and processing completion, the part leaves the system. By convention, however, we consider it as a job acceding to a fictitious resource r_{R}. Consequently, the set of all the available resources is $R = \{r_i \ i = 1, 2, \cdots, \mathrm{R}\}$. For example, $\boldsymbol{w} = (r_3, r_1, r_4, r_3, r_{\mathrm{R}})$ means that the job requests the resources r_3, r_1 and r_4 in strict ordering; then it holds r_3 again and, finally, it leaves the system.

At this point, to describe all the admissible working procedures, it is convenient to introduce a digraph representation of the specific order in which resources appear in all the operation sequences. We assume that the reader is familiar with the basic elements of digraph theory. So let $D_{\boldsymbol{w}} = (N, E_{\boldsymbol{w}})$ be the digraph associated with $\boldsymbol{w} \in W$, where N and $E_{\boldsymbol{w}}$ represent the node set and the edge set, respectively. Each vertex in N corresponds to a resource r_i, so that we use the same symbols for vertices and resources, i.e., $N = R$**.** An edge e_{im} is an ordered pair (r_i, r_m) of nodes in N such that $e_{im} = (r_i, r_m)$, directed from r_i to r_m, belongs to $E_{\boldsymbol{w}} \subset N \times N$ iff (if and only if) r_m immediately follows r_i in $\boldsymbol{w}$. With these notions as background, we define the Working Procedure Digraph as $D_W = (N, E_W)$, with

$$E_W = \bigcup_W E_{\boldsymbol{w}}. \tag{1}$$

Example 1: Fig. 1 associates the digraphs $D_{\boldsymbol{w}\mathbf{1}}$, $D_{\boldsymbol{w}\mathbf{2}}$ and D_W with $\boldsymbol{w_1} = (r_1, r_2, r_3, r_4, r_6)$, $\boldsymbol{w_2} = (r_5, r_1, r_3, r_4, r_1, r_6)$, and with $W = \{\boldsymbol{w_1}, \boldsymbol{w_2}\}$, respectively. Each edge of D_W is marked by the working procedures involving it.

Remark 1: In opposition to the digraph introduced by Wysk *et al.* [16], D_W is drawn once for all and does not change as the jobs advance through the system. Morever D_W does not admit parallel edges.

Clearly the digraph D_W depicts a static situation and is not able to describe the progress of the jobs in process. Namely,

parts flowing through the system obey a dynamic mechanism in which they continuously acquire and release resources. Hence a dynamic and detailed model is necessary for tracking a job's complete experience as it enters the system and makes transition from a resource to the next one and for determining the time occurrence of each transition. Recently, DEDS models have been used to describe concurrency and contention for resources by multiple jobs in the system. Such models need an explicit definition of scheduling policies, ruling the sequencing (determining the sequence of part loading into the system), the dispatch timing (fixing the instants at which the pieces have to enter the system), the job routing (choosing the path each part has to follow, to be processed), the service priority setting (defining priorities for jobs concurring on the same resource). Here we do not treat DEDS models in detail but we refer to [4], [5], and [19]. Indeed, we only assume that a DEDS model S describes the production system and we denote by $\boldsymbol{q}(t) \in Q$ the S-state at time t, where Q is the complete state set. In particular, at each time $t, \boldsymbol{q}$ must describe

a) the set $J_{\boldsymbol{q}}$ of the jobs in process, the resources currently held by each job $j \in J_{\boldsymbol{q}}$, the working procedure associated with such a job and, finally, the residual working procedure, i.e., the sequence of resources necessary for each $j \in J_{\boldsymbol{q}}$ to complete the processing.

In the sequel, we use the symbol $\mathrm{WP}(j) = \boldsymbol{w}$ to indicate that the job $j \in J_{\boldsymbol{q}}$ is processed according to $\boldsymbol{w}$. Analogously, $\mathrm{RW}(j)$ designates the residual working procedure corresponding to j and $\mathrm{HR}(j)$ and $\mathrm{SR}(j)$ stand for the first (holding the job) and the second resource of $\mathrm{RW}(j)$, respectively. Note that if $j \in J_{\boldsymbol{q}}, \mathrm{RW}(j)$ contains two resources at least. Namely, by assumption, a job leaving the system accedes to the fictitious resource r_{R}, but, once it acquires r_{R}, it is removed from the set $J_{\boldsymbol{q}}$. Obviously, since WP, RW, HR, and SR are defined on $J_{\boldsymbol{q}}$, they depend on the current state, even if we drop the argument $\boldsymbol{q}$ for simplicity.

As the time moves on, a state change takes place at any event occurrence. According to the above characterization of $\boldsymbol{q}$, we assume that

b) an event of the DEDS model occurs whenever a job acquires and/or releases a resource.

Of course a DEDS model may also consider more event types than assumption b) does. However point b) indicates the only events relevant to our discussion. Finally, we assume that

1) the occurrences of system events are strictly ordered in time.

The information given by the state and the state transition mechanism allows us to approach the concurrent flow of multiple jobs in the system, which all compete for a finite set of resources. In considering this problem, a digraph representation is useful to describe the dynamic interaction job-resources. Thus we define a subgraph of D_W, named Transition Digraph $D_{\mathrm{Tr}}(\boldsymbol{q}) = [N, E_{\mathrm{Tr}}(\boldsymbol{q})]$, as follows: edge $e_{im} \in E_{\mathrm{Tr}}(\boldsymbol{q})$ iff a job $j \in J_{\boldsymbol{q}}$ holds r_i in the state $\boldsymbol{q}$ and requires r_m as next resource: i.e., $\mathrm{HR}(j) = r_i$ and $\mathrm{SR}(j) = r_m$. Hence, $D_{\mathrm{Tr}}(\boldsymbol{q})$ describes all the next transitions of jobs in S. Remarkable properties of this digraph are the following. First, in a generic state $\boldsymbol{q}$, any vertex r_i $(i = 1, 2, \cdots, \mathrm{R} - 1)$ has outdegree [6] either equal to zero or equal to one: in the first case the resource r_i is *idle* (in other words, no job holds r_i), whereas in the second one r_i is *busy* (i.e., a job holds r_i). In particular, the vertex r_{R} has zero outdegree in any state $\boldsymbol{q}$, since the fictitious resource is ever at disposal. The indegree of a vertex r_m $(m = 1, 2, \cdots, \mathrm{R})$ is a nonnegative integer giving the number of jobs currently requiring r_m as next resource. Isolated vertices, if any, correspond to *idle* resources that are not requested next.

Not always a transition represented by an edge $e_{im} \in E_{\mathrm{Tr}}(\boldsymbol{q})$ (for brevity: transition e_{im}) can be immediately executed. Namely, for the transition e_{im} to occur, r_m must be *idle* in the state $\boldsymbol{q}$. In this case, we call e_{im} a *feasible* transition for the state $\boldsymbol{q}$. Vice versa, if r_m is *busy*, we say that e_{im} is a *blocked* transition for $\boldsymbol{q}$. In other words, any edge of $E_{\mathrm{Tr}}(\boldsymbol{q})$ represents either a *feasible* transition, if it is incident to an *idle* vertex, or a *blocked* transition, if it ends in a *busy* node. In the sequel we denote as $E_F(\boldsymbol{q})$ the set of the *feasible* transitions in $E_{\mathrm{Tr}}(\boldsymbol{q})$. Moreover, dealing with *busy/idle* resources or with *feasible/blocked* transitions, we implicitly refer to the Transistion Digraph $D_{\mathrm{Tr}}(\boldsymbol{q})$, unless we specify the contrary.

Example 2: Considering the working procedures of Example 1, let S be in the state $\boldsymbol{q}$, with: $J_{\boldsymbol{q}} = \{j_1, j_2, j_3, j_4\}$; $\mathrm{WP}(j_1) = \boldsymbol{w_1}$; $\mathrm{RW}(j_1) = (r_3, r_4, r_6)$; $\mathrm{WP}(j_2) = \boldsymbol{w_1}$, $\mathrm{RW}(j_2) = (r_2, r_3, r_4, r_6)$; $\mathrm{WP}(j_3) = \boldsymbol{w_2}$, $\mathrm{RW}(j_3) = (r_4, r_1, r_6)$; $\mathrm{WP}(j_4) = \mathrm{RW}(j_4) = \boldsymbol{w_2}$. The solid lines in Fig. 2(a) form the Transistion Digraph $D_{\mathrm{Tr}}(\boldsymbol{q})$. The blank circles depict the idle vertices r_1 and r_6, while edges pointing to r_1 represent feasible transitions. For convenience, Fig. 2(a) also depicts the second transitions (dashed lines) if they follow *feasible* transitions in the residual working procedures.

Before concluding this section, we show how to use the digraph $D_{\mathrm{Tr}}(\boldsymbol{q})$ to provide a versatile representation of the transition mechanism. Suppose S be in the state $\boldsymbol{q}$ at time t and consider a working procedure $\boldsymbol{w} \in W$. At such an instant for a job $j \in J$ to enter S and to receive service according to $\boldsymbol{w}$, the first resource in $\boldsymbol{w}$ must be necessarily *idle*. On the occurrence of this event, S makes transition from $\boldsymbol{q}$ to a new state $\boldsymbol{q'}$. Let $D_{\mathrm{Tr}}(\boldsymbol{q}, \boldsymbol{w}) = [N, E_{\mathrm{Tr}}(\boldsymbol{q}, \boldsymbol{w})]$ indicate the Transition Digraph $D_{\mathrm{Tr}}(\boldsymbol{q'})$. If r_m and r_p are, respectively, the first and the second resource in $\boldsymbol{w}$, then $E_{\mathrm{Tr}}(\boldsymbol{q}) \cup \{e_{mp}\}$ yields the edge set $E_{\mathrm{Tr}}(\boldsymbol{q}, \boldsymbol{w})$. Clearly r_m is *idle* in $D_{\mathrm{Tr}}(\boldsymbol{q})$ but *busy* in $D_{\mathrm{Tr}}(\boldsymbol{q}, \boldsymbol{w})$.

Analogously, let a job $j \in J_{\boldsymbol{q}}$ hold r_i at a given instant t and request access to r_m, i.e., $\mathrm{HR}(j) = r_i$ and $\mathrm{SR}(j) = r_m$. Moreover let r_m be *idle* at time t, i.e., $e_{im} \in E_F(\boldsymbol{q})$. The transition e_{im}, occurring when j leaves r_i to hold r_m, updates the state from $\boldsymbol{q}$ to $\boldsymbol{q'}$. In particular, if $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{im}) = [N, E_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})]$ denotes the Transition Digraph associated with $\boldsymbol{q'}$, the edge set $E_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$ results from the following operation executed on $E_{\mathrm{Tr}}(\boldsymbol{q})$:

1) removing the edge e_{im} from $E_{\mathrm{Tr}}(\boldsymbol{q})$;
2) adding the edge e_{mp} to $E_{\mathrm{Tr}}(\boldsymbol{q})$, provided that $\mathrm{RW}(j)$ contains three resources at least and r_p is the third resource in $\mathrm{RW}(j)$.

In conclusion, r_i is *idle* and r_m is *busy* (if $m \neq \mathrm{R}$) in $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$. Of course all the remaining nodes keep unchanged the *busy/idle* condition they had in $D_{\mathrm{Tr}}(\boldsymbol{q})$.

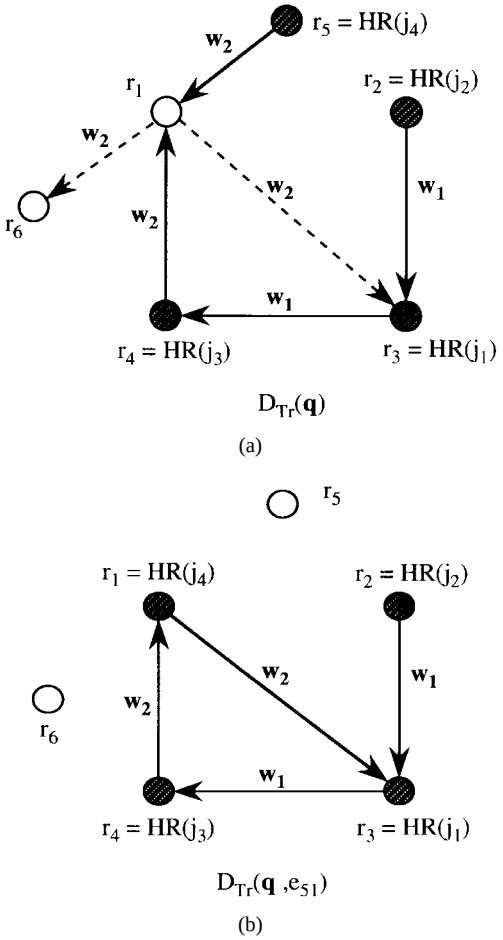

Fig. 2. (a) Transistion Digraph for Example 2. (b) Transistion Digraph $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{51})$.

With reference to Example 2, Fig. 2(b) shows the Transition Digraph $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{51})$ resulting from $D_{\mathrm{Tr}}(\boldsymbol{q})$ of Fig. 2(a).

III. Necessary and Sufficient Conditions for Deadlock

We begin this section by introducing a formal definition of deadlock state for S.

