A Banker's Solution for Deadlock Avoidance in FMS With Flexible Routing and Multiresource States

J. Ezpeleta, F. Tricas, F. García-Vallés, and J. M. Colom

***Abstract*—Banker's-like approaches to deadlock avoidance are based on a decision procedure to grant active processes resources using information about the maximum needs of resources that a process can request in order to ensure termination. This paper presents an extension of the classical Banker's algorithm to a class of flexible manufacturing systems modeled by means of Petri nets. These systems have two interesting characteristics from the application point of view. First, flexible routing of parts is allowed, and second, a multiset of resources is allowed to be used at each processing step. The decision procedure introduced is polynomial in the Petri net model size.**

***Index Terms*—Banker's algorithm, concurrent systems, deadlock avoidance, flexible manufacturing systems.**

I. Introduction

Petri nets are a formal model widely used in flexible manufacturing systems (FMS). In the case of nonassembly systems, the Petri net model is based on a set of state machines, each one modeling the processing of a type of part (process plans). A process plan represents the different sequences a part of a given type can follow in order to be processed in the system. These state machines are not independent. In fact, each state needs a multiset of resources in order to carry out the operations modeled by a state. In the models considered here, each state change is related to resource-allocation operations. It corresponds to an operation in which a set of resources is freed, and a set of resources is engaged. Due to their nature, these systems are called resource allocation systems (RAS). We are considering *serially reusable* resources, in classical terminology. This means that the number of resource units is constant, each resource unit is either available or allocated to one and only one process, and a process may release a resource only if it has previously acquired it.

Different classes of RAS have been considered in the literature. The differences between them mainly concern the constraints imposed on *the use of resources* at each processing step of a part (the number and types of resources which can be used by the part) as well as the constraints imposed on the *structure of the set of processing sequences*.

This paper focuses on the deadlock problem. A deadlock means that the processing of some parts, once started, cannot finish because each of these parts requests for its advancement some resource(s) currently held by some other part(s) in this set. In order to solve this problem, a deadlock avoidance approach will be adopted in this paper. Alternative strategies to cope with the deadlock problem are deadlock prevention [1], [18] or deadlock detection and recovery [19], [2].

In a deadlock avoidance approach, the controller must ensure that the granting of resources to any process will lead to a resulting state which is "safe," i.e., a state from which all the parts being processed can terminate. In this context, the decision procedure of the Banker's algorithm [4], [8] needs to know, for each active process, its maximum need of resources throughout its life. This information is static and is used together with the dynamic information about the resources assigned to each process and the set of available resources in order to determine if an ordering for the sequential termination of the active processes exists. Sequential termination here means that an ordering in the set of active processes exists, such that the first process can terminate using its granted resources plus the free ones, the second process can terminate using the resources it holds plus the ones freed upon termination of the first process, and so on.

In this paper, the concept of (global) maximum needs for a whole process (as defined in [4] and [8]) is transformed into the concept of needs of resources for a process to terminate from the current state. This idea has already been used in previous approaches to deadlock avoidance for more restricted classes of systems than the ones we are dealing with in this paper. In [12] a control policy is obtained for the single-unit RAS (SU-RAS) (only a resource is allowed to be used at each processing step). The authors apply an adapted version of the Banker's algorithm, obtaining an efficient solution. The improvement is not only based on the knowledge of the process structure, but also on the concept of "partially ordered set of active processes." It is not necessary to find an ordered termination for all the active processes, but just for some subsets of them. For the same class of RAS, [9] evaluates, from a performance point of view, a set of different methods for manufacturing systems with automated guided vehicle (AGV) systems, one of which was the classical Banker's approach. References [11] and [14] apply an adaptation of the Banker's method, obtaining polynomial solutions for an extension of the SU-RAS class, where each processing step can be executed in any resource from a given set. Reference [15] removes some of the constraints usually imposed on the process structure, developing a Banker's solution for AGV systems where controlled recirculation is allowed in the routing of guided vehicles.

Outside the scope of manufacturing systems, [10] extends the Banker's approach to a class of systems where a multiset of resources can be used at each processing step and flexible routing is allowed. However, in order to obtain a polynomial solution, the process structure is constrained so that the set of states a process can be in has a tree structure.

The paper introduces a class of Petri nets, named S^*PR, able to deal with sequential RAS with routing flexibility and the use of multiple copies of different resources per state. We also adapt and extend previous versions of the Banker's algorithm for deadlock avoidance in FMS [11], [12], [14] to the modeling framework provided by S^*PR nets.

