New Methods for Deadlock Prevention and Avoidance in Concurrent Systems *

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

Departamento de Informática e Ingeniería de Sistemas
Centro Politécnico Superior – U. Zaragoza – María de Luna 3, 50015 Zaragoza
ftricas@posta.unizar.es

Abstract

When a set of processes must share a set of common resources, deadlock problems can arise. To face them, three approaches are usually applied: deadlock detection and recovery, deadlock prevention or deadlock avoidance. In this paper we present one prevention and two avoidance deadlock algorithms for a class of models that appear in manufacturing systems. We use Petri nets as the tool for both, modelling and controlling the system. The prevention algorithm is based on an intensive use of the structure of the Petri net model. The avoidance algorithms correspond to improvements to the well-known Banker's algorithm introduced by Dijkstra in the framework of Operating Systems.

1 Introduction

In [1] we proposed a new method for deadlock prevention in Flexible Manufacturing Systems (FMS). To do that, we modelled the manufacturing system by means of a class of Petri nets that we called S^4PR. One of such nets is composed of a set of state machines (one per type of part to be processed in the system) which use a multi-set of system resources at each processing step. Focusing on the deadlock problem (we want to ensure that the processing of each raw material will finish once started), in [1] we characterised the good behaviour of the system in terms of the set of siphons of the Petri net model, and used such characterisation in order to control the system.

Solving the general deadlock problem is a very difficult task and, in consequence, some constraints are usually imposed to the class of systems a method is able to deal with. These constraints refer mainly to two elements: the possibility of flexible routing of parts, and the number and class of system resources used at each processing step. From this point of view, in this paper the general problem is solved: flexible routing is allowed and also a multi-set of resources can be used at each processing state.

One criterion that is usually accepted as a measure of the quality of a deadlock control policy is its "permissivity": more states are reachable in the controlled system, more interesting the policy is. This is so because, except for some

* This work has been partially supported by the Spanish research projects CICYT TAP98–0679, CICYT TIC98–0671 and CICYT TIC98–0587

very special types of systems [2], an "optimal" (maximally permissive controlled system) solution is not known.

In this paper we focus on the problem from two points of view: deadlock prevention (that is, given a system, modify it off–line in order to obtain a new one with adequate behaviour) and deadlock avoidance (that is, let the system evolve and check before each step if the system will remain in a 'safe state'). We present a synthesis of the methods presented in [3] and [1].

The paper is organised as follows: in section 2 we present the class of systems we are going to deal with. Next, we present the proposed prevention policy and later the two deadlock avoidance approaches. In this paper it is assumed that the reader has knowledge of the main elements related to Petri nets. Please, refer to [4, 5] for a good presentantion of this formalism.

2 The class of S^4PR nets

A net of this class is composed of a set of sequential processes that have to share (in a free way) a set of non–consumable common resources. Let us consider the manufacturing cell depicted in Figure 1. It is composed of two machines ($M1$, which is able to process two parts at a time, and $M2$, which can process just one part at a time). A part, before entering the system, needs to be loaded into a pallet; then, it can move across the system; once processed, the part leaves the cell, freeing the pallet. There are five pallets in the system; they can move following the arrows depicted in the figure. The cell also contains a buffer, named B, with capacity for two pallets. In this cell two types of parts have to be processed. Parts of Type 1 are loaded into a pallet in entry point I, processed in machine $M1$, moved to buffer B, and then loaded into machine $M2$. When finished, the part leaves the cell via point $O1$. Parts of Type 2 are loaded into a pallet in I, then processed in machine $M1$, and stored in the buffer B, to finally leave the system through $O2$. The Petri Net in Figure 2 models the previously described system. In this net there are two different kinds of places: *Places that model the state of parts* ($P1\_I$, $P1\_M1$, $P1\_B$, $P1\_M2$, $P1\_O1$, $P2\_I$, $P2\_M1$, $P2\_B$, $P2\_O2$). For example, parts of Type 2 must follow the sequence of states represented by places whit names beginning with $P2$: place $P2\_I$ models that a part of this type has been loaded into a pallet in the entry point; $P2\_M1$ represents that the part is being processed in machine $M1$, the storing of the part in the buffer is represented by the place $P2\_B$ and, finally, the part unloading from the pallet in $O2$ is represented by place $P2\_O2$. These places will be called "state places". *Places that model the state of system resources* ($M1$, $M2$, B, P). The initial marking of these places represents the capacity/availability of the resource. For example, initial marking of P is 5 because there are 5 pallets in the system. These places will be called "resource places". Notice that this model captures the dynamics of the system: the state places corresponding to a type of part need to be configured as a state machine joining the different states a part needs to visit in order to be processed. This configuration allows to model on–line decisions or alternative routings. This model also captures how