Definition 1: Let $\boldsymbol{q} \in Q$. If there exist two nonempty subsets $J_D \subset J_{\boldsymbol{q}}$ and $R_D \subset R$ such that:

$$\mathrm{HR}(J_D) = R_D \tag{2}$$

$$\mathrm{SR}(J_D) \subset R_D \tag{3}$$

then $\boldsymbol{q}$ is a deadlock state for S.

By the above definition, each job in J_D remains indefinitely blocked because it is waiting for a *busy* resource held by another job in J_D.

Example 3: Let the system of Example 1 be in state $\boldsymbol{q}$, with: $J_{\boldsymbol{q}} = \{j_1, j_2, j_3, j_4\}$; $\mathrm{WP}(j_1) = \boldsymbol{w_1} = (r_1, r_2, r_3, r_4, r_6)$; $\mathrm{RW}(j_1) = (r_3, r_4, r_6)$; $\mathrm{WP}(j_2) = \boldsymbol{w_1}, \mathrm{RW}(j_2) = (r_2, r_3, r_4, r_6)$; $\mathrm{WP}(j_3) = \boldsymbol{w_2} = (r_5, r_1, r_3, r_4, r_1, r_6), \mathrm{RW}(j_3) = (r_4, r_1, r_6)$; $\mathrm{WP}(j_4) = \boldsymbol{w_2}$, $\mathrm{RW}(j_4) = (r_1, r_3, r_4, r_1, r_6)$. Putting $J_D = J_{\boldsymbol{q}}$ and $R_D = \{r_1, r_2, r_3, r_4\}$, we get $\mathrm{HR}(J_D) = R_D$, $\mathrm{SR}(J_D) = \{r_1, r_3, r_4\} \subset R_D$. Hence, by Definition 1, $\boldsymbol{q}$ is a deadlock state.

The next result confirms that the deadlock is related to the presence of cycles [6] in $D_{\mathrm{Tr}}(\boldsymbol{q})$. It is quite intuitive and reminds us of an analogous result due to Wysk *et al.* [16]. However, using $D_{\mathrm{Tr}}(\boldsymbol{q})$ makes unnecessary the circuit validation requested by Wysk *et al.*

Theorem 1: Necessary and sufficient condition for $\boldsymbol{q}$ to be a deadlock state for S, is that there is at least one cycle in $D_{\mathrm{Tr}}(\boldsymbol{q})$.

Proof: Sufficiency.

Let γ be a cycle of $D_{\mathrm{Tr}}(\boldsymbol{q})$. Clearly each vertex of γ represents a *busy* resource. Then, if R_D is the set of such resources and J_D is the set of jobs holding them, we get $\mathrm{HR}(J_D) = R_D$. Moreover, we infer $\mathrm{SR}(J_D) \subset R_D$ because edges outgoing from vertices in R_D are incident to vertices in the same set. By Definition 1, this proves the sufficiency of the Theorem.

Necessity.

Suppose there exist two nonempty subsets $J_D \subset J_{\boldsymbol{q}}$ and $R_D \subset R$ satisfying (2) and (3). We are therefore led to conclude that each edge of $E_{\mathrm{Tr}}(\boldsymbol{q})$ starting from a vertex in R_D is directed to a vertex still belonging to R_D. In other words, no vertex from $R - R_D$ is reachable from any vertex in R_D. Furthermore, by (2) all the resources in R_D are *busy*, so that the corresponding nodes have outdegree equal to one in $D_{\mathrm{Tr}}(\boldsymbol{q})$. Hence any vertex in R_D is the first node of a walk [6] consisting of an infinite sequence of edges, that join nodes from R_D. Since the cardinality of R_D is finite, this walk must contain a cycle.

According to the above theorem, if γ is a cycle of $D_{\mathrm{Tr}}(\boldsymbol{q})$ we say that γ is in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q})$ and that $D_{\mathrm{Tr}}(\boldsymbol{q})$ shows a deadlock. The above consideration allows us to restate Theorem 1 as follows. Let D_W contain G cycles and let $\gamma_n = (N_n, E_n)$ $n \in \{1, 2, \cdots, G\}$ be one of them, where $N_n \subset N$ and $E_n \subset E_W$. With this notation, γ_n is in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q})$ iff:

$$E_n \cap E_{\mathrm{Tr}}(\boldsymbol{q}) = E_n. \tag{4}$$

Now, we give two definitions that allow us to deduce additional elements of evidence from Theorem 1.

Definition 2: Let γ_n be a cycle of D_W. We define *Cycle Capacity* of γ_n the integer $C(\gamma_n) = \mathrm{Card}(N_n) = \mathrm{Card}(E_n)$, where $\mathrm{Card}(\cdot)$ stands for "cardinality of $\cdots$."

Definition 3: Let γ_n be a cycle of D_W. The Overlap Degree of γ_n in state $\boldsymbol{q}$ is the integer $O_{\boldsymbol{q}}(\gamma_n) = \mathrm{Card}[E_n \cap E_{\mathrm{Tr}}(\boldsymbol{q})]$.

Because of Theorem 1 and Definition 3 we can immediately state the following corollary.

Corollary 1: Necessary and sufficient condition for the cycle γ_n of D_W to be in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q})$, is that $O_{\boldsymbol{q}}(\gamma_n) = C(\gamma_n)$.

Now, let

$$C_0 = \min_{D_W} C(\gamma_n) \tag{5}$$

where the minimum refers to all the cycles of D_W and where we put $C_0 = \infty$ if D_W is acyclic. The following corollary directly follows from Theorem 1 and Corollary 1.

Corollary 2: Necessary condition for $\boldsymbol{q}$ to be a deadlock state is that D_W contains at least one cycle and that $J_{\boldsymbol{q}}$ satisfies the following relation

$$\mathrm{Card}(J_{\boldsymbol{q}}) \geq C_0. \tag{6}$$

IV. Second Level Deadlock

Deadlocks are highly undesirable. However, it is also possible that some critical situations occur that are not deadlocks, even if they necessarily evolve to circular waits in the immediate future. In this section we develop theoretic principles to identify and characterize such circumstances, which are called SLD. Subsequently, the conditions for a SLD occurrence allow us to introduce deadlock-free control rules. To begin with, we give the following definition.

Definition 4: Let $\boldsymbol{q}$ be not a deadlock state for S. We say that $\boldsymbol{q}$ is a SLD state for S, if there exist two nonempty subsets $J_S \subset J_{\boldsymbol{q}}$ and $R_S \subset R$, with $\mathrm{Card}(J_S) < \mathrm{Card}(R_S)$ satisfying the following properties:

- D4a) $\mathrm{HR}(J_S) \subset R_S, \mathrm{SR}(J_S) \subset R_S$;
- D4b) all the resources belonging to $[R_S - \mathrm{HR}(J_S)]$ are *idle*;
- D4c) for each job $j \in J_S$ such that $\mathrm{SR}(j) \in [R_S - \mathrm{HR}(J_S)]$, the transition releasing $\mathrm{HR}(j)$ to hold $\mathrm{SR}(j)$ leads to a deadlock involving jobs of J_S only, i.e., $J_D \subset J_S$.

For simplicity, in the sequel we denote by E_S the set of *feasible* transitions in $D_{\mathrm{Tr}}(\boldsymbol{q})$ defined by D4c), i.e., $E_S = \{e_{im} \in E_F(\boldsymbol{q})$: $r_i \in \mathrm{HR}(J_S)$, $r_m \in [R_S - \mathrm{HR}(J_S)]\}$.

Definition 4 identifies a critical situation in which each job of J_S either is blocked, because it requests access to a resource held by another part in J_S, or leads to a deadlock on acquiring the next resource in its Residual Working Procedure. The following example clarifies the idea of SLD.

Example 4: Consider a system S with $\mathrm{R} = 9, W = \{\boldsymbol{w_1}, \boldsymbol{w_2}, \boldsymbol{w_3}\}$, $\boldsymbol{w_1} = (r_1, r_7, r_2, r_8, r_3, r_4, r_8, r_9)$, $\boldsymbol{w_2} = (r_5, r_8, r_6, r_7, r_9)$, $\boldsymbol{w_3} = (r_3, r_4, r_8, r_1, r_7, r_9)$. Fig. 3 shows D_W. Moreover, let the current state $\boldsymbol{q}$ be characterized by: $J_{\boldsymbol{q}} = \{j_i\ i = 1, 2, \cdots, 7\}, \mathrm{WP}(j_1) = \mathrm{WP}(j_2) = \mathrm{WP}(j_3) = \boldsymbol{w_1}, \mathrm{WP}(j_4) = \mathrm{WP}(j_5) = \boldsymbol{w_2}$, $\mathrm{WP}(j_6) = \mathrm{WP}(j_7) = \boldsymbol{w_3}; \mathrm{RW}(j_1) = \boldsymbol{w_1}, \mathrm{RW}(j_2) = (r_7, r_2, r_8, r_3, r_4, r_8, r_9), \mathrm{RW}(j_3) = (r_2, r_8, r_3, r_4, r_8, r_9)$, $\mathrm{RW}(j_4) = (r_6, r_7, r_9)$, $\mathrm{RW}(j_5) = \boldsymbol{w_2}$, $\mathrm{RW}(j_6) = \boldsymbol{w_3}$, $\mathrm{RW}(j_7) = (r_4, r_8, r_1, r_7, r_9)$. In view of Definition 4, $\boldsymbol{q}$ is a SLD state with $J_S = J_{\boldsymbol{q}}$, $R_S = \{r_i\ i = 1, 2, \cdots, 8\}$ where $\mathrm{Card}(J_S) < \mathrm{Card}(R_S)$, $\mathrm{HR}(J_S) = \{r_i\ i = 1, \cdots, 7\}$, $\mathrm{SR}(J_S) = \{r_2, r_4, r_7, r_8\}$, $[R_S - \mathrm{HR}(J_S)] = \{r_8\}$ and r_8 is *idle*. Fig. 4 shows digraph $D_{\mathrm{Tr}}(\boldsymbol{q})$: the *idle* resources r_8 and r_9 are indicated by blank circles. The nonblocked jobs in J_S are j_3, j_5, and j_7, so that the first transitions in their residual working procedures form the set $E_S = \{e_{28}, e_{58}, e_{48}\}$. If j_7 executes transition e_{48}, then it establishes e_{81} (dashed line in Fig. 4) as its next transition and this edge deadlocks the cycle $\gamma_1 = (\{r_1, r_7, r_2, r_8\}, \{e_{17}, e_{72}, e_{28}, e_{81}\})$ involving jobs in the set $\{j_1, j_2, j_3, j_7\}$. Analogously, if j_3 makes transition e_{28}, then e_{83} becomes its next transition and deadlocks the job set $\{j_3, j_6, j_7\}$ in the cycle $\gamma_2 = (\{r_8, r_3, r_4\}, \{e_{83}, e_{34}, e_{48}\})$. Finally, if j_5 executes e_{58}, it establishes e_{86} as its next transition. Hence cycle $\gamma_3 = (\{r_8, r_6, r_7, r_2\}, \{e_{86}, e_{67}, e_{72}, e_{28}\})$ shows a deadlock involving the job set $\{j_5, j_4, j_2, j_3\}$.

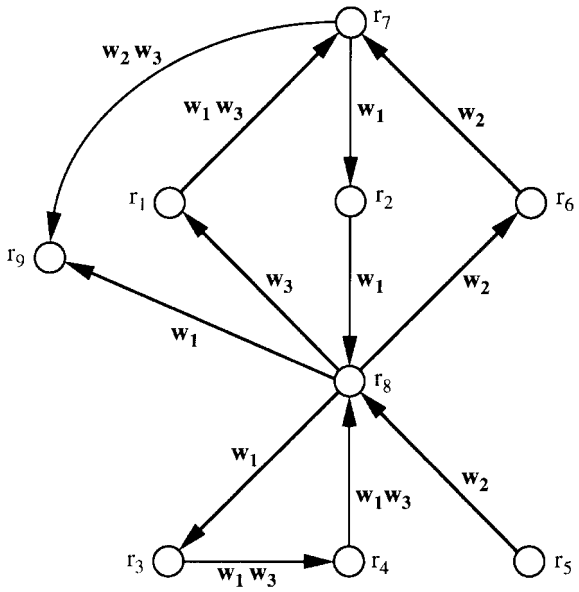

Fig. 3. Working Procedure Digraph for Example 4..

Next we introduce an original result, which establishes a necessary and sufficient condition for an actual SLD. It is based on both the Working Procedure Digraph and the Transition Digraph and it is consistent with other partial results obtained in a rather intuitive form [10], [16].

Theorem 2: Let $\boldsymbol{q}$ be not a deadlock state for S. Necessary and sufficient condition for $\boldsymbol{q}$ to be a SLD state is that there exists an ordered cycle set $\Gamma_o = \{\gamma_1, \cdots, \gamma_n, \cdots, \gamma_H\}$ in D_W, such that, putting

$$N_{\Gamma_o} = \bigcup_{\Gamma_o} N_n \tag{7a}$$

$$E_{\Gamma_o} = \bigcup_{\Gamma_o} E_n \tag{7b}$$

the following conditions hold true:

- Th2a) there exists only one *idle* resource in N_{Γ_o}, say r_m, and it is such that $N_n \cap N_h - \{r_m\}$ for any $\gamma_n, \gamma_h \in \Gamma_o$, with $\gamma_n \neq \gamma_h$. Moreover, for each $\gamma_n \in \Gamma_o$, there is only one *feasible* transition in E_n, say $e_F(\gamma_n)$.
- Th2b) Γ_o enjoys the following cyclic relation: γ_{n+1} is in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q}, e_F(\gamma_n))$ for $n = 1, 2, \cdots, H-1$ and γ_1 is in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q}, e_F(\gamma_H))$.