The proposed deadlock avoidance algorithm is applicable for those sequential RAS where transitions related to the granting of resources are controllable, and for which the set of states a part can stay during its processing in the system is known. In fact, such set of states (and the corresponding state changes) can be modeled by means of a state machine, so that any cycle containing the initial state is a correct production sequence for the given type of part. Moreover, all the paths joining a given state and the initial state are "equivalent," in the sense that any of them is a valid alternative to terminate the processing of any part in the considered state.

This paper is organized as follows. Section II introduces the class of systems and models we are considering. Section III introduces the control method we are proposing. Finally, some conclusions are presented. We assume that the reader knows about Petri nets. In any case, [13] is a very good introduction to the main concepts related to these models.

Manuscript received June 30, 2000; revised February 1, 2001, July 23, 2001. This paper was recommended for publication by Associate Editor S. Reveliotis and Editor N. Viswanadham upon evaluation of the reviewers' comments. This work was supported by the Spanish research projects TIC98-0671 (Comisión Interministerial de Ciencia y Tecnología) and TIC2001-1819 (Comisión Interministerial de Ciencia y Tecnología and Fondo Europeo de Desarollo Regional). This paper was presented in part at the IEEE International Conference on Robotics and Automation, San Francisco, CA, April 24–28, 2000.

The authors are with the Departamento de Informática e Ingeniería de Sistemas, Centro Politécnico Superior, Universidad de Zaragoza, 50015 Zaragoza, Spain (e-mail: ezpeleta@posta.unizar.es).

Digital Object Identifier 10.1109/TRA.2002.801048

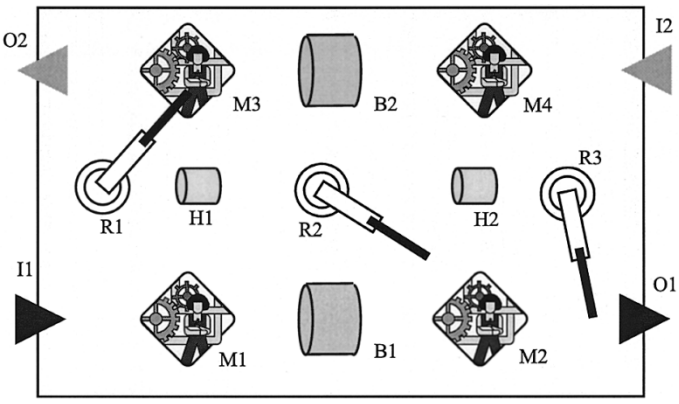

Fig. 1. Layout of a manufacturing cell.

II. Class of S^*PR Nets

Let us introduce by means of an example the class of systems we are going to deal with in this paper. Fig. 1 sketches a production cell where two types of parts, $WP1$ and $WP2$, have to be processed.

The cell is composed of four machines, $M1,\ M2,\ M3,\ M4$, whose processing capacity is of two parts each at a time. The cell also contains two tool stores, $H1$ and $H2$. The first one contains two classes of tools, $h1$ and $h3$, which have to be shared by $M1$ and $M3$. There are two copies of each one of such tools. Machine $M3$ uses one copy of each tool for the processing of each part, while machine $M1$ uses only one copy of $h1$. $H2$ contains two copies of $h2$ tool and two copies of $h4$ tool. Machines $M2$ and $M4$ use one $h2$ tool and one $h4$ tool for the processing of each part.

In order to move parts between the components of the cell, there are three robots, $R1,\ R2,\ R3$. Robot $R1$ loads machines $M1$ and $M3$ from $I1$, and unloads machine $M3$ toward point $O2$. Robot $R2$ loads and unloads the four machines, and can also interact with the intermediate buffers $B1$ and $B2$. These buffers are used for the intermediate storage of parts of types $WP1$ and $WP2$, respectively, whose processing has not finished yet. Each one of these buffers has capacity for simultaneously storing a maximum of four parts. Finally, the cell also contains a robot $R3$, which can load parts into machine $M4$ from $I2$ and unload parts from $M2$ to $O1$.

Parts of type $WP1$ are taken from a conveyor at point $I1$, processed in machine $M1$ or $M3$, then in machine $M2$ and finally unloaded on a conveyor at point $O1$. Parts of type $WP2$ are first loaded into the system from a conveyor at point $I2$, then processed in machine $M4$ and machine $M3$, and finally unloaded to another conveyor at point $O2$.

Let us now present the formal definition of the class of Petri nets we are introducing to model our systems.