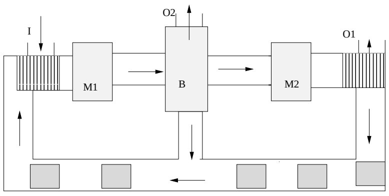

Fig. 1. The layout of a Flexible Manufacturing Cell.

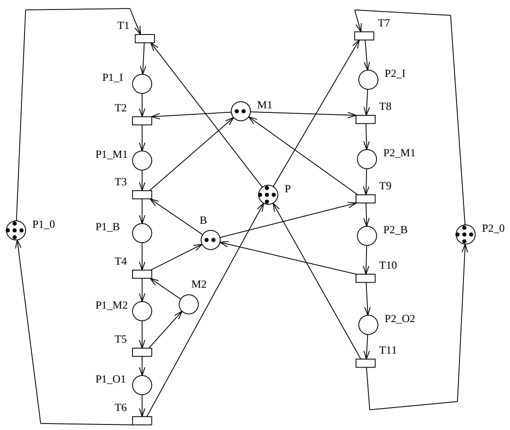

Fig. 2. Model of the Flexible Manufacturing Cell.

the resources are used: for example, in order to put a token in $P2\_M1$ (a part of Type 2 to be processed in machine $M1$), the net shows that a token has been withdrawn from P (a pallet was allocated for the transportation of the part) and another one has been withdrawn from $M1$ (the part needs one of the two available processing positions). We are interested in the modelling of *reusable* resources; that is, neither creation nor destruction of them is allowed.

Definition 1. *Let $I_{\mathcal{N}} = \{1, 2, ..., m\}$ be a finite set of indices. A S^4PR is a generalised connected self–loop free Petri net $\mathcal{N} = \langle P, T, C \rangle$ ($C = Post - Pre$ is the incidence matrix) where: (1) $P = P_S \cup P_0 \cup P_R$ 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 $i, j \in I_{\mathcal{N}}$, $i \neq j$. (b) $P_0 = \bigcup_{i \in I_{\mathcal{N}}} \{p_{0_i}\}$. (c) $P_R = \{r_1, r_2, ...r_n\}$, $n > 0$. (2) $T = \bigcup_{i \in I_{\mathcal{N}}} T_i$, $T_i \neq \emptyset$, $T_i \cap T_j = \emptyset$ for all $i, j \in I_{\mathcal{N}}$, $i \neq j$ (3) $\forall i \in I_{\mathcal{N}}$ the subnet $\mathcal{N}_i$ generated by $P_{S_i} \cup \{p_{0_i}\} \cup T_i$ is a strongly connected state machine, such that every cycle contains p_{0_i}. (4) $\forall r \in P_R$, $\exists$ a unique minimal P–Semiflow $Y_r \in \mathbb{N}^{|P|}$ such that $\{r\} = \|Y_r\| \cap P_R$, $P_0 \cap \|Y_r\| = \emptyset$, $P_S \cap \|Y_r\| \neq \emptyset$ and $Y_r(r) = 1$. (5) $P_S = \bigcup_{r \in P_R} (\|Y_r\| \setminus \{r\})$* □