The Proof is given in the Appendix

The cyclic relation established by Theorem 2 exhibits formally the condition of "interaction among circuits" underlying a SLD. This condition is clearly depicted by the following example.

Example 5: Consider the system S of Example 4 and the corresponding SLD state depicted by Fig. 4. The cycle set $\{\gamma_1, \gamma_2\}$ of D_W (see Fig. 3 and Example 4) satisfies Th2a). Namely, $\{r_8\} = N_1 \cap N_2$ is the only *idle* resource. In addition, $e_{28} = e_F(\gamma_1)$ and $e_{48} = e_F(\gamma_2)$ are the only

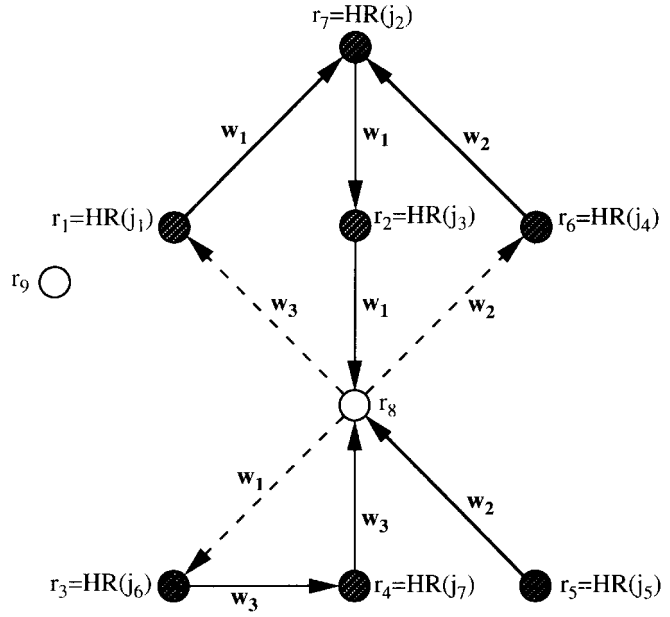

Fig. 4. Transition Digraph for Example 5.

feasible transitions belonging to γ_1 and γ_2, respectively. In $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{28}), e_{28}$ deadlocks γ_2; in $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{48}), e_{48}$ deadlocks γ_1.

It is also worthwhile to note that γ_3 does not belong to Γ_o. Namely the intersection $N_1 \cap N_3 = \{r_7, r_2, r_8\}$ violates Th2a). Moreover γ_3 is deadlocked by e_{58}, which belongs neither to γ_1 nor to γ_2.

At this point we use D_W to determine all the potential cyclic relations among its circuits. To this aim, each cycle of D_W is collapsed into a vertex of a new digraph, called SLD, defined as follows.

Definition 5: Let $\{\gamma_1, \gamma_2, \cdots, \gamma_G\}$ be the complete set of all the cycles of D_W. The Second Level Digraph $D_W^2 = (N^2, E_W^2)$ associates a vertex n_h^2 with each cycle γ_h of D_W, so that $N^2 = \{n_1^2, \cdots, n_h^2, \cdots n_G^2\}$. Moreover, the edge $e_{hs}^2 = (n_h^2, n_s^2)$ belongs to E_W^2, iff:

D5a) γ_h and γ_s have only one vertex in common (say r_m) and;

D5b) there exists a working procedure $\boldsymbol{w} \in W$, containing vertices r_i, r_m and r_p in strict order of succession, with $e_{im} \in E_h$ and $e_{mp} \in E_s$.

Now we introduce the following definition.

Definition 6: Let $\gamma_n^2 = (N_n^2, E_n^2)$ be a cycle of D_W^2 (second level cycle) and let $\Gamma_n = \{\gamma_1, \cdots \gamma_h, \cdots, \gamma_{P_n}\}$, with $\gamma_h = (N_h, E_h)$ for $h = 1, 2, \cdots, P_n$, be the set of cycles in D_W corresponding to the vertices in N_n^2. The Capacity, $C(\gamma_n^2)$, of the second level cycle γ_n^2 equals the number of resources associated with the cycles in Γ_n:

$$C(\gamma_n^2) = \mathrm{Card}\left(\bigcup_{h=1}^{P_n} N_h\right). \tag{8}$$

Moreover we indicate as $O_{\boldsymbol{q}}(\gamma_n^2)$ the Overlap Degree of γ_n^2 in the state $\boldsymbol{q}$, where

$$O_{\boldsymbol{q}}(\gamma_n^2) = \sum_{h=1}^{P_n} O_{\boldsymbol{q}}(\gamma_h). \tag{9}$$

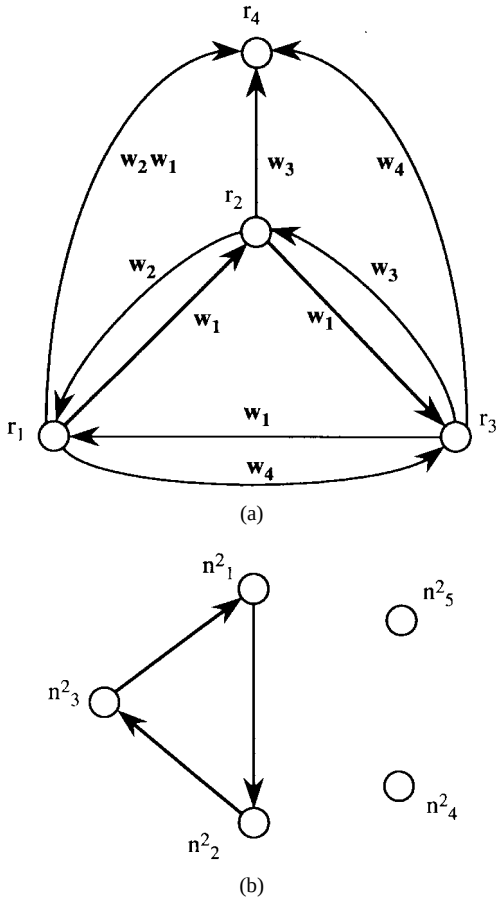

Fig. 5. (a) Transition Digraph for Example 6. (b) Second Level Digraph for Example 6.

By comparing the properties of the set Γ_o stated by Theorem 2 and the definition of Second Level Digraph we conclude that all the potential situations of SLD are related to second level cycles. However, not all the second level cycles indicate potential SLD. The next example clarifies this point.

Example 6: Let a system S consist of R = 4 resources with $W = \{\boldsymbol{w_1}, \boldsymbol{w_2}, \boldsymbol{w_3}, \boldsymbol{w_4}\}$ where $\boldsymbol{w_1} = (r_1, r_2, r_3, r_1, r_4), \boldsymbol{w_2} = (r_2, r_1, r_4), \boldsymbol{w_3} = (r_3, r_2, r_4)$ and $\boldsymbol{w_4} = (r_1, r_3, r_4)$. Fig. 5(a) exhibits the corresponding D_W which contains five cycles $\gamma_1 = (\{r_1, r_2\}, \{e_{12}, e_{21}\})$, $\gamma_2 = (\{r_2, r_3\}, \{e_{23}, e_{32}\}), \gamma_3 = (\{r_3, r_1\}, \{e_{31}, e_{13}\}), \gamma_4 = (\{r_1, r_2, r_3\}, \{e_{12}, e_{23}, e_{31}\}), \gamma_5 = (\{r_1, r_3, r_2\}, \{e_{13}, e_{32}, e_{21}\})$. Fig. 5(b) shows that the Second Level Digraph has only one cycle: $\gamma_1^2 = (\{n_1^2, n_2^2, n_3^2\}, \{e_{12}^2, e_{23}^2, e_{31}^2\})$. However γ_1^2 does not correspond to a potential SLD. This is because the first level cycles leading to γ_1^2 (i.e., γ_1, γ_2, and γ_3) do not share a unique vertex as Th2a) requests.

The only second level cycles representing potential conditions of SLD belong to a subset Γ^2 defined as follows:

$$\Gamma^2 = \{\gamma_n^2 \text{ of } D_W^2 \colon \mathrm{Card}\left(\bigcap_{h=1}^{P_n} N_h\right) = 1 \text{ and } \mathrm{Card}(N_h \cap N_k) = 1 \text{ for any } h, k \in \{1, 2, \cdots, P_n\} \text{ with } h \neq k\}. \tag{10}$$

An element from Γ^2 corresponds to a set of cycles of D_W that are disjoint but for a vertex shared by all of them.

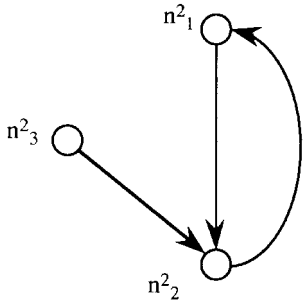

Fig. 6. Second Level Digraph for Example 7.

Obviously, in Example 6 Γ^2 is empty so that no SLD can occur. On the contrary, Γ^2 is not empty in the following example.

Example 7: Fig. 6 shows the Second Level Digraph D_W^2, for the system of Examples 4 and 5, whose Working Procedure Digraph contains three cycles (see Fig. 3): $\gamma_1 = (\{r_1, r_7, r_2, r_8\}, \{e_{17}, e_{72}, e_{28}, e_{81}\}), \gamma_2 = (\{r_8, r_3, r_4\}, \{e_{83}, e_{34}, e_{48}\})$ and $\gamma_3 = (\{r_8, r_6, r_7, r_2\}, \{e_{86}, e_{67}, e_{72}, e_{28}\})$. We note a second level cycle $\gamma_1^2 = (N_1^2, E_1^2)$ where $N_1^2 = \{n_1^2, n_2^2\}$ and $E_1^2 = \{e_{12}^2, e_{21}^2\}$. Since $N_1 \cap N_2 = \{r_8\}, \gamma_1^2$ is in Γ^2.

Remark 2: We have shown that, if $\boldsymbol{q}$ is a SLD state, there exists an ordered cycle set Γ_o in D_W satisfying Th2a) and Th2b). Furthermore, Definition 5 shows that Γ_o corresponds to a second level cycle (say γ_n^2) from Γ^2. Moreover, for each $\gamma_h \in \Gamma_o$ it holds: $O_{\boldsymbol{q}}(\gamma_h) = [C(\gamma_h) - 1]$ (see Lemma 1 in Appendix). Thus the topological properties of set Γ_o and (8) and (9) lead to: $O_{\boldsymbol{q}}(\gamma_n^2) = [C(\gamma_n^2) - 1]$.

The following corollary is a consequence of Remark 2.

Corollary 3: Let $\boldsymbol{q}$ be a SLD state. The Second Level Digraph D_W^2 has a cycle $\gamma_n^2 \in \Gamma^2$, whose overlap degree in the state $\boldsymbol{q}$ is $O_{\boldsymbol{q}}(\gamma_n^2) = [C(\gamma_n^2) - 1]$.

V. Deadlock Avoidance Techniques

Efficient use of resources in flexible production systems requires a real-time control strategy optimizing throughput, resource utilization, etc. In this section, our concern is avoiding deadlock rather than developing a complete strategy for resource management. In other words, our aim is to propose real-time control rules that avoid deadlock by allocating resources at each event occurrence, on the basis of information on the current system state.

As stated earlier in Section II, resource acquiring/releasing events are relevant for the underlying DEDS model. In particular, the DEDS model must consider two event types:

1) a new job enters the system (1-type event);
2) a job progresses from a resource to another one, or it leaves the system (2-type event).

We identify 1-type event by a pair $(j, \boldsymbol{w})$, where $j \in J$ is the job entering S and $\boldsymbol{w} \in W$ is the working procedure that the job has to follow. Furthermore, we specify 2-type event by a *feasible* transition $e_{im} \in E_F(\boldsymbol{q})$ where $\boldsymbol{q}$ is the current state of S. An event is *inhibited* if we keep it from occurring. For example, event $(j, \boldsymbol{w})$ is inhibited when we prevent the job $j \in J$ from entering S to execute the working procedure $\boldsymbol{w}$. Similarly, $e_{im} \in E_F(\boldsymbol{q})$ is inhibited when we do not allow the transition e_{im}. Vice versa we define *enabled* the not inhibited events.