Definition 1: Let $I_{\mathcal{N}} = \{1, 2, \ldots, m\}$ be a finite set of indexes. An S^*PR net is a connected generalized self-loop-free Petri net $\mathcal{N} = \langle P, T, C\rangle$ where

1) $P = P_0 \cup P_S \cup P_R$ is a partition such that: a) $P_S = \bigcup_{i\in I_{\mathcal{N}}} P_{S_i}$, $P_{S_i} \neq \emptyset$, and $P_{S_i} \cap P_{S_j} = \emptyset$, for all $i \neq j$; b) $P_0 = \bigcup_{i\in I_{\mathcal{N}}} \{p_{0_i}\}$; and c) $P_R = \{r_1, r_2, \ldots, r_n\}$, $n > 0$.

2) $T = \bigcup_{i\in I_{\mathcal{N}}} T_i$, $T_i \neq \emptyset$, $T_i \cap T_j = \emptyset$, for all $i \neq j$.

3) For all $i \in I_{\mathcal{N}}$, the subnet $\mathcal{N}_i$ generated by $P_{S_i} \cup \{p_{0_i}\} \cup T_i$ is a strongly connected state machine.

4) For each $r \in P_R$, there exists a unique minimal P-Semiflow, $Y_r \in \mathbb{N}^{\|P\|}$, such that $\{r\} = \|Y_r\| \cap P_R$, $Y_r(r) = 1$, $P_0 \cap \|Y_r\| = \emptyset$, and $P_S \cap \|Y_r\| \neq \emptyset$.

5) $P_S = \bigcup_{r\in P_R} (\|Y_r\| \backslash \{r\})\|$.

Places of P_S are called *process state places* (or *process places*). At a reachable marking, a token in a place $p \in P_{S_i}$ models an in-process part whose processing state is modeled by means of place p. Each place p_{0_i} is called *idle state place* (or *idle place*), and represents the state in which the corresponding processes (or parts) are idle. Each strongly connected state machine in *Definition 1* 3) represents the states that a part, initially in p_{0_i}, can reach during its processing. Places of P_R, called *resource places*, model how resources can be used by the active processes.

Let us present a few properties about some components of the Petri net model structure. *Definition 1* imposes the existence of a minimal P-Semiflow Y_r, such that $Y_r(r) = 1$. For a given $p \in P_S$, $Y_r(p) = k\ (\geq 0)$ means that k copies of resource r are used by each process (token) in the state modeled by means of place p. Moreover, the invariant imposed by $Y_r\ (\forall M \in \mathcal{R}(\mathcal{N}, M_0), Y_r \cdot M = Y_r \cdot M_0 = M_0(r))$ represents the fact that resources can neither be created nor destroyed. These P-Semiflows impose the resources to be serially reusable. For each $r \in P_R, \mathcal{H}_r = \|Y_r\| \setminus \{r\}$ denotes the set of state places that can use resource r.

Fig. 2 shows the S^*PR modeling the system in Fig. 1. In that figure, solid lines represent the flow of parts. Places $P1\_0$ and $P2\_0$ are the idle state places, places whose name starts with $P1$ and $P2$ correspond to state places, while the rest are the resource places. Dotted lines and connected places model the engaging/releasing of system resources. In order to have a compact notation, for every place $p \in P_S$, $Y_R(p)$ is the multiset of resources used by a process at state p: $\forall r \in P_R$, $Y_R(p)(r) = Y_r(p)$. In our example, $Y_R(P2M3) = 1 \cdot M3 + 1 \cdot h1 + 1 \cdot h2$.

Only initial markings representing no activity in the system and allowing the processing of each part in isolation will be considered; they will be called *acceptable*. These initial markings must be as follows.

1) *They represent the system idle state*. This means that: a) the initial marking of process places must be 0 (there is no activity in the initial state); b) the initial marking of each idle place represents the maximal number of parts of the type modeled by the state machine that are allowed to be concurrently processed; and c) considering the previous points, all resources are available. Consequently, the initial markings of places in P_R correspond to the resource capacities, and therefore, are greater than 0.

2) *The system is well defined*. This means that each possible processing sequence for any given part can be executed in isolation. Point 3) in *Definition 2* ensures this property.

Definition 2: Let $\langle \mathcal{N}, M_0\rangle$ be a S^*PR. An initial marking M_0 is acceptable for $\mathcal{N}$ if and only if

1) $\forall i \in I_{\mathcal{N}}, M_0(p_{0_i}) > 0$.

2) $\forall p \in P_S, M_0(p) = 0$.