P–Semiflows in point (4) state that resources are reusable. Places in P_S, P_0, and P_R are the *state places*, the *idle state places*, and the *resource places*, respectively. At a reachable marking M, a token in place P_S models an active process (a part being processed) whose state is modeled by means of place p. The marking of a place $r \in P_R$ models the available capacity of resource r. For each $r \in P_R$, $Y_r(p)$ is the capacity of resource r used by the process modeled by a token in p. Then, we can denote $Y_R(p) = \sum_{r \in P_R} Y_r(p)$ for the set of system resources. Following definitions presented in [6] we will say that a transition t is *process–*

enabled if $M({}^\bullet t \cap P_S) \neq 0$ and *resource–enabled* if and only if $\forall r \in ({}^\bullet t \cap P_R)$, $M(r) \geq Pre[r,t]$. Let us define some terms related to paths that we will use in the following. A *path* in a Petri net $\mathcal{N} = \langle P, T, C \rangle$ is a sequence of nodes $x_1, x_2, \ldots, x_n$ from $P \cup T$ such that $x_{i+1} \in x_i{}^\bullet$, for all $i \in \{1, ..., n-1\}$. A path is *simple* if and only if for all $i \neq j$, $x_i \neq x_j$. A *circuit* is a path such that $x_1 \ldots x_{n-1}$ and $x_2 \ldots x_n$ are simple and $x_1 = x_n$. A path whose places belong to $(P_S \cup P_0)$ is called a *processing path*. Let $F = p_0, t_0, ..., p_k t_k p_0$, $p_0 \in P_0$, be a processing circuit. For each $p \in F \cap (P_S \cup P_0)$, we denote $SUC(p, F) = pt_p ... p_k t_k$. We will denote $\mathcal{T}$ to the set of all processing circuits of the net. For $p \in (P_S \cup P_0)$, $\mathcal{T}(p) = \{F \in \mathcal{T} \mid p \in F\}$ is the *set of circuits related to* p. As initial state we will only consider what we call *acceptable initial markings*: (1) there is no part being processed (the system is idle, $\forall p \in P_S$, $M_0(p) = 0$) (2) there is at least one copy (or one buffering capacity) of each resource ($\forall r \in P_R$, $M_0(r) > 0$) and (3) the capacity of each resource is so that each processing step can be done ($\forall p \in P_S,\ r \in P_R,\ M_0(P_r) \geq Y_r(p)$). Moreover, the initial marking of each place in P_0 represents the maximal number of parts of the type modelled by the embedded state machine that are allowed to be concurrently processed in the system ($\forall i \in I_{\mathcal{N}}$, $M_0(p_{0_i}) > 0$). Notice that we can choose this marking in such a manner that places in P_0 become implicit, being able in this way to model open systems. From previous definitions we have that the marking of every state machine models the state of the parts of the type it represents, while the marking of each place modelling a resource represents either the number of available copies of it or its remaining capacity.

We also will use the following concepts: let F be a circuit. Let $p \in F \cap P_S$, $r \in P_R$. (1) The *potential needs of resource* r *at state* p *over the circuit* F is defined as $PNR_F(p,r) = \max_{q \in SUC(p,F)} \{Y_r(q)\}$ (2) The *potential needs of resource* r *of place* p is defined as $PNR(p,r) = \max_{F \in \mathcal{T}} \{PNR_F(p,r)\}$ (3) The *vector of potential needs of place* p is the P_R–indexed vector such that $PNR(p)(r) = PNR(p,r)$.

It will be convenient to define the *set of holders of the resource* r, $\mathcal{H}(r)$, as the set of state places that belong to the support of this P–Semiflow. It will also be useful to extend this definition to sets when needed: if R is a set of resources, $\mathcal{H}(R) = \bigcup_{r \in R} \mathcal{H}(r)$. With respect to the initial marking of the net, only what we call *acceptable initial markings* (as defined in the following) will be considered.

3 The deadlock prevention control policy

Theorem 1. *[1] Let $\langle \mathcal{N}, M_0 \rangle$ be a marked S^4PR. Let $M \in \mathcal{R}(\mathcal{N}, M_0)$ and let $t \in T$ be a dead transition for M. Then there exists a siphon[1] $D = D_R \cup D_S \neq \emptyset$ and $M' \in \mathcal{R}(\mathcal{N}, M)$ so that (1) $D \cap P_R = D_R \neq \emptyset$ (2) $D \cap P_S = D_S \neq \emptyset$ (3) Each place in D_R is preventing the firing of some process–enabled transitions, (4) $D_S \subset \mathcal{H}(D_R)$ and each place in D_S is unmarked at M'* □

[1] A subset of places $D \subseteq P$ is said to be a siphon if and only if ${}^\bullet D \subseteq D^\bullet$