At ths point we define the Control Rules by introducing the sets $X_1 \subset Q \times W$ and $X_2 \subset Q \times E_W$ as follows:

$$X_1 = \{(\boldsymbol{q}, \boldsymbol{w}) \in Q \times W\text{: the first resource of } \boldsymbol{w} \text{ is } idle \text{ in } \boldsymbol{q}\}$$
$$X_2 = \{(\boldsymbol{q}, e_{im}) \in Q \times E_W\text{: } e_{im} \in E_F(\boldsymbol{q})\}.$$

Then the Control Rule for 1-type events is a function

$$f_1\text{: } X_1 \to \{0, 1\} \tag{11}$$

where $f_1(\boldsymbol{q}, \boldsymbol{w}) = 0$ $(f_1(\boldsymbol{q}, \boldsymbol{w}) = 1)$ means that, for S in the state $\boldsymbol{q}$, every 1-type event associated with $\boldsymbol{w}$ is inhibited (enabled). Similarly, a Control Rule for 2-type events is a function

$$f_2\text{: } X_2 \to \{0, 1\} \tag{12}$$

where $f_2(\boldsymbol{q}, e_{im}) = 0$ $(f_2(\boldsymbol{q}, e_{im}) = 1)$ indicates that, for S in the state $\boldsymbol{q}$, 2-type event associated with the *feasible* transition e_{im} is inhibited (enabled).

We name restriction policy a pair (f_1, f_2). Clearly, a restriction policy can select enabled events to avoid deadlock. However, since it keeps events from occurring, a restriction policy itself can eventually lead to a situation similar to a deadlock, which is known as restricted deadlock (RD) in [1]. A trivial example is the following: assume that the Control Rule f_2 be $f_2(\boldsymbol{q}, e_{im}) = 0$ for every $(\boldsymbol{q}, e_{im}) \in X_2$. Clearly, no enabled transition exists and jobs in process remain indefinitely blocked, since no resource can be released. The next definition formulates this concept in details.

Definition 7: Let $\boldsymbol{q} \in Q$ be not a deadlock state for S and let (f_1, f_2) be a restriction policy. We say that $\boldsymbol{q}$ is a RD state for S under (f_1, f_2), if:

D7a) there exist two nonempty sets $J_R \subset J_{\boldsymbol{q}}$ and $R_R \subset R$ such that $\mathrm{HR}(J_R) \subset R_R$, $\mathrm{SR}(J_R) \subset R_R$ and all the resources in $[R_R - \mathrm{HR}(J_R)]$ are *idle;*

D7b) for each $j \in J_R$ with $\mathrm{SR}(j) \in [R_R - \mathrm{HR}(J_R)]$, f_2 inhibits the *feasible* transition from $\mathrm{HR}(j)$ to $\mathrm{SR}(j)$. Moreover, such transition remains inhibited if no job in J_R releases the resource that it currently holds.

The conditions of the above definition lead to "indefinite blocking." Namely, each job $j \in J_R$ keeps on holding $\mathrm{HR}(j)$ indefinitely, because it requires a transition which is blocked or inhibited. Such a transition can not become feasible and enabled unless a job in J_R releases its currently held resource. So a typical circular wait occurs.

Remark 3: By Definition 7, only the Control Rule f_2 may lead to a RD. Restricted deadlock, indeed, is independent of f_1. Of course, if $f_2(\boldsymbol{q}, e_{im}) = 1$ for every $(\boldsymbol{q}, e_{im}) \in X_2$ (i.e., f_2 does not inhibit any transition), there exists no RD state for S.

Obviously, deadlock avoidance must prevent both deadlock and RD from occurring. This motivates the following definition.

Definition 8: Let $Q_0 \subset Q$ and (f_1, f_2) be a restriction policy. Moreover, assume that, for every $\boldsymbol{q_0} \in Q_0$, any state $\boldsymbol{q} \in Q$ reachable from $\boldsymbol{q_0}$ under (f_1, f_2) is not a deadlock or a RD state. In this case, we say that S, starting at Q_0 under the restriction policy (f_1, f_2), is deadlock-free.

Now we are ready to introduce the first two restriction policies that falsify necessary conditions for a deadlock occurrence. The first one is quite simple and puts a bound on the jobs in process.

Restriction Policy 1 (RP1): Let $(\boldsymbol{q}, \boldsymbol{w}) \in X_1$. We put

$$\begin{aligned} f_1(\boldsymbol{q}, \boldsymbol{w}) &= 1 \quad \text{if } \mathrm{Card}(J_{\boldsymbol{q}}) < (C_0 - 1) \\ f_1(\boldsymbol{q}, \boldsymbol{w}) &= 0 \quad \text{otherwise} \\ f_2(\boldsymbol{q}, e_{im}) &= 1 \quad \text{for any } (\boldsymbol{q}, e_{im}) \in X_2. \end{aligned}$$

The following Proposition proves that RP1 avoids deadlock.

Proposition 1: Let $Q_0 \subset Q$ be the state subset $Q_0 = \{\boldsymbol{q_0} \in Q\text{: } \mathrm{Card}(J_{\boldsymbol{q_0}}) < C_0\}$. System S, starting at Q_0, is deadlock-free under the Restriction Policy 1.

Proof: By the definitions of Q_0 and of the Control Rule f_1, it holds $\mathrm{Card}(J_{\boldsymbol{q}}) < C_0$ in any reachable state $\boldsymbol{q}$. Therefore, according to Corollary 2, S can not reach a deadlock state. Finally, by Remark 3, $\boldsymbol{q}$ can not be a RD state for S under the RP1. •

At this point, we need some notations to easily introduce the second restriction policy. So if $\boldsymbol{w} \in W$, we define the following cycle set:

$$\Gamma_{\boldsymbol{w}} = \{\gamma_n \text{ of } D_W\text{: the set } E_n \cap E_{\boldsymbol{w}} \text{ is not empty}\}. \quad (13)$$

In other words, $\Gamma_{\boldsymbol{w}}$ contains cycles having at least one edge in common with $D_{\boldsymbol{w}}$. Furthermore, for any γ_n of D_W and $\boldsymbol{q} \in Q$, we introduce the following job set:

$$\begin{aligned} J_{\boldsymbol{q}}(\gamma_n) = \{j \in J_{\boldsymbol{q}}\text{: the set } E_n \cap E_{\boldsymbol{w}} \text{ is not empty} \\ \text{with } \boldsymbol{w} = \mathrm{WP}(j)\}. \end{aligned} \quad (14)$$

Thus, $J_{\boldsymbol{q}}(\gamma_n)$ contains all the jobs in process under working procedures involving edges of γ_n. Given these notations we define the second restriction policy.

Restriction Policy 2 (RP2): Let $(\boldsymbol{q}, \boldsymbol{w}) \in X_1$. We put

$$\begin{aligned} f_1(\boldsymbol{q}, \boldsymbol{w}) &= 1 \quad \text{if, for any } \gamma_n \in \Gamma_{\boldsymbol{w}}, \text{ it holds:} \\ &\qquad \mathrm{Card}[J_{\boldsymbol{q}}(\gamma_n)] < C(\gamma_n) - 1; \\ f_1(\boldsymbol{q}, \boldsymbol{w}) &= 0 \quad \text{otherwise} \\ f_2(\boldsymbol{q}, e_{im}) &= 1 \quad \text{for any } (\boldsymbol{q}, e_{im}) \in X_2. \end{aligned}$$

The next result shows how we can apply the above restriction policy to prevent the occurrence of events leading to deadlock.

Proposition 2: Let $Q_0 \subset Q$ be the state subset

$$Q_0 = \{\boldsymbol{q_0} \in Q\text{: } \mathrm{Card}[J_{\boldsymbol{q_0}}(\gamma_n)] < C(\gamma_n) \text{ for each } \gamma_n \text{ of } D_W\}.$$

System S, starting at Q_0, is deadlock-free under the Restriction Policy 2.

Proof: Comparing Definition 3 and (14) leads to $O_{\boldsymbol{q}}(\gamma_n) \leq \mathrm{Card}[J_{\boldsymbol{q}}(\gamma_n)]$. Thus, owing to the condition on $\boldsymbol{q_0}$ and to the application of RP2, we have $O_{\boldsymbol{q}}(\gamma_n) < C(\gamma_n)$ for any cycle γ_n of D_W and for any state $\boldsymbol{q}$ reachable from $\boldsymbol{q_0}$. Hence by Corollary 1, S can not reach a deadlock state. In addition, by Remark 3, $\boldsymbol{q}$ can not be a RD state for S under the RP2. •

Next we consider a restriction policy that controls both 1-type and 2-type events to avoid deadlocks. To put the least restrictive constraints on the resource assignment, this control rule avoids only transitions immediately leading to a deadlock, by using a one-step look ahead procedure.

Restriction Policy 3 (RP3):

$$\begin{aligned} f_1(\boldsymbol{q}, \boldsymbol{w}) &= 1 \quad \text{if } D_{\mathrm{Tr}}(\boldsymbol{q}, \boldsymbol{w}) \text{ does not contain any cycle} \\ f_1(\boldsymbol{q}, \boldsymbol{w}) &= 0 \quad \text{otherwise} \\ f_2(\boldsymbol{q}, e_{im}) &= 1 \quad \text{if } D_{\mathrm{Tr}}(\boldsymbol{q}, e_{im}) \text{ does not contain any cycle} \\ f_2(\boldsymbol{q}, e_{im}) &= 0 \quad \text{otherwise.} \end{aligned}$$

Although RP3 prevents deadlock from occurring, it might not avoid RD as the following example clearly shows.

Example 8: Consider the system S and the state $\boldsymbol{q}$ of Example 4. By Definition 9, it follows that $\boldsymbol{q}$ is a RD state for S under RP3. Indeed putting $J_R = J_{\boldsymbol{q}}, R_R = \{r_i \ i = 1, 2, \cdots, 8\}$, we get $\mathrm{HR}(J_R) = \{r_i \ \ i = 1, 2, \cdots, 7\} \subset R_R, \mathrm{SR}(J_R) = \{r_2, r_4, r_7, r_8\} \subset R_R, [R_R - \mathrm{HR}(J_R)] = \{r_8\}$ where r_8 is *idle*. Moreover, the function f_2 of RP3 inhibits the *feasible* transitions e_{28}, e_{48} and e_{58}. Such transitions remain inhibited if no job in J_R releases the resource it currently holds.

The above considerations call for a characterization of RD. The next proposition shows that if system S is under RP3, RD and SLD coincide.

Proposition 3: Let us define f_2 according to Restriction Policy 3. For any f_1, necessary and sufficient condition for $\boldsymbol{q} \in Q$ to be a RD state for S under (f_1, f_2) is that $\boldsymbol{q}$ is a SLD state.

Proof: Necessity.

Let $\boldsymbol{q}$ be a RD state for S under (f_1, f_2). By Definition 7, there exist two subsets $J_R \subset J_{\boldsymbol{q}}$ and $R_R \subset R$ satisfying D7a). Moreover, for each $j \in J_R$ with $\mathrm{SR}(j) \in [R_R - \mathrm{HR}(J_R)]$, RP3 gives $f_2(\boldsymbol{q}, e_{im}) = 0$, where $r_i = \mathrm{HR}(j)$ and $r_m = \mathrm{SR}(j)$. Hence, due to the definition of f_2, e_{im} leads to a deadlock. Transition e_{im} remains inhibited if no job in J_R releases the resource that it currently holds. Thus, if we ideally remove all the jobs in $(J_{\boldsymbol{q}} - J_R)$ from the system, e_{im} remains still inhibited. This leads us to recognize that the deadlock condition of $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$ is characterized by a job subset $J_D \subset J_R$. When compared with Definition 4 in which $J_S = J_R$ and $R_S = R_R$, the above considerations prove the necessity.

Sufficiency

Bearing in mind Definition 4, the proof follows on using the same arguments developed for proving necessity. •

Note that if Γ^2 is empty, by Corollary 3 no SLD can arise and, in this case, RP3 is deadlock-free. Moreover it is the least restrictive policy one can find because it inhibits only events leading immediately to a deadlock. In all the remaining cases we must modify f_1 to falsify one or more of the necessary

conditions for a SLD occurrence. To this point, putting

$$C_0^2 = \min_{\Gamma^2} C(\gamma_n^2) \tag{15}$$

(with $C_0^2 = \infty$ if Γ^2 is empty) we define a further restriction policy.

Restriction Policy 4 (RP4):

$f_1(\boldsymbol{q}, \boldsymbol{w}) = 1$ if $D_{\mathrm{Tr}}(\boldsymbol{q}, \boldsymbol{w})$ does not contain any cycle and $\mathrm{Card}(J_{\boldsymbol{q}}) < (C_0^2 - 2)$;
$f_1(\boldsymbol{q}, \boldsymbol{w}) = 0$ otherwise;
$f_2(\boldsymbol{q}, e_{im}) = 1$ if $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$ does not contain any cycle;
$f_2(\boldsymbol{q}, e_{im}) = 1$ otherwise.

The next result shows that no deadlock may occur under RP4.

Proposition 4: Let $Q_0 \subset Q$ be the state subset:

$$Q_0 = \{\boldsymbol{q_0} \in Q\text{: } \mathrm{Card}(J_{\boldsymbol{q_0}}) < (C_0^2 - 1) \text{ and } \boldsymbol{q_0} \text{ is not a deadlock state}\}$$

Then the system S, starting from Q_0 is deadlock-free under the Restriction Policy 4.