3) $\forall r \in P_R, M_0(r) \geq \max_{p\in\mathcal{H}_r} Y_r(p)$.

At a reachable marking M, M_R denotes the multiset of free resources at M. $\forall r \in P_R, M_R(r) = M(r)$. A transition $t \in T \backslash P_0^\bullet$ is enabled at M if two conditions hold. First, the input state place is marked (the transition is said to be *process enabled*), and each input resource place also enables t (t is said to be *resource enabled* at M). Let us consider M and the case of two state places $p,\ q \in P_{S_i}$, for some i, such that $p^\bullet \cap {}^\bullet q = \{t\}$ and $M(p) > 0$ (t is process enabled). When is t resource enabled at M? Using the previous notation and taking into account the class definition, this occurs if and only if $M_R + Y_R(p) \geq Y_R(q)$, which is equivalent to $M_R + Y_R(p) - Y_R(q) \geq 0$.

Let us now introduce some terms related to paths in S^*PR nets. Let $\mathcal{N} = \langle P_S \cup P_0 \cup P_R, T, C\rangle$, and let $p \in P_S$. Let us assume that $p \in P_{S_i}$. A *p-processing path* is a path $pt_0p_1t_1\cdots p_{k-1}t_kp_{0_i}$ such that $\{p_{0_i}\} = P_{0_i}$, $\{p, p_1\cdots p_{k-1}\} \subseteq P_{S_i}$, and $\{t_0, t_1\cdots t_k\} \subseteq T_i$. A *simple p-processing path* is a p-processing path which is simple. $\mathcal{T}(p)$ denotes the set of all p-processing paths, while $\mathcal{T}_{\mathcal{S}}(p)$ denotes the set of all simple p-processing paths.

As a last comment related to that class of nets we are dealing with, we would like to remark that the term S^*PR does not have any specific