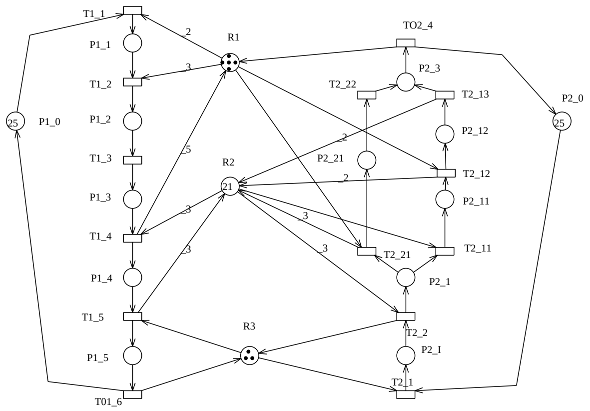

Fig. 3. A net with a siphon that can be problematic.

A direct consequence of this theorem is that if there exists a dead transition for a reachable marking, there exists a minimal siphon verifying conditions (1)–(4) of the previous theorem.

As proved in [1], the reason for a deadlock is a siphon that lost too many tokens. Since initially all the tokens of a siphon are in the resource places, the reason for deadlock is that these tokens moved to those holders of the resources that do not belong to the siphon themselves. That is, the problem is related to siphons $D = D_R \cup D_S$ and tokens that move from D_R to $Th(D) = \mathcal{H}(D_R) \setminus D_S$ (Th(D) is called the set of "thiefs" of D). Based on these ideas, the control in [1] was established as follows. For each siphon D (not being the support of a P–Semiflow) add a control place p_D in such a way that p_D and $Th(D)$ generate a new P–Semiflow ensuring that not "too many" tokens flow from D_R to $Th(D)$ (see Definition 4). The integer linear programming problem in Definition 3 gives a bound for the number of tokens that are allowed to flow to $Th(D)$. For example, in Fig. 3 there are four siphons. One of them is $D1 = \{P1\_2, P1\_3, P2\_21, P2\_12, P2\_3, R1\}$. The set $Th(D1) = P1\_1$ and the solution of the corresponding set of ILPPs show that that if we impose to $Th(D1)$ to have three or less tokens no marking will be reached with too many tokens out of the siphon. This is implemented by means of the addition of place pD1, as shown in Fig. 4. For short, any marking in the potential reachability set ($M \in \mathcal{PR}(\mathcal{N}, M_0)$, we say 'potential' because we use the state equation for markings instead constructing the actual reachability set), verifying the properties of marking M' of Theorem 1 for a siphon D will be called a D*–deadlocked marking*. We are using the $\mathcal{PR}(\mathcal{N}, M_0)$ instead of $\mathcal{R}(\mathcal{N}, M_0)$ because, as it will be introduced later, our method is based on the use of the state equation. As proved in [1] the siphons of Theorem 1 cannot be the support of a P–Semiflow; so, we will concentrate on the set of *dangerous siphons*: $\mathcal{D} = \{D$ minimal siphon $\mid D$ is not the support of any minimal P–Semiflow $\}$. In order

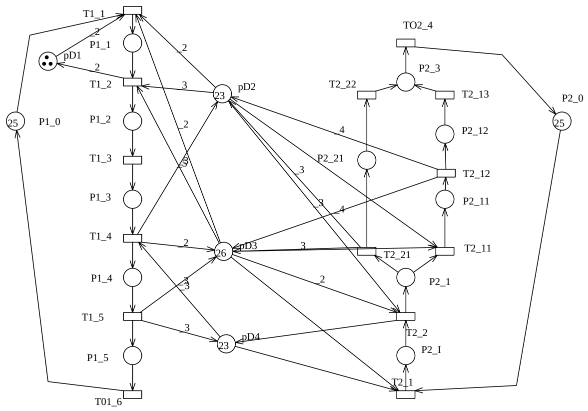

Fig. 4. Control Places added to the net after the first iteration.

to establish the control policy we first define the set of restrictions asociated to a "dangerous" siphon.