Proof: Let $\boldsymbol{q} \in Q$ be any state reachable from $\boldsymbol{q_0} \in Q_0$ under RP4. Such a Policy prevents $\boldsymbol{q}$ from being a deadlock state. In addition, the definition of the Control Rule f_1 leads to $\mathrm{Card}(J_{\boldsymbol{q}}) < (C_0^2 - 1)$. Since $O_{\boldsymbol{q}}(\gamma_n^2) \leq \mathrm{Card}(J_{\boldsymbol{q}})$, the necessary condition of Corollary 3 is violated and, by Proposition 3, $\boldsymbol{q}$ is not a RD state for S under RP4. This completes the proof. •

To introduce the next restriction policy, it is necessary to define two more sets. So let $\gamma_n^2 = (N_n^2, E_n^2)$ be a second level cycle and let $\Gamma_n = \{\gamma_1, \gamma_2, \cdots, \gamma_{P_n}\}$ be the set of cycles in D_W associated with vertices from N_n^2. For any $\boldsymbol{w} \in W$, we define the following set:

$$\Gamma_{\boldsymbol{w}}^2 = \left\{\gamma_n^2 \in \Gamma^2\text{: } \left(\bigcup_{h=1}^{P_n} E_h\right) \cap E_{\boldsymbol{w}} \text{ is not empty}\right\}. \tag{16}$$

In other terms, $\Gamma_{\boldsymbol{w}}^2$ collects all the second level cycles from Γ^2 corresponding to sets of first level cycles, where each set has at least one edge in common with $E_{\boldsymbol{w}}$. Moreover, for any second level cycle γ_n^2 and any state $\boldsymbol{q} \in Q$, we introduce the following subset of $J_{\boldsymbol{q}}$:

$$J_{\boldsymbol{q}}(\gamma_n^2) = \left\{j \in J_{\boldsymbol{q}}\text{: the set } \left(\bigcup_{h=1}^{P_n} E_h\right) \cap E_{\boldsymbol{w}} \text{ is not empty, with } \boldsymbol{w} = \mathrm{WP}(j)\right\}. \tag{17}$$

Set $J_{\boldsymbol{q}}(\gamma_n^2)$ collects all the jobs of $J_{\boldsymbol{q}}$ following working procedures, that involve some edge of the first level cycle associated with γ_n^2. At this point we can derive the fifth restriction policy.

Restriction Policy 5 (RP5):

$f_1(\boldsymbol{q}, \boldsymbol{w}) = 1$ if $D_{\mathrm{Tr}}(\boldsymbol{q}, \boldsymbol{w})$ does not contain any cycle and it holds: $\mathrm{Card}[J_{\boldsymbol{q}}(\gamma_n^2)] < [C(\gamma_n^2) - 2]$ for every $\gamma_n^2 \in \Gamma_{\boldsymbol{w}}^2$;
$f_1(\boldsymbol{q}, \boldsymbol{w}) = 0$ otherwise
$f_2(\boldsymbol{q}, e_{im}) = 1$ if $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$ does not contain any cycle
$f_2(\boldsymbol{q}, e_{im}) = 0$ otherwise.

Given the above definition, we prove the next result.

Proposition 5: Let $Q_0 \subset Q$ be the state subset

$$Q_0 = \{\boldsymbol{q_0} \in Q\text{: } \mathrm{Card}[J_{\boldsymbol{q_0}}(\gamma_n^2)] < [C(\gamma_n^2) - 1] \text{ for every } \gamma_n^2 \in \Gamma^2 \text{ and } \boldsymbol{q_0} \text{ is not a deadlock state.}\}$$

System S, starting from Q_0, is deadlock-free under Restriction Policy 5.

Proof: The condition on the initial state $\boldsymbol{q_0}$ and the RP5 guarantee that each $\boldsymbol{q}$ reachable from $\boldsymbol{q_0}$ is not a deadlock state. Moreover, since $O_{\boldsymbol{q}}(\gamma_n^2) \leq \mathrm{Card}\,[J_{\boldsymbol{q}}(\gamma_n^2)]$, it holds $O_{\boldsymbol{q}}(\gamma_n^2) < [C(\gamma_n^2) - 1]$ for any cycle $\gamma_n^2 \in \Gamma^2$. Therefore, by Corollary 3, a SLD can not occur and, by Proposition 3, neither $\boldsymbol{q}$ can be a RD state for S under RP5. Hence the Proposition. •

Note that, if Γ^2 is empty ($C_0^2 = \infty$), RP3, RP4, and RP5 coincide.

Before concluding this section, we remark that RP1 and RP2 need no look ahead. Moreover RP3, RP4, and RP5 use a simple look ahead of one step only.

VI. Computational Complexity

The computational complexity deeply affects the chances that a deadlock avoidance policy shall succeed. In particular, control laws with low computational effort are necessary in real-time applications. To discuss the computational complexity of the just introduced restriction policies, we distinguish between on-line and off-line costs. The first ones concern the real-time algorithms. The second ones characterize the algorithms which are employed once, before the proper real-time control.

For what concerns RP1, the only off-line computation involves the minimum capacity C_0. This parameter can be easily computed by searching the (nontrivial) shortest path starting from each vertex in D_W and ending in the same vertex. An algorithm of such a kind requires $O\{[\mathrm{card}(N)]^3\}$ operations [13]. On the other hand, the on-line burden lies only in memorizing and in updating the number of jobs in progress $\mathrm{Card}(J_{\boldsymbol{q}})$.

To apply RP2, it is necessary to determine off-line the cycles from D_W with their capacities and to build up a relation between working procedures and cycles, for identifying the sets $\Gamma_{\boldsymbol{w}}$ [see (13)]. Since the relationship between edges of D_W and working procedures is an input datum, determining cycles from D_W becomes the main problem. To this aim, the technical literature provides algorithms for generating cycles in $O\{[\mathrm{Card}(N)+\mathrm{Card}(E)](c_1+1)\}$ time, where c_1 represents the number of cycles from D_W [13]. Of course, the complexity

of these algorithms becomes prohibitive, if c_1 is very high (e.g., if D_W is complete). However, in many real systems, the adjacency matrix of D_W is fairly sparse so that determining all the cycles in such a digraph is not too time-consuming. The on-line computational costs required by RP2 consists only in updating the number of jobs in process corresponding to each working procedure. Namely, the relation between working procedures and cycles determined off-line allows us to calculate $\mathrm{Card}[J_{\boldsymbol{q}}(\gamma_n)]$ for any cycle from D_W.

RP3 demands no off-line computations. Moreover, it needs a little on-line computational effort for transforming $D_{\mathrm{Tr}}(\boldsymbol{q})$ into $D_{\mathrm{Tr}}(\boldsymbol{q},\boldsymbol{w})$ (or $D_{\mathrm{Tr}}(\boldsymbol{q},e_{im})$), (see Section II) and for checking if such a new digraph contains a cycle. A depth-first search algorithm [13] can easily perform this check, starting with the edge just added to $D_{\mathrm{Tr}}(\boldsymbol{q})$ for obtaining $D_{\mathrm{Tr}}(\boldsymbol{q},\boldsymbol{w})$ or $D_{\mathrm{Tr}}(\boldsymbol{q},e_{im})$. Namely, the complexity of the depth-first search algorithm is $O[\mathrm{Card}(N)]$, because the outdegree of the vertices of $D_{\mathrm{Tr}}(\boldsymbol{q})$ equals zero or one.

To implement RP4 it is necessary to generate off-line the cycles from D_W and the digraph D_W^2. Moreover, the cycles from D_W^2 with their capacities, the subset Γ^2 and the minimum capacity C_0^2 must be specified. Building D_W^2 can be performed in $O[(c_1)^2 L]$ operations, where L indicates the sum of the lengths of all the working procedures (i.e., the sum of resources appearing in all the working procedures, counting repetitions). Moreover, generating the cycles of D_W^2 and characterizing Γ^2 need $O\{[c_1+\mathrm{Card}(E_W^2)](c_2+1)\}$ and $O(c_1\ c_2)$ operations, respectively, where c_2 indicates the number of second level cycles. Finally, applying RP4 requires the same on-line computations as RP3 and, in addition, the on-line updating of $\mathrm{Card}(J_{\boldsymbol{q}})$.

RP4 and RP5 are based on the same off-line computation steps. Moreover, to use RP5 it is necessary to determine the relationship between working procedures and second level cycles, for characterizing the sets $\Gamma^2_{\boldsymbol{w}}$ (see (16)). Such a relationship and the number of jobs in process for each working procedure allow us to update the cardinality of each set $J_{\boldsymbol{q}}(\gamma_n^2)$ on-line, as a job enters or leaves the system. RP5 also demands the on-line modification of the Transition Digraph and the detection of its possible cycles.

To sum up, the off-line computational cost is small, in the case of RP1 and RP3, while it depends on the number of cycles from D_W and D_W^2, in the case of RP2, RP4, and RP5. When the number of cycles is not too high (as it is the case in many real applications), simple algorithms are known for performing the off-line computations. Since the five policies have small on-line computational costs, they are all suitable for real-time applications.

Finally, a remark about RP3 is appropriate. Namely, this policy uses a one-step look ahead procedure, it is simple and requires low computational cost but, as we have already noted, it needs that Γ^2 is empty. In such a case, RP3 is the least restrictive policy one can implement: namely, it inhibits only the events immediately leading to a deadlock. Finally, it is also interesting that Γ^2 is empty in many cases. F.e. this happens in FMS's consisting only in some workstations equipped with input (or output) buffers. In fact, the vertex corresponding to any work station is always preceded (or followed) by the node corresponding to the buffer in the Working Procedure Digraph. For this reason, cycles from D_W sharing only one vertex are not allowed.

VII. Examples

Now we present three examples to show the characteristics of restriction policies introduced in the previous section. In the first two examples RP3 is deadlock-free. Hence it is the less restrictive policy one can use. On the contrary, RP3 can not be applied to the third example because Γ^2 is not empty. So, assuming throughput as performance index, we use simulation to compare Restriction Policies 1, 2, 4, 5, and the control law introduced by Banaszak and Krogh (RPBK) [1].

Before introducing the examples, we briefly discuss some modeling issues that are relevant in the context of our interest. The approach proposed in this paper considers only systems with single resources. The Example C in the following shows a case study of this kind.

A problem that arises is related to the representation of a set of multiple items composing the same resource type (multiple resource). This issue is of undoubted interest because the automated manufacturing systems may have many multiple resources (i.e., buffer with multiple spaces, multiserver machines, etc.). At a first glance, a straightforward solution to the problem consists in replacing a multiple resource with as many independent resources as the corresponding component items. However, the drawback of this modeling method lies in a considerable rise in the number of working procedures representing the paths of jobs in the system. Indeed, since a job requiring a multiple resource can occupy any one of the single independent items corresponding to this resource, the route of each job type formally generates a set of working procedures. This makes the method unsuitable from a practical point of view.

In some cases, however, we can easily model multiple resources by means of a fairly limited extension of our formalism. To this aim, let us observe that the different items of a multiple resource are indistinguishable from the functional point of view; that the jobs leave a multiple resource one at a time; and finally, that a multiple resource is deadlocked only if all its component items are *busy*. The previous considerations allow us to model a multiple resource as a sequence of unit resources or, equivalently, by only one vertex in D_W and in $D_{\mathrm{Tr}}(\boldsymbol{q})$. Such a single vertex must be considered *busy* only if the multiple resource is completely full and must be regarded as *idle* in all the remaining cases. This statement is completely consistent with the model developed in this paper, provided that the destination of the first job leaving the resource is known. The problem is easily solved in the following situations.

1) The order in which jobs leave the multiple resource is fixed. In this case the next transition of the first job releasing the resource identifies the edge in $D_{\mathrm{Tr}}(\boldsymbol{q})$ uniquely. e.g., this happens for jobs in a buffer subject to a First In First Out discipline.
2) All the jobs hosted by the multiple resource require the same next resource in the Residual Working Procedure.

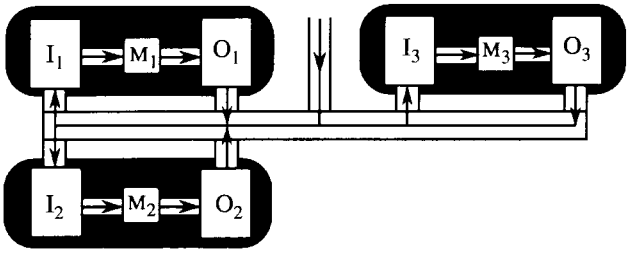

Fig. 7. Flexible Manufacturing System with three work cells (Example A).

Clearly, the edge representing the corresponding transition is uniquely determined. e.g., this condition holds true when an input buffer serves only one machine (or a set of identical machines represented by a vertex).

Obviously, in determining the capacity of first and second level cycles from D_W, a vertex representing a multiple resource must be taken into account as many times as the number of items of the resource. Examples A and B in the following show two systems in which each multiple resource is represented by only one vertex.

If conditions 1) and 2) do not hold, then the results of this paper can not be used in their present form, to deal with multiple-resource systems. In this case it is necessary to extend such results, using a more general model.