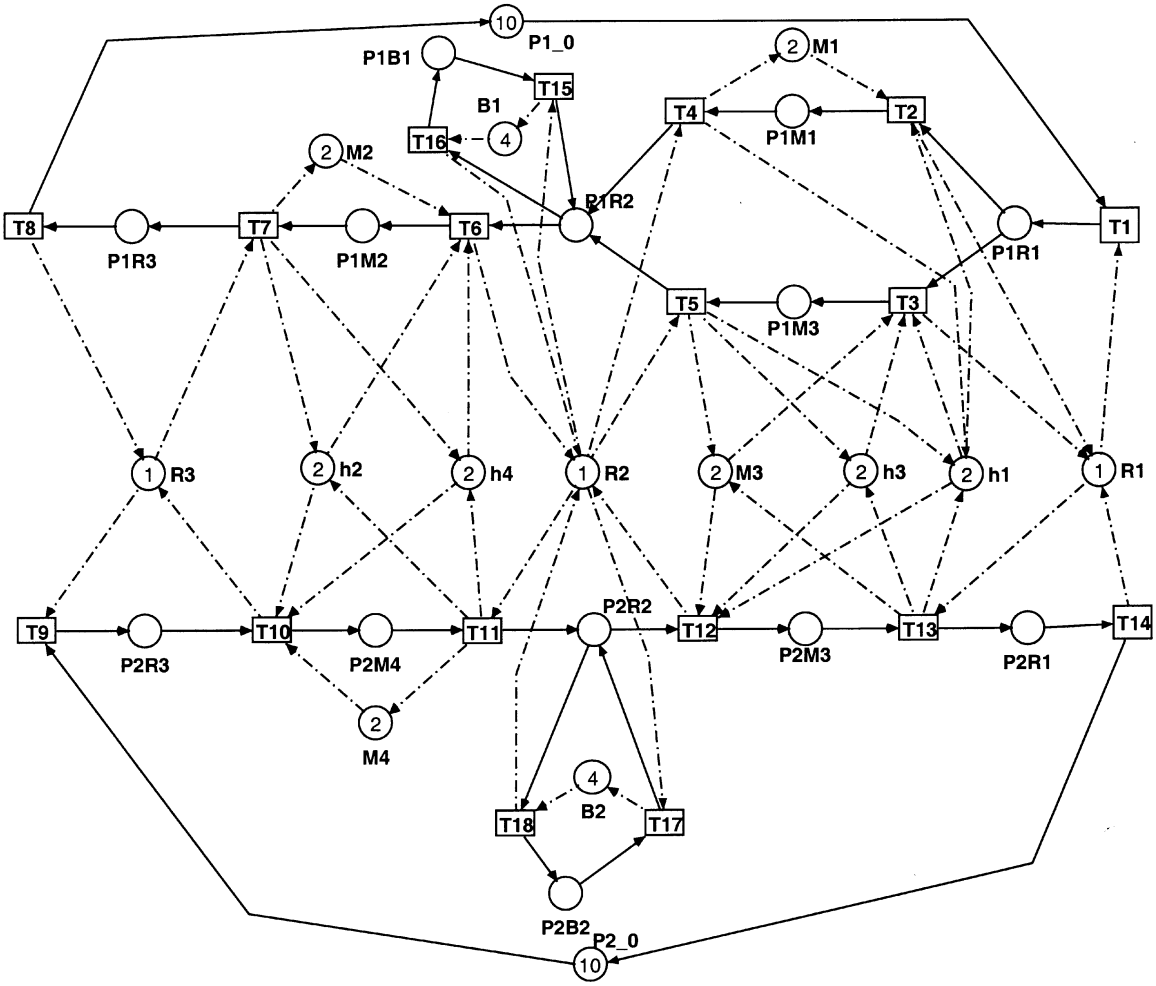

Fig. 2. S^*PR modeling the system in Fig. 1.

meaning. It has been chosen since this class of nets is a generalization of previously introduced classes named as S^3PR [5], $L-S^3PR$ [6] and S^4PR [17]. In fact, $L-S^3PR \subset S^3PR \subset S^4PR \subset S^*PR$.

III. A Banker's Algorithm for Deadlock Avoidance in S^*PR Nets

In order to explain the approach, let us first introduce some new notation relating the concepts managed in the Banker's solutions and the Petri net elements. Let M be a reachable marking. In the algorithm, each active process has to be identified. As stated above, in a S^*PR model, a process is a token in a state place. Let us now denote by $|M|$ the number of active processes at marking M: $|M| = \sum_{p \in P_S} M(p)$. Also, associated to each reachable marking, $\mathcal{A}_M = \{a_1 \cdots a_{|M|}\}$ will denote the set of tokens in state places (active processes), while π_M: $\mathcal{A}_M \longrightarrow P_S$ defines the mapping relating each active process to its corresponding state place.

In order to verify that a given system state is safe, we will proceed as follows. First, we must find an active process able to terminate using the resources it holds plus the available resources. If no such process exists, the state is considered to be nonsafe. If it exists, add the resources used by such process to the free resources, withdraw the process from the set of active process, and iterate the method. If every process can be removed from the set of active processes, the state is safe. It must be pointed out that this approach gives a sufficient condition for a state to be safe. However, it is possible that some safe states exist for which such an ordering does not exist. The immediate consequence of that is that the approach we are proposing is not, in general, maximally permissive.

Definition 3: Let $\langle \mathcal{N}, M_0 \rangle$, $\mathcal{N} = \langle P_S \cup P_0 \cup P_R, T, C \rangle$, be a marked S^*PR and $M \in \mathcal{R}(\mathcal{N}, M_0)$. An active process $a \in \mathcal{A}_M$ is said to be M_R-*terminable* if and only if there exists a path $F_a = (q_0 t_1 q_1 \cdots q_{k-1} t_k q_k) \in \mathcal{T}_S(p)$, $(q_0 = \pi_M(a),\ q_k \in P_0)$, such that $M[t_1 t_2 \cdots t_k\rangle M^a$.

Notice that M^a will be as follows: $M^a(q_0) = M(q_0) - 1$, $M^a(q_k) = M(q_k) + 1$, $M^a(p) = M(p),\ \forall p \in P_S \backslash \{q_0\}$, and $M_R^a = M_R + Y_R(q_0)$. A path, such as the one in the previous definition, is said to be M_R-*executable* for the process a. Notice that a process is M_R-terminable (at marking M) if there exists a path joining the state place where the process stays and the corresponding idle state, in such a way that the path can be followed by the process using the free resources plus those it is currently holding.

Let us now introduce the conditions under which a path is executable.

Proposition 1: Let $\langle \mathcal{N}, M_0 \rangle$, $\mathcal{N} = \langle P_S \cup P_0 \cup P_R, T, C \rangle$, be a marked S^*PR, let $M \in \mathcal{R}(\mathcal{N}, M_0)$, let $a \in \mathcal{A}_M$, and let $F_a = (q_0 t_1 q_1 \cdots q_{k-1} t_k q_k) \in \mathcal{T}_S(p)$, $(q_0 = \pi_M(a),\ q_k \in P_0)$. F_a is M_R-*executable* for a if and only if $\forall\, \beta \in \{1..k\}$, $M_R + Y_R(q_0) - Y_R(q_\beta) \geq 0$.