As stated in [1], the added places can be considered as 'virtual' resources of the system. Then, an iterative application of the control policy is needed. Finally, notice that we only add a control place if in the potential reachability set there exists a solution to, at least, one integer linear programming problem as considered above. In consequence, the addition of a control place always reduces the size of the potential state space (in, at least, one marking). Then, the procedure will be executed a finite number of times

Definition 2. *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^4PR. Let D be a dangerous siphon. For each $r \in D_R$, and $t \in r^\bullet$, we define the D–restriction associated to r and t as:*

$Rest(D, r, t) :: [M({}^\bullet t \cap P_S) > 0] \wedge [M(r) < Pre[r, t]]$ [2] $\square$

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^4PR. Let D be a dangerous siphon; we define its associated set of integer linear programming problems, $ILPP_D$, as follows:*

$$
\begin{aligned}
M_D^{max} = maximize & \sum_{r \in D_R} M(r) \\
subject\ to\ & M = M_0 + C \cdot \overline{\sigma} \\
& \overline{\sigma} \geq 0, M \geq 0, M(D_S) = 0 \\
& \bigwedge_{r \in D_R} \Big(\bigvee_{t \in r^\bullet} Rest(D, r, t) \Big)
\end{aligned}
$$

[2] In fact, it should be for those $t \in r^\bullet$ such that $({}^\bullet t \cap P_S \notin D)$. Note that, from a theoretical point of view, this is equivalent since a D–deadlocked marking verifies $M(D_S) = 0$, which is incompatible with the restriction

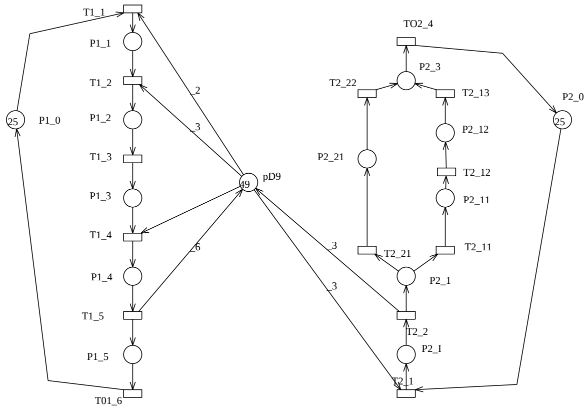

Fig. 5. Control Places added to the net after the second iteration.

At this point, it is necessary to divide the set of dangerous siphons of a given S^4PR in two sets: those dangerous siphons for which $ILPP_D$ has at least one solution, which will be called *bad* and those for which $ILPP_D$ has no solution, which will be called just dangerous. Notice that only bad siphons will generate some control places.

Definition 4. *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^4PR. Let D be a bad siphon. $\langle \mathcal{N}^{p_D}, M_0^{p_D} \rangle$, the D–*controlled *system of $\langle \mathcal{N}, M_0 \rangle$, is defined as follows: (1) $\mathcal{N}^{p_D} = \langle P_S \cup P_0 \cup P_R \cup \{p_D\}, T, C^{p_D} \rangle$ (2) $\forall p \in P_S \cup P_0 \cup P_R$, $C^{p_D}[p,T] = C[p,T]$, $M_0^{p_D}(p) = M_0(p)$ (3) $C^{p_D}[p_D,T] = -\sum_{p \in Th(D)} Y_{D_R}(p) \cdot C[p,T]$, (4) $M_0^{p_D}(p_D) = M_0(D) - (M_D^{max} + 1)$* □

The controlled system is obtained by means of the following algorithm:

Algorithm `controlII`S^4PR
Input: $\langle \mathcal{N}, M_0 \rangle$`, a `$S^4PR$` net`
Output: $\langle \mathcal{N}^{\mathtt{C}}, {M_0}^{\mathtt{C}} \rangle$`, the controlled `$\langle \mathcal{N}, M_0 \rangle$
Begin
  $\langle \mathcal{N}^{\mathtt{C}}, {M_0}^{\mathtt{C}} \rangle$ `:=` $\langle \mathcal{N}, M_0 \rangle$
  Repeat
    $\mathcal{D} := \{\mathtt{D} \mid \mathtt{D}$` is a dangerous siphon of `$\mathcal{N}^{\mathtt{C}}\}$
    For_Each $\mathtt{D} \in \mathcal{D}$
      `Solve the ILPP`$_{\mathtt{D}}$` presented in Def. 3 for `$(\langle \mathcal{N}^{\mathtt{C}}, {M_0}^{\mathtt{C}} \rangle, \mathtt{D})$
      If $\exists\, M_D^{max}$ **Then**
        `Add a control place p`$_{\mathtt{D}}$` to `$\langle \mathcal{N}^{\mathtt{C}}, {M_0}^{\mathtt{C}} \rangle$` as stated in Def. 4`
      End_If
    End_For
  Until `No new control place is added`