Example A: Recently, Hsieh and Chang [8] have discussed the following example, formerly considered by Banaszak and Krogh [1]. A flexible manufacturing system contains three work cells $\mathrm{WC}_1, \mathrm{WC}_2$ and WC_3 (see Fig. 7), where each work cell (WC_i $i = 1,2,3$) consists of a machine M_i, an input buffer I_i and an output buffer O_i. The capacity of both I_i and O_i is of five spaces. This system produces jobs of a single type, following the route

$$I_3 \to M_3 \to O_3 \to I_1 \to M_1 \to O_1 \to I_2 \to M_2 \to O_2 \to I_1 \to M_1 \to O_1.$$

We note that each input buffer can be described by a single vertex of D_W, because, according to the previous note (see 2)), each input buffer is followed by only one resource (i.e., by the corresponding machine). Buffers O_2 and O_3 can be modeled in the same way because they are always followed by I_1. For what concerns O_1, this buffer is followed either by I_2 or by the fictitious resource, whch takes into account jobs at the end of the working procedure. Since on the processing completion a job immediately leaves the system, also the buffer O_1 can be modeled by only one vertex. Clearly, there are two edges outgoing from this vertex. The former reaches I_2 and the latter ends in the fictitious resource (r_{10}). Fig. 8 shows the working procedure digraph, also indicating the resource capacity of the multiple resources. In such a digraph there is one cycle (γ) involving WC_1 and WC_2, whose capacity $C(\gamma) = 22$ is the total amount of the capacities of I_1, M_1, O_1, I_2, M_2 and O_2. Clearly in this case the second level digraph is acyclic so that Γ^2 is empty and RP3 is deadlock-free. As we have already pointed out, RP3 is the least restrictive policy one can define, because it limits the freedom in resource allocation as little as necessary to avoid deadlock. Namely, other policies put unnecessary constraints on the resource allocation. e.g.,

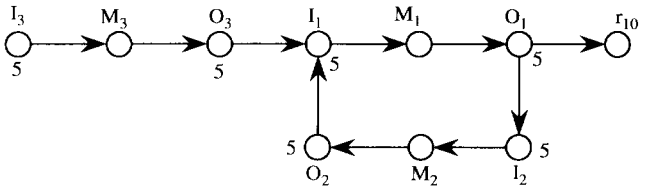

Fig. 8. Working Procedure Digraph for Example A.

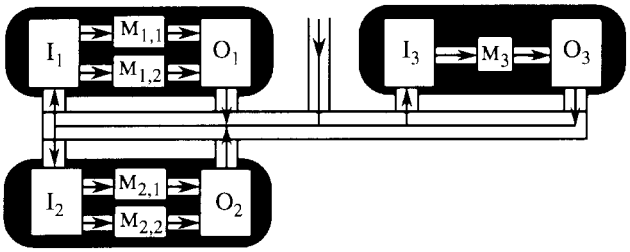

Fig. 9. Flexible Manufacturing System with three work cells (Example B).

in the policy proposed by Hsieh and Chang [8], the actions inhibiting or enabling transitions (i.e., the validity of a "control action") depend on the Sufficient Validity Test Algorithm. This algorithm requires that a sufficient but not necessary condition for deadlock avoidance be satisfied. More precisely, as the authors remark, "a control action may pass the Sufficient Validity Test Algorithm using one specific priority rule while fails the test using another one." Moreover for what concerns the algorithm introduced by Banaszak and Krogh, this policy puts nonnecessary constraints in using resources. In fact, as shown by Hsieh and Chang, such a policy can inhibit some transitions even if they do not lead to deadlock.

Example B: The system shown in Fig. 9 is made up of three work cells ($\mathrm{WC}_1, \mathrm{WC}_2$ and WC_3). A two-space input buffer (I_1), two identical machines ($M_{1,1}$ and $M_{1,2}$) and a single-space output buffer (O_1) constitute the work cell WC_1. Analogously, the work cell WC_2 has an input buffer with two spaces, two identical machines ($M_{2,1}$ and $M_{2,2}$) and a single-space output buffer (O_2). Finally, work cell WC_3 includes a two-space input-buffer (I_3), a machine M_3 and a two-space output buffer (O_3). The system produces two job-types (P_1 and P_2) following the routes:

$$\text{Route}(P_1): I_1 \to M_{1,1} \text{ (or } M_{1,2}) \to O_1 \to I_2 \to M_{2,1} \text{ (or } M_{2,2}) \to O_2 \to I_3 \to M_3 \to O_3$$

$$\text{Route}(P_2): I_3 \to M_3 \to O_3 \to I_2 \to M_{2,1} \text{ (or } M_{2,2}) \to O_2 \to I_1 \to M_{1,1} \text{ (or } M_{1,2}) \to O_1.$$

Machines $M_{1,1}$ and $M_{1,2}$ are identical and can be considered as a single resource having multiplicity equal to two. Since they are always followed by buffer O_1, machines $M_{1,1}$ and $M_{1,2}$ correspond to a single vertex (say M_1), in D_W and $D_{\mathrm{Tr}}(\boldsymbol{q})$. Using the same arguments, we represent $M_{2,1}$ and $M_{2,2}$ with a unique vertex (say M_2). Of course, since the buffers I_1, I_2 and I_3 are, respectively, followed by M_1, M_2 and M_3, each of them corresponds to a unique vertex. Finally the buffer O_3 is a multiple resource, with capacity equal to two, containing jobs addressed to I_2 and jobs which have completed their processing. As in the previous example, we

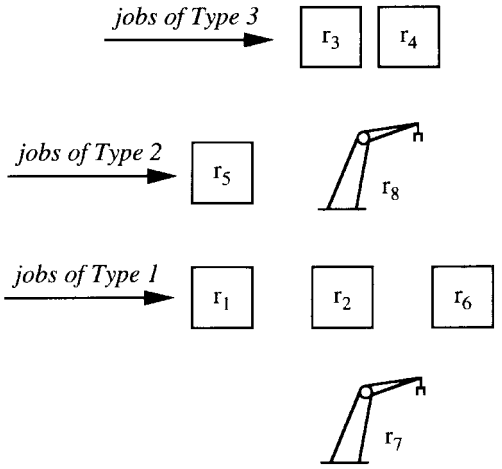

Fig. 10. A robotized cell (Example C).

can consider O_3 as a vertex with two edges outgoing from it. The former reaches the resource I_2 and the latter ends in the fictitious resource. Note that each machine is preceded by a unique resource (its input buffer) and is also followed by a unique resource (its output buffer). For this reason, without drawing D_W^2, we can conclude that the Working Procedure Digraph contains no set of cycles sharing only one resource. So, as in the previous example, the set Γ^2 is empty and we can apply RP3, with the minimum constraints in the freedom to allocate resources.

Example C: Consider a system with six machines ($r_1, r_2, r_3,\ r_4, r_5,$ and r_6) and two robots r_7) and r_8) which transfer jobs from a machine to another one (see Fig. 10). Hence we have R $= 9$, where r_9 is the fictitious resource. There are three jobs to produce, following the working procedures $\boldsymbol{w_1}$, $\boldsymbol{w_2}$ and $\boldsymbol{w_3}$, respectively. Each 1-type job is automatically loaded on machine r_1, where the first operation takes place. Then the robot r_7 transfers the job to the machine r_2. Afterwards the robot r_8 transports the job to machine r_3. From here the piece directly goes on to machine r_4 and successively it is unloaded by robot r_8 into an output buffer (of infinite capacity). Analogously each 2-type job is directly loaded on machines r_5, transferred to machine r_6 by robot r_8 and unloaded into the output buffer by robot r_7. Finally, each 3-type job is loaded on machine r_3 and, from here directly transferred to machine r_4. Later robot r_8 carries the job to machine r_1 and robot r_7 unloads it into the output buffer. Therefore the working procedures are $\boldsymbol{w_1} = \{r_1, r_7, r_2, r_8,\ r_3, r_4, r_8, r_9\}$, $\boldsymbol{w_2} = \{r_5, r_8, r_6,\ r_7, r_9\}$ and $\boldsymbol{w_3} = \{r_3, r_4,\ r_8, r_1, r_7, r_9\}$.

The corresponding Working Procedure Digraph has been used in Example 4 and is shown by Fig. 3. It has three cycles $\gamma_1 = (\{r_1, r_7, r_2, r_8\},\ \{e_{17}, e_{72}, e_{28}, e_{81}\})$, $\gamma_2 = (\{r_8, r_3, r_4\}, \{e_{83},\ e_{34}, e_{48}\})$ and $\gamma_3 = (\{r_8, r_6, r_7, r_2\}, \{e_{86}, e_{67}, e_{72}, e_{28}\})$. The Second Level Digraph contains a unique cycle $\gamma_1^2 = (N_1^2, E_1^2)$ where $N_1^2 = \{n_1^2, n_2^2\}$ and $E_1^2 = \{e_{12}^2, e_{21}^2\}$, as Fig. 6 and Example 7 clearly indicate. Moreover, since γ_1^2 belongs to Γ^2, RP3 can not be applied to this case. Hence we report the results of a simulation study comparing RP1, RP2, RP4, RP5 and the control law proposed by Banaszak and Krogh.

TABLE I
MEAN AND STANDARD DEVIATION OF SERVICE TIMES FOR CASE 1 (EXAMPLE C)

r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8
μ=45	μ=90	μ=45	μ=45	μ=90	μ=90	μ=30	μ=22.5
σ=9	σ=18	σ=9	σ=9	σ=18	σ=18	σ=6	σ=4.5

To apply RP1, we observe that the minumum capacity of cycles from D_W is $C_0 = 3$. Hence such a policy does not allow more than two jobs in the system.

For what concerns RP2, we build up the relationship between working procedures and cycles, which identifies the following sets: $\Gamma_{\boldsymbol{w_1}} = \{\gamma_1, \gamma_2, \gamma_3\}, \Gamma_{\boldsymbol{w_2}} = \{\gamma_3\}$ and $\Gamma_{\boldsymbol{w_3}} = \{\gamma_1, \gamma_2\}$. According to this policy, a new 1-type job can enter the system only if it holds $\mathrm{Card}[J_{\boldsymbol{q}}(\gamma_1)] < 3, \mathrm{Card}[J_{\boldsymbol{q}}(\gamma_2)] < 2$ and $\mathrm{Card}[J_{\boldsymbol{q}}(\gamma_3)] < 3$. Analogously a new 2-type job can be loaded into the system only if the relation $\mathrm{Card}[J_{\boldsymbol{q}}(\gamma_3)] < 3$ is verified. Finally, a new 3-type job can be supplied to the system only if the constraints $\mathrm{Card}[J_{\boldsymbol{q}}(\gamma_1)] < 3$ and $\mathrm{Card}[J_{\boldsymbol{q}}(\gamma_2)] < 2$ are satisfied. Note that $J_{\boldsymbol{q}}(\gamma_1) = J_{\boldsymbol{q}}(\gamma_2)$ because both sets identify 1-type and 3-type in-process jobs. On the other hand, the set $J_{\boldsymbol{q}}(\gamma_3)$ collects 1-type and 2-type in-process jobs.

To implement RP4 we have to determine the capacity of the second level cycle. Since $C_0^2 = C(\gamma_1^2) = \mathrm{Card}(N_1 \cup N_2) = 6$, such a policy allows four jobs in the system simultaneously.

To apply RP5 we determine the following sets: $\Gamma^2_{\boldsymbol{w_1}} = \Gamma^2_{\boldsymbol{w_3}} = \{\gamma_1^2\}$, while $\Gamma^2_{\boldsymbol{w_2}}$ is empty. Hence a 1-type (or a 3-type) job can enter the system iff $\mathrm{Card}[J_{\boldsymbol{q}}(\gamma_1^2)] < 4$. On the contrary, a 2-type job can always be loaded into the system. Note that $J_{\boldsymbol{q}}(\gamma_1^2)$ collects all the first-type and the third-type jobs in-process.

The quality of the policies described above is shown by implementing a SIMAN simulation model [12] and by assuming the system throughput as performance index. The following conditions rule the simulation. First, the job types enter the system following the periodic order: types 1, 2, 3, 1, 2, 3, etc. Second, a job is loaded into the system as soon as the first resource of its working procedure is *idle*. Finally, the law "First In First Out" rules the service priority setting. We simulate the system in four different working conditions denoted by Cases 1, 2, 3, and 4. Service times for each resource are generated randomly by a gamma distribution with mean μ and standard deviation σ. In particular, in Case 1 the service times balance the workloads of the eight resources (machines and robots). Table I reports μ and σ for each resource. On the other hand, Cases 2, 3, and 4 are unbalanced and all the resources, but one, have the same service times as Case 1. In particular, Case 2 assumes robot r_7 as bottleneck, with service time distribution characterized by $\mu = 4.5$ and $\sigma = 0.9$. Moreover, resources r_8 (with $\mu = 3$ and $\sigma = 0.6$) and r_2 (with $\mu = 10$ and $\sigma = 2$) are, respectively, bottlenecks in Case 3 and Case 4.