Proof:

$\Longrightarrow$) Let us assume that $M[t_1\rangle M_1[t_2\rangle M_2 \cdots [t_{i-1}\rangle\ M_{i-1}\ [t_i\rangle M_i \cdots [t_k\rangle M^a$.

By contradiction, let i be the minimal index, such that $M_R + Y_R(q_0) - Y_R(q_i) \ngeq 0$. Firing $t_1 t_2 \cdots t_{i-1}$, M_{i-1} is reached, and $M_{(i-1)_R} = M_R + Y_R(q_0) - Y_R(q_{i-1})$. Since

$M_{i-1}[t_i\rangle$, $M_{(i-1)_R} + Y_R(q_{i-1}) - Y_R(q_i) \geq 0$, which is equivalent to $M_R + Y_R(q_0) - Y_R(q_{i-1}) + Y_R(q_{i-1}) - Y_R(q_i) \geq 0$. This is clearly a contradiction.

$\Longleftarrow$) Let us prove by induction over the prefix of $t_1 \cdots t_k$ which is firable from M that F_a is M_R-executable.

1) Since $M_R + Y_R(q_0) - Y_R(q_1) \geq 0$, and $M(q_0) > 0$, t_1 is both, process- and resource-enabled. Therefore, t_1 can be fired.
2) Let us now assume that $M[t_1 \cdots t_{i-1}\rangle M_{i-1}$. First, since $t_{i-1} \in {}^\bullet q_i$, $M_{i-1}(q_i) > 0$ (t_i is process enabled). Moreover, $M_{(i-1)_R} = M_R + Y_R(q_0) - Y_R(q_{i-1})$. To prove that t_i is also resource enabled, we must ensure that $M_{(i-1)_R} + Y_R(q_{i-1}) - Y_R(q_i) \geq 0$. This is equivalent to $M_R + Y_R(q_0) - Y_R(q_{i-1}) + Y_R(q_{i-1}) - Y_R(q_i) \geq 0$, which is true by hypothesis.

■

The following algorithm verifies whether an active process corresponding to a reachable marking M is M_R-*terminable*. Or, in other words, if there are enough resources for the process termination when the rest of active processes do not move from their current state.

```
Function isTerminable
    (In N, a marked S*PR;
    In M ∈ RS(N, M0)\{M0};
    In a: an active process)
Return (iT:Boolean)
- -| Pre: a ∈ A_M, π_M(a) ∈ P_Si
- -| Post: iT= Is a M_R-Terminable?
Begin
    For Each p ∈ P_Si
      If M_R + Y_R(π_M(a)) − Y_R(p) ≥ 0 Then
        mark p
      End If
    End For
    iT := a simple path of marked P_Si nodes
        exists, joining π_M(a) and p_0i
    Return (iT)
End
```

Complexity: The cost of the `For` loop is $O(|P_{S_i}|\,|P_R|)$, while the cost of looking for a simple path joining two nodes in the graph of marked nodes is $O(|P_{S_i}| + |T_i|)$ [3]. Then, the cost of the function is $O(\max\{|P_{S_i}| + |T_i|, |P_R|\,|P_{S_i}|\})$. Since $|P_{S_i}| \leq |T_i|$ and $|P_R| \geq 1$, the cost is $O(|P_R|\,|T_i|)$.

The following function checks whether a reachable marking is safe in the terms we are considering: an ordering for the sequential termination of the active processes can be found. Notice that this is only a sufficient condition for the state to be safe.

```
Function isBankerAdmissible
    (In N: a marked S*PR;
    In M ∈ RS(N, M0)\{M0})
Return (iBA:Boolean)
- -| Pre:
- -| Post: iBA     = Is there a sequential
  termination of the set of
       M-active processes?
Begin
    A := A_M;  M' := M;
    iBA := TRUE;  P_M' := {p ∈ P_S | M(p) > 0}
    While iBA ∧ P_M' ≠ ∅
      look for a process a ∈ A,
      π_M'(a) = p ∈ P_M',
        s.t. isTerminable    (N, M', a)
      If such a exists Then
        P_M' := P_M'\{p};
        update M' and A assuming all the
          active processes in p teminate
      Else
        iBA := FALSE
      End If
    End While
    Return (iBA)
End
```

The second statement in the `Then` part is based on the property that if at a given state a process can terminate, all the processes in the same state place can terminate one after the other.