In each simulation we use the method of batch means to compute the 95% confidence intervals for throughputs. To this aim, after a transient period corresponding to the completion of 500 jobs, we simulate 60 000 completed parts, divided into 40 batches of the same size. Fig. 11 shows the throughputs for each working condition. The width of the confidence interval

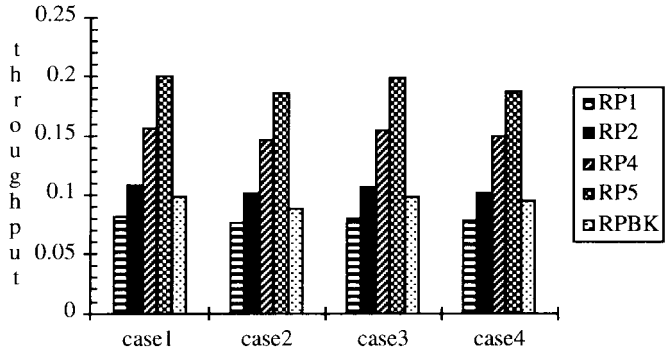

Fig. 11. Example C: throughput values (jobs per time unit).

is less than 0.6% in any simulation, and confirms the good precision of the throughput estimations. Fig. 11 also shows that throughput becomes higher and higher using restriction policies in the following order: RP1, RP2, RP4, and RP5. Moreover RPBK is better than RP1, but it gives poor results especially if compared with RP4 and RP5.

VIII. Conclusions

In this paper we focus on the circular wait problem in automated production systems, by using a digraph theoretic approach to develop necessary and sufficient conditions for the occurrence of deadlocks and Second Level Deadlocks. In particular, the definition of the Transition Digraph allows us to enlighten the mechanism of interaction among jobs and resources that leads to such critical situations. We assume that the dynamics of the production system is described by a DEDS model whose state provides the information necessary to build the Transition Digraph. Only events that involve resource releasing or resource acquiring modify such a digraph. In this context, the control activity consists in inhibiting or in enabling the above events in dependence of the current system's state. In other words, the control laws appear as restriction policies preventing event occurrences that lead to deadlocks. The basic idea is that such policies must falsify one of the necessary conditions for the deadlock, derived in the paper. At the same time, however, one must avoid that a condition of indefinite blocking occurs as consequence of the restriction policies themselves.

Any scheduling policy not involving events inhibited by deadlock-free restriction policies is deadlock-free. Thus each restriction policy implicitly identifies a class of scheduling strategies. In Section IV we propose five restriction policies, involving different computational complexities and performances. RP1 and RP3 have small off-line costs. On the other hand, the off-line computations pertaining to RP2, RP4 and RP5 have exponential worst-case complexity, because the number of cycles in a digraph can be exponential in the number of vertices. However, in many practical cases such computations can be executed in a reasonably short time. Moreover, they are carried out only once, before the proper on-line control.

All the five restriction policies require small on-line computational costs, so that they are suitable for real-time implementation. An interesting result concerns RP3. This policy is deadlock-free only for particular, but not infrequent, systems. On the other hand, in these cases it is the least restrictive policy one can find. Moreover, it is very simple because it only uses a look ahead of one step.

The approach proposed in this paper considers only single resources. However, the examples developed in Section V show that in some cases it is easy to extend our formalism to take into account multiple items of the same resource type. Starting from the results of this work, a more general approach considering multiple resources in a graph-theoretic framework will be developed in a next paper nowadays in progress.

Before concluding, a final consideration is in order. All the avoidance approaches proposed in the literature offer both advantages and drawbacks. Future researches should determine classes of system configurations for which a particular avoidance method is more convenient than the remaining ones.

Appendix

Proving Theorem 2 requires some preparation that leads to three lemmas. The details of the mechanism leading to the deadlock of a cycle are the main concern of Lemma 1. In particular, the distribution of *busy* and *idle* resources inside the cycle is discussed and the *feasible* transition leading to the deadlock is characterized. Lemma 2 establishes that the cycle deadlocked by a given transition is unique. Finally, Lemma 3 uses the previous results to derive some important facts concerning the configuration of the *busy/idle* resources that is a necessary characteristic of a SLD.

Lemma 1: Let the cycle γ_n of D_W be not in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q})$. Suppose there exists a *feasible* transition e_{im} in the state $\boldsymbol{q}$ such that γ_n is in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$. Then, with reference to state $\boldsymbol{q}, \gamma_n$ satisfies the following properties:

- L1a) nodes from N_n represent $[C(\gamma_n) - 1]$ *busy* resources and one *idle* resource;
- L1b) r_m is the *idle* resource of N_n (with $r_m \neq r_{\mathrm{R}}$) and $r_i \notin N_n$;
- L1c) edges from $E_n \cap E_{\mathrm{Tr}}(\boldsymbol{q})$ form an open path linking all the vertices of N_n and ending in r_m with a *feasible* transition. Thus such a path contains $[C(\gamma_n) - 1]$ edges (i.e., $O_{\boldsymbol{q}}(\gamma_n) = [C(\gamma_n) - 1]$) which represent $[C(\gamma_n) - 2]$ *blocked* transitions followed by one *feasible* transition.

Proof: L1a) is true.

By contradiction, assume L1a) false and consider two cases.

1) All the resources from N_n are *busy* in the state $\boldsymbol{q}$ so that the outdegree of each vertex of N_n in $D_{\mathrm{Tr}}(\boldsymbol{q})$ is equal to one. Now, if $O_{\boldsymbol{q}}(\gamma_n) = C(\gamma_n)$, Corollary 1 implies that γ_n is in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q})$, which is a contradiction. So consider $O_{\boldsymbol{q}}(\gamma_n) < C(\gamma_n)$. Since r_m is *idle* and all the nodes of N_n are *busy*, it follows: $r_m \notin N_n$. Thus the digraph transformation of $E_{\mathrm{Tr}}(\boldsymbol{q})$ leading to $E_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$ does not cancel or add any edge to $E_n \cap E_{\mathrm{Tr}}(\boldsymbol{q})$. Consequently, executing e_{im} keeps the Overlap Degree of γ_n unchanged, so that γ_n is not in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$, in contradiction with the assumption.

2) At least two resources from N_n are *idle* in the state $\boldsymbol{q}$. Since two *idle* resources can not become *busy* as result of a single transition, there exists no *feasible* transition, leading γ_n in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$: a contradiction.

L1b) is true.

By L1a), the *feasible* transition e_{im} increases the number of *busy* resources of N_n from $[C(\gamma_n) - 1]$ to $C(\gamma_n)$. Such an increment occurs only if e_{im} corresponds to a job leaving $r_i \notin N_n$ to occupy the *idle* resource $r_m \in N_n$. Finally, since the outdegree of r_{R} is zero, there is no cycle in D_W containing the fictitious resource. Hence we get: $r_m \neq r_{\mathrm{R}}$.

L1c) is true.

By assumption, e_{im} makes γ_n to be in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$. Hence we get

$$E_n \cap E_{\mathrm{Tr}}(\boldsymbol{q}, e_{im}) = E_n \tag{18}$$

where $\mathrm{Card}[E_n \cap E_{\mathrm{Tr}}(\boldsymbol{q})] < C(\gamma_n)$ and $\mathrm{Card}[E_n \cap E_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})] = C(\gamma_n)$. According to the transformation mechanism leading from $D_{\mathrm{Tr}}(\boldsymbol{q})$ to $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$, the set $E_n \cap E_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$ differs from $E_n \cap E_{\mathrm{Tr}}(\boldsymbol{q})$ in the fact that it contains one more edge which starts from r_m (say $e_{mp} \in E_n$). It turns out that edges in $E_n \cap E_{\mathrm{Tr}}(\boldsymbol{q})$ form an open path joining all the vertices from N_n, containing $[C(\gamma_n) - 1]$ edges and ending in r_m. Obviously, the last edge of such a path is a *feasible* transition. Hence the proof. •

Remark A1: From the proof of L1c) it follows that there exists a job $j \in J_{\boldsymbol{q}}$ having $r_i \notin N_n$ and $r_m, r_p \in N_n$, respectively, as first, second, and third resource in $\mathrm{RW}(j)$.

Remark A2: Assume that γ_n satisfies Lemma 1. Since the outdegree of any vertex of $D_{\mathrm{Tr}}(\boldsymbol{q})$ is one at most, if $r_k \in N_n$ and $e_{kp} \in E_{\mathrm{Tr}}(\boldsymbol{q})$, from L1a) and L1c) it follows that $r_p \in N_n$ and $e_{kp} \in E_n$. In other words, each edge of $D_{\mathrm{Tr}}(\boldsymbol{q})$ outgoing from a vertex of N_n is in E_n and, of course, ends in a node still belonging to N_n.

Now, it is natural to ask whether there exist several cycles deadlocked by the same *feasible* transition. The next result enlightens this point.

Lemma 2: Let $\boldsymbol{q}$ be not a deadlock state and let e_{im} be a *feasible* transition of $D_{\mathrm{Tr}}(\boldsymbol{q})$. If there exists a cycle γ_n of D_W in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$, then γ_n is the only cycle of D_W in such a condition.

Proof: By contradiction, assume that D_W contains a cycle $\gamma_{n_1} \neq \gamma_n$ that is in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$. According to L1b), this means that $r_m \in N_n \cap N_{n_1}$. Since the output degree of each vertex of the Transition Digraph is less or equal to one, $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$ can not contain distinct cycles with nodes in common. It follows $\gamma_{n_1} = \gamma_n$ and, hence, a contradiction. •

Now let $\boldsymbol{q}$ be a SLD state for S and J_S and R_S be the corresponding job and resource sets enjoying the properties stated in Definition 4. We say that such sets are *minimal*, if no proper subsets $R_{S_1} \subset R_S$ and $J_{S_1} \subset J_S$ exist enjoying the conditions of Definition 4.

With this notion as background, we state the following necessary conditions for the occurrence of a SLD.

Lemma 3: Let $\boldsymbol{q}$ be a SLD state for S and let J_S and R_S be the corresponding minimal sets. Then there exists a set Γ of cycles of D_W satisfying the following conditions:

L3a) for each $\gamma_n \in \Gamma$, there exists a *feasible* transition $e_{im} \in E_S$ such that γ_n is in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$;

L3b) for each transition $e_{im} \in E_S$, there exists a cycle $\gamma_n \in \Gamma$ which is in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$;

L3c) the set N_Γ of all the vertices of the cycles from Γ is included in R_S;

L3d) the set of all the *feasible* transitions of $D_{\mathrm{Tr}}(\boldsymbol{q})$ belonging to cycles from Γ is equal to E_S;

L3e) in the $\boldsymbol{q}$ state, there exists only one *idle* resource in N_Γ (say r_m). Such resource is common to all the cycles from Γ.

Proof: By Definition 4 and Theorem 1, for each $e_{im} \in E_S$ there exists a cycle of D_W in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$. So, denoting with Γ the set of such cycles, we can now prove the statements of the Lemma.

L3a) and L3b) are true.

This proof is a straightforward consequence of the construction of Γ.

L3c) is true.

Let r_k be a vertex of $\gamma_n \in \Gamma$. By L3a), there exists $e_{im} \in E_S$ such that γ_n is in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$. Two cases can occur. If $r_k \neq r_m$ then, by Lemma 1, r_k is *busy* and $\mathrm{HR}(j_0) = r_k$ where j_0 indicates the job holding it. If $r_k = r_m$ then $\mathrm{SR}(j_0) = r_k$, where j_0 is the job holding r_i. Obviously in both cases e_{im} leads to a deadlock involving j_0. Thus, $j_0 \in J_D$ and, by D4c), $j_0 \in J_S$. Moreover D4a) yields $\mathrm{HR}(j_0) \in R_S$ and $\mathrm{SR}(j_0) \in R_S$. In any case the above arguments imply $r_k \in R_S$ and complete the proof.

L3d) is true.

First we show that each *feasible* transition of $D_{\mathrm{Tr}}(\boldsymbol{q})$ belonging to cycles from Γ is in E_S. Let $\gamma_n \in \Gamma$ and $e_{im} \in E_F(\boldsymbol{q}) \cap E_n$. Since both vertices r_i and r_m are in N_n, L3c) gives $r_i, r_m \in R_S$. Moreover, since r_i is *busy* and r_m is *idle*, D4b) leads to: $r_i \in \mathrm{HR}(J_S)$ and $r_m \in [R_S - \mathrm{HR}(J_S)]$. Hence we conclude that $e_{im} \in E_S$.

Vice versa we show that each *feasible* transition from E_S belongs to cycles from Γ. Assume the contrary, i.e., that there exists a *feasible* transition $e_{im} \in E_S$ which belongs to no cycle from Γ. By Remark A2, this means that $r_i \notin N_\Gamma$. Hence, by L3c), the sets $R_{S_1} = N_\Gamma$ and $J_{S_1} = \{j \in J_{\boldsymbol{q}}: \mathrm{HR}(j) \in R_{S_1}\}$ are proper subsets of R_S and J_S. It is now easy to realize that R_{S_1} and J_{S_1} satisfy the conditions of Definition 4. Therefore R_S and J_S are not minimal, in contradiction with the assumption.

L3e) is true.

By L1a) there exists an *idle* vertex in N_Γ (say r_m). To prove the unicity of this vertex, we proceed by contradiction. In particular, we show that if there exists another *idle* resource $r_\ell \in N_\Gamma$, with $r_\ell \neq r_m$ then R_S and J_S are not minimal, in contradiction with the assumption.

Let Γ_1 be the subset of Γ, collecting all the cycles containing the vertex r_m. Moreover let $N_{\Gamma_1} \subset N_\Gamma$ indicate the proper

subset of all the vertices from D_W belonging to cycles from Γ_1. We get $r_\ell \notin N_{\Gamma_1}$ because by L1a), r_m represents the only *idle* resource from N_{Γ_1}. With this as background we put $R_{S_1} = N_{\Gamma_1}$ and $J_{S_1} = \{j \in J_{\boldsymbol{q}}\colon \mathrm{HR}(j) \in R_{S_1}\}$ and show that R_{S_1} and J_{S_1} satisfy the conditions established by Definition 4. Obviously, L3c) leads to $r_\ell \in R_S$ and, since $r_\ell \notin N_{\Gamma_1} = R_{S_1}$, we clearly get $R_{S_1} \neq R_S$.

To begin with, we first observe that, by construction, R_{S_1} and J_{S_1} satisfy both the property D4b) and the condition $\mathrm{HR}(J_{S_1}) \subset R_{S_1}$. Moreover, since by Remark A2 each edge from $D_{\mathrm{Tr}}(\boldsymbol{q})$ outgoing from a vertex of N_{Γ_1} still ends in N_{Γ_1}, it holds $\mathrm{SR}(J_{S_1}) \subset R_{S_1}$. Hence D4a) is also verified.

To complete our arguments we have to prove that R_{S_1} and J_{S_1} satisfy D4c). We first observe that $[R_{S_1} - \mathrm{HR}(J_{S_1})] = \{r_m\}$, by construction. Then let $j \in J_{S_1}$ be such that $\mathrm{SR}(j) = r_m$. Putting $r_i = \mathrm{HR}(j)$, let γ_n indicate the cycle from Γ in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$. As a result, L1b) yields $r_m \in N_n$ and hence $\gamma_n \in \Gamma_1$. Consequently, as the job j, also all the jobs holding resources from γ_n are in J_{S_1}. Thus the subset J_D of jobs in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$ satisfies the relation $J_D \subset J_{S_1}$. It follows that, for each $j \in J_{S_1}$ such that $\mathrm{SR}(j) \in [R_S - \mathrm{HR}(J_S)]$, the transition releasing $\mathrm{HR}(j)$ to hold $\mathrm{SR}(j)$ leads to a deadlock state for which $J_D \subset J_{S_1}$.

The above arguments show that there exist the proper subsets $R_{S_1} \subset R_S$ and $J_{S_1} \subset J_S$ satisfying conditions of Definition 4. However this means that R_S and J_S are not minimal, which is the contradiction we had to show.

Finally, to complete the proof, it is sufficient to note that by L1a), each cycle from Γ has one *idle* vertex: so r_m is common to all the cycles from Γ. •

Now we are ready to prove Theorem 2.

Proof of Theorem 2: Sufficiency.

Our claim is that if there exists an ordered cycle set Γ_o satisfying Th2a) and Th2b), then $\boldsymbol{q}$ is a SLD state (see Definition 4). The proof is by construction. To begin with, let R_S be the set of resources corresponding to the vertices of the cycles in Γ_o:

$$R_S = N_{\Gamma_o} \tag{19}$$

and let J_S be the set of jobs holding resources in R_S:

$$J_S = \{j \in J_{\boldsymbol{q}}\colon \mathrm{HR}(j) \in R_S\}. \tag{20}$$

The definition (20) implies $\mathrm{HR}(J_S) \subset R_S$. Furthermore, by Remark A2, for any cycle $\gamma_n \in \Gamma_o$ each edge of $D_{\mathrm{Tr}}(\boldsymbol{q})$ outgoing from a vertex in N_n ends in a node still belonging to N_n. This means that $\mathrm{SR}(J_S) \subset R_S$. Moreover, (20) and Th2a) leads to $[R_S - \mathrm{HR}(J_S)] = \{r_m\}$ and $\mathrm{Card}(J_S) < \mathrm{Card}(R_S)$. Thus, all the resources in $[R_S - \mathrm{HR}(J_S)]$ are *idle* and we conclude that D4a) and D4b) hold true.

Finally, consider any job $j \in J_S$ with $\mathrm{SR}(j) \in [R_S - \mathrm{HR}(J_S)]$ (i.e. $\mathrm{SR}(j) = r_m$) and put $r_i = \mathrm{HR}(j)$. Since by (19) and (20) r_i is a vertex of a cycle from Γ_o (say γ_n), Remark A2 yields $e_{im} = e_F(\gamma_n)$. Hence, by Th2b), there exists a cycle from Γ_o in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q}, e_{im})$. Because of (19) and (20), the jobs involved in such a deadlock form a subset $J_D \subset J_S$, as D4c) requires.

Necessity.

Let Γ be the set satisfying Lemma 3. We now show that suitably ordering elements from Γ leads to a set Γ_o satisfying Th2a) and Th2b).

Let γ_n be any cycle from Γ. By L1c), γ_n possesses only one edge representing a *feasible* transition in $D_{\mathrm{Tr}}(\boldsymbol{q})$ (say $e_F(\gamma_n)$). In addition, by L3d) it holds $e_F(\gamma_n) \in E_S$ and, by L3b), there exists a cycle $\gamma_h \in \Gamma$ in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q}, e_F(\gamma_n))$. Hence γ_n is related to γ_h in the fact that "the only *feasible* transition of γ_n deadlocks γ_h." To exhibit this relation between cycles from Γ, it is convenient to build up a digraph $D^2(\boldsymbol{q})$, by associating a vertex to each cycle and an edge from vertex γ_n to vertex γ_h iff the only feasible transition of γ_n deadlocks γ_h. This new digraph enjoys the following properties:

1) the outdegree of each vertex equals one. Namely, by Lemmata 1 and 2, each cycle from Γ has only one *feasible* transition in the state $\boldsymbol{q}$, which deadlocks only one cycle from Γ;
2) by L3a) and L3d) the indegree of each vertex is greater than zero.

Clearly the above properties imply that the indegree of each vertex is one and that $D^2(\boldsymbol{q})$ is the union of disjoint cycles. We now show that, since R_S and J_S are minimal, $D^2(\boldsymbol{q})$ is nothing but a unique cycle containing all the nodes associated with the elements from the set Γ.

This proof is by contradiction. So suppose that $D^2(\boldsymbol{q})$ is the union of two or more disjoint cycles. Let $\Gamma_1 \subset \Gamma$ be the node set of one of such cycles and $N_{\Gamma_1} \subset N_\Gamma$ be the corresponding set of nodes of D_W belonging to cycles from Γ_1. Putting $R_{S_1} = N_{\Gamma_1}$ and $J_{S_1} = \{j \in J_{\boldsymbol{q}}\colon \mathrm{HR}(j) \in R_{S_1}\}$ and using the same arguments that prove L3e), we can show that R_{S_1} and J_{S_1} satisfy conditions of Definition 4. Hence R_S and J_S are not minimal and the contradiction is established. To sum up, $D^2(\boldsymbol{q})$ is a unique cycle containing all the nodes associated with elements from Γ.

Now, putting $H = \mathrm{Card}(\Gamma)$, choose a node of $D^2(\boldsymbol{q})$, say γ_1. The cycle of $D^2(\boldsymbol{q})$ establishes an ordering of the cycle set Γ: $\Gamma_o = \{\gamma_1, \gamma_2, \cdots, \gamma_H\}$.

At this point we show that Γ_o satisfies Th2a) and Th2b).

Th2a) is true.

Statement L3e) establishes that N_{Γ_o} contains only one *idle* resource (say r_m) which is shared by all the cycles from Γ_o. Now, given any two distinct cycles γ_n and $\gamma_h \in \Gamma$, we show that r_m is the only resource shared by γ_n and γ_h. By contradiction, suppose there exists $r_\ell \in N_n \cap N_h$, with $r_\ell \neq r_m$. Clearly r_ℓ is *busy*. Therefore, by Remark A2, $e_F(\gamma_n) = e_F(\gamma_h)$. Now let $\gamma_\nu \in \Gamma_o$ be the cycle in deadlock condition in $D_{\mathrm{Tr}}(\boldsymbol{q}, e_F(\gamma_n))$. The digraph $D^2(\boldsymbol{q})$ contains two edges both ending in γ_ν and outgoing from γ_n and γ_h, respectively. This is a contradiction because the indegree of vertex γ_ν must be one.

Th2b) is true.

The proof follows immediately from the definition of $D^2(\boldsymbol{q})$. •

REFERENCES

[1] Z. A. Banaszak and B. H. Krogh, "Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows," *IEEE Trans. Robot. Automat.*, vol. 6, pp. 724–734, Dec. 1990.

[2] E. G. Coffman, M. J. Elphick, and A. Shoshani, "System deadlocks," *Comput. Surveys*, vol. 3, no. 2, pp. 67–78, June 1971.
[3] J. Ezpeleta, J. M. Colom, and J. Martìnez, "A Petri net based deadlock prevention policy for flexible manufacturing systems," *IEEE Trans. Robot. Automat.*, vol. 11, pp. 173–184, Apr. 1995.
[4] M. P. Fanti, B. Maione, G. Piscitelli, and B. Turchiano, "System approach to a generic software specification for Flexible Manufacturing System job flow management," *Int. J. Syst. Sci.*, vol. 23, no. 11, pp. 1889–1902, 1992.
[5] ———, "System approach to design generic software for FMS real-time control of flexible manufacturing systems," *IEEE Trans. Syst., Man, Cybern. A*, vol. 26, pp. 190–202, Mar. 1996.
[6] F. Harary, *Graph Theory*. Reading, MA: Addison-Wesley, 1971.
[7] A. N. Habermann, "Prevention of system deadlocks," *Commun. ACM*, vol. 12, no. 7, pp. 373–378, 1969.
[8] F. S. Hsieh and S. C. Chang, "Dispatching-driven deadlock avoidance controller synthesis for flexible manufacturing systems," *IEEE Trans. Robot. Automat.* vol. 10, pp. 196–209, Apr. 1994.
[9] R. C. Holt, "Some deadlock properties of computer systems," *Comput. Surveys*, vol. 4, no. 3, pp. 179–196, Sept. 1972.
[10] T. K. Kumaran, W. Chang, H. Cho, and R. A. Wysk, "A structured approach to deadlock detection, avoidance and resolution in flexible manufacturing systems," *Int. J. Prod. Res.*, vol. 32, no. 10, pp. 2361–2379, 1994.
[11] S. S. Isloor and T. A. Marsland, "The deadlock problem: An overview," *Comput.*, pp. 58–78, Sept. 1980.
[12] C. D. Pedgen, *Introduction to SIMAN*. State College, PA: Systems Modeling Corp. 1983.
[13] E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms: Theory and Practice.* Englewood Cliffs, NJ: Prentice-Hall, 1977.
[14] A Silberschatz, J. Peterson, and P. Galvin, *Operating System Concepts*. Reading, MA: Addison-Wesley, 1992.
[15] M. Singhal, "Deadlock detection in distributed systems," *Comput.*, pp. 37–48, Nov. 1989.
[16] R. A. Wysk, N. S. Yang, and S. Joshi, "Detection of deadlocks in flexible manufacturing cells," *IEEE Trans. Robot. Automat.*, vol. 7, pp. 853–859, Dec. 1991.
[17] N. Viswanadham and Y. Narahari, *Performance Modeling of Automated Manufacturing Systems*. Englewood Cliffs, NJ: Prentice Hall, 1992.
[18] N. Viswanadham, Y. Narahari, and T. L. Johnson, "Deadlock prevention and deadlock avoidance in flexible manufacturing systems using Petri net models," *IEEE Trans. Robot. Automat.*, vol. 6, pp. 713–723, Dec. 1990.
[19] B. P. Zeigler, *Multifacetted Modeling and Discrete Event Simulation.* London, U.K.: Academic, 1984.

Maria Pia Fanti (M'94) was born in Siena, Italy, on February 21, 1957. She received the degree in electronic engineering from University of Pisa, Italy, in 1983 .

She is currently an Assistant Professor with the Department of Electrical and Electronic Engineering, Polytechnic of Bari, Italy. Her research focuses on structural properties of linear systems and on FMS control and modeling.

Bruno Maione (M'84) was born in Naples, Italy, on April 30, 1940. He received the degree in electrical engineering with honors from the University of Naples in 1964.

He is currently a Full Professor of automatic control with the Department of Electrical and Electronic Engineering Polytechnic of Bari, Italy. He held the position of Faculty Dean at the Polytechnic of Bari from 1986 to 1992. In 1983 and 1985 he was a visiting professor with the University of Florida, Gainesville. His primary areas of research and teaching are intelligent control, discrete event dynamical system modeling, systems and control theory.

Saverio Mascolo (M'95) was born in Bari, Italy, on March 1, 1966. He received the degree in electrical engineering with honors in 1991 and the Ph.D. degree in 1995 both from Polytechnic of Bari.

In 1995 he was a visiting scholar with the Department of Computer Science, University of California, Los Angeles. Since 1996 he has been a Researcher with the Department of Electrical and Electronic Engineering of Polytechnic of Bari. His main research interests include manufacturing systems and control of high speed communication networks.

Biagio Turchiano (M'94) was born in Bitetto (Bari), Italy, on July 25, 1953. He received the degree in electrical engineering with honors from University of Bari in 1979.

In 1984, he joined the Department of Electrical and Electronic Engineering, Polytechnic of Bari, as an assistant researcher. Currently, he is an Associate Professor of automatic control at the Polytechnic of Bari. His research and teaching interests are in the areas of production automation, systems and control theory, modeling and control of discrete event systems.