Complexity: The worst case corresponds to the situation in which every state place is marked at M, which implies that the number of iterations of the `While` loop is bounded by $|P_S|$. Let us consider $I_{\mathcal{N}} = \{1, 2, \ldots, m\}$, and let us denote $T_i^{\max} = \max\{|T_i| \mid i \in \{1 \cdots m\}\}$. The cost of each iteration is dominated by the sentence looking for a terminable process among those in $\mathcal{A}$, which is $O(|P_{M'}|\,|P_R|T_i^{\max})$. Since during the execution $|P_{M'}|$ is ranging from $|P_S|$ until 1, the total complexity is $O(\sum_{\alpha=1}^{|P_S|} \alpha |P_R| T_i^{\max}) = O(|P_S|^2 |P_R| T_i^{\max})$, which is polynomial in the Petri net model size.

IV. Conclusions

This paper presents two main contributions. The first one is the definition of a class of Petri nets, called S^*PR, able to model a wide set of manufacturing systems. This class is a natural extension of previous classes of nets used in the literature. These nets model RAS with routing flexibility in the processing of parts and multiset of resources at each processing step of a part. The second main contribution is the introduction of a Banker's-like algorithm for deadlock avoidance in S^*PR nets. The cost of applying the algorithm to decide whether a state is safe (and then, to ensure that all the in-process parts can be terminated) is polynomial in the model size.

As a possible improvement, looking for more permissive and efficient avoidance policies, the concept of "zone" (applied in [16] to a subclass of S^*PR), or the concept of "control point" introduced in [7], could be used as a way of improving the Banker's approaches. Instead of ensuring enough resources to terminate the processing of a given part, it is sufficient to move the part to a state in which there is no interaction with the rest of parts.

Acknowledgment

The authors are in debt to the anonymous referees and the associate editor, whose comments helped to improve previous versions of this paper.

References

[1] K. Barkaoui, A. Chaoui, and B. Zouari, "Supervisory control of discrete event systems based on structure of Petri nets," in *Proc. SMC'97 Conf.* Orlando, FL, pp. 3750–3755.
[2] H. Cho, T. K. Kumaran, and R. A. Wysk, "Graph-theoretic deadlock detection and resolution for flexible manufacturing systems," *IEEE Trans. Robot. Automat.*, vol. 11, pp. 413–421, June 1995.
[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1990.
[4] E. W. Dijkstra, *Co-Operating Sequential Processes*. New York: Academic, 1965, Programming Languages.
[5] J. Ezpeleta, J. M. Colom, and J. Martínez, "A Petri net based deadlock prevention policy for flexible manufacturing systems," *IEEE Trans. Robot. Automat.*, vol. 11, pp. 173–184, Apr. 1995.

[6] J. Ezpeleta, F. García-Vallés, and J. M. Colom, "A class of well structured Petri nets for flexible manufacturing systems," in *Advances in Petri Nets 1993*, J. Desel and M. Silva, Eds. New York: Springer-Verlag, 1998, vol. 1420, pp. 64–83.
[7] J. Ezpeleta and S. Haddad, *A Distributed Algorithm for Resource Management*. Amsterdam, The Netherlands: Elsevier, 1993.
[8] A. N. Habermann, "A new approach to avoidance of system deadlocks," *Revue Francaise d'Automatique, Informatique et Recherche Opérationnelle (R.A.I.R.O.) B-3*, pp. 19–28, Sept. 1975.
[9] C. W. Kim, J. M. A. Tanchoco, and P. H. Koo, "Deadlock prevention in manufacturing systems with AGV systems: Banker's algorithm approach," *J. Manuf. Sci. Eng.*, vol. 119, pp. 849–854, 1997.
[10] S.-D. Lang, "An extended Banker's algorithm for deadlock avoidance," *IEEE Trans. Software Eng.*, vol. 25, pp. 428–432, May/June 1999.
[11] M. Lawley, "Integrating flexible routing and algebraic deadlock avoidance policies in automated manufacturing systems," *Int. J. Prod. Res.*, vol. 38, no. 13, pp. 2931–2950, 2000.
[12] M. Lawley, S. Reveliotis, and P. Ferreira, "The application and evaluation of Banker's algorithm for deadlock-free buffer allocation in flexible manufacturing systems," *Int. J. Flexible Manuf. Syst.*, vol. 10, pp. 73–100, 1998.
[13] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, pp. 541–580, Apr. 1989.
[14] S. A. Reveliotis, "Variations on Banker's algorithm for resource allocation systems," in *Proc. 1998 Japan–USA Symp. Flexible Automation*, Otsu, Japan, pp. 1185–1192.
[15] S. A. Reveliotis, "Conflict resolution in AGV systems," *IEE Trans.*, vol. 32, no. 7, pp. 647–659, 2000.
[16] Z. Banaszak, R. Wojcik, and E. Roszkowska, "Automation of self-recovery resource allocation procedures synthesis in FMS," in *Proc. IFAC CIM Process and Manufacturing Industries*, K. Leiviska, Ed., Espoo, Finland, 1992, pp. 127–132.
[17] F. Tricas, J. M. Colom, and J. Ezpeleta, "A solution to the problem of deadlocks in concurrent systems using Petri nets and integer linear programming," in *Proc. 11th European Simulation Symp.*, G. Horton, D. Moller, and U. Rude, Eds., Erlangen, Germany, Oct. 1999, pp. 542–546.
[18] F. Tricas, F. García-Vallés, J. M. Colom, and J. Ezpeleta, "An iterative method for deadlock prevention in FMS," in *Proc. Workshop Discrete Event Systems 2000*, R. Boel and G. Stremersch, Eds., Ghent, Belgium, pp. 139–148.
[19] R. A. Wysk, N.-S. Yang, and S. Joshi, "Resolution of deadlocks in flexible manufacturing systems: Avoidance and recovery approaches," *J. Manuf. Syst.*, vol. 13, no. 2, pp. 128–138, 1994.

Divergence of Linear Acceleration-Based Redundancy Resolution Schemes

Kevin A. O'Neil

***Abstract*—The minimum torque norm control scheme for redundant manipulators has long been known to exhibit instabilities. In this paper, an analysis is made of a class of acceleration-level redundancy resolution schemes that includes the minimum torque, minimum acceleration, and other well known schemes. It is proved that divergence of joint velocity norm to infinity *in finite time* is possible, and in fact does occur for self motions of mechanisms with one degree of redundancy under almost all such schemes in the class. The schemes that do not diverge in finite time are associated with conserved quantities and include minimum acceleration norm and dynamically consistent redundancy resolution schemes. Besides the divergence, all these schemes can exhibit local instabilities in the form of abrupt increases in joint velocities and torques. Simulations are presented that illustrate the types of divergence and instability. The details of the analysis should be useful to designers seeking to modify these resolution schemes. For example, it is proved that linear dissipation of nullspace velocity cannot stabilize the algorithms.**

***Index Terms*—Dynamically consistent control, instability, minimum torque control, optimization, redundancy resolution.**

I. Introduction

A redundancy resolution scheme is a method which, given the primary task of following the desired workspace trajectory in position and orientation, selects one jointspace solution (trajectory) from the infinite number of possible solutions, often so as to accomplish some secondary task. Many redundancy resolution schemes are implemented at the acceleration level by selecting a jointspace acceleration lying in the nullspace of the derivative of the manipulator's kinematic function. (The other components of acceleration are determined by the primary task.) Examples of secondary tasks based on acceleration-level criteria appearing in the literature include minimization of the norm of the joint torque vector or joint acceleration vector [1], or minimization of the integral with respect to time of the squared norm of joint velocity or mechanism kinetic energy [2]. There has also been proposed a "dynamically consistent" redundancy resolution algorithm that takes manipulator dynamics into consideration in an effort to find the most natural motion consistent with the primary task [3], [4].

In practice, all of the redundancy resolution schemes mentioned have exhibited *instabilities*, abrupt temporary increases in joint torques and/or accelerations by one or two orders of magnitude. Instabilities of the minimum torque norm and minimum joint acceleration norm schemes were noted in an early paper [1]. The minimum acceleration norm instability was analyzed in [5] and shown to be associated with singularities of the kinematic function. A number of authors have devised *ad hoc* modifications to stabilize the minimum torque norm algorithm, but no general agreement on the precise cause of the instability is found in the literature [6]–[9].

An early study of the behavior of the minimum torque norm algorithm [9] examined self motions of mechanisms with one degree of redundancy and gave an expression for the rate of change of joint speed as the self-motion manifold of the manipulator was traced out. This showed that the nullspace component of joint acceleration responsible

Manuscript received July 10, 2001; revised December 19, 2001. This paper was recommended for publication by Associate Editor G. Oriolo and Editor A. De Luca upon evaluation of the reviewers' comments.

The author is with the Department of Mathematical and Computer Sciences, University of Tulsa, Tulsa, OK 74104 USA (e-mail: koneil@utulsa.edu).

Digital Object Identifier 10.1109/TRA.2002.801046