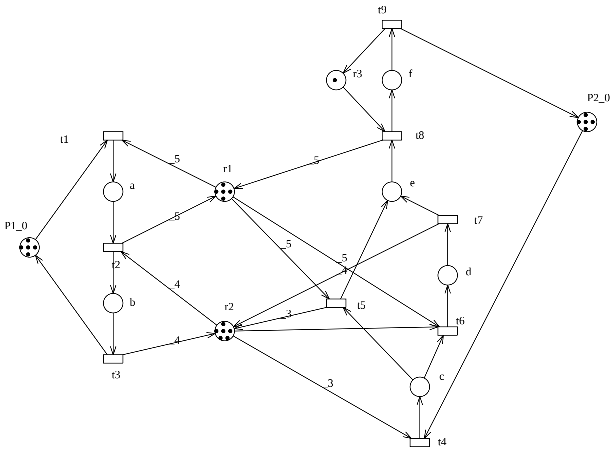

Fig. 6. An S^4PR with weighted arcs (generalized Petri Net).

End

As proved in [1] the obtained system is live.

4 A Deadlock Avoidance approach: improving the Banker's Algorithm

In the following, we will focus on the deadlock avoidance problem. Perhaps, the most known algorithm for deadlock avoidance is the Banker's algorithm. It is based on the idea that before activation each process must declare the maximum number of instances of each type of resource it may need. The algorithm requires the concept of *safe* state: a state is considered to be safe if we can find an ordering in the execution of the active processes such that the system is able to ensure that the needs of each active process could be granted in the future. In order to implement this, the following data structures are used, where n is the number of types of resources in the system and m the number of active processes: (1) **Available**: An $1 \times card(P_R)$–vector[3] $Available[k] = j$ means that there are k available copies of resource type j (2) **Max**: An $m \times card(P_R)$–matrix. $Max[i, j] = k$ means that the process i may request at most k copies of resource j. In the Banker's algorithm this matrix is static (3) **Allocation**: An $m \times card(P_R)$–matrix. $Allocation[i, j] = k$ means that the process i is using k copies of resource j (4) **Need**: For each process, the remaining needs of resources, obtained as *Need* = *Max* - *Allocation*. Let M be a reachable state. Let $|M|$ be the number of active processes at M :$| M |= \sum_{p \in P_S} M(p)$. For example, in

[3] $card(A)$ is used to represent the cardinality of the set A

Figure 6, marking $3 \cdot a + c + f + 3 \cdot r2$ can be described as a set of processes: $\{a_1, a_2, a_3, c_1, f_1\}$ where a_1, a_2, a_3 correspond to processes in the state modelled by means of place a, c_1 to a process in c and f_1 to a process in state f. In order to formulate the classical Banker's algorithm, let us identify each active process, and denote them as $\mathcal{A} = \{a_1, \ldots, a_{|M|}\}$. The Banker's algorithm considers that a given state (marking) is safe if there exists an ordering of the active processes allowing to finish all of them in the following way: the first one can finish with the resources it is holding and the rest of needs can be granted by the *Available* resources; the second one must be able to finish in a similar way to the first one but now the *Available* resources are increased by the resources allocated to the first one; and so on for the rest of active processes. Formally, it can be formulated as follows. Let $M \in \mathcal{R}(\mathcal{N}, M_0)$; the Banker's algorithm considers that this state is safe if and only if there exists an ordering function $\alpha_M : \{a_1, ..., a_{|M|}\} \longrightarrow \{1, ..., |M|\}$ such that, for each $i \in \{1, ..., |M|\}$, $Available_M + Allocation(a_i) + \sum_{\substack{j \in \{1,\ldots,|M|\} \\ \alpha_M(a_j) < \alpha_M(a_i)}} Allocation(a_j) \geq Max(a_i)$. The number of free resources depends on the considered marking. This is the reason to denote this number as $Available_M$. Our proposal is based on the observation that in the class of models we are working with we know at each step an upper bound of the future need of resources, looking forward at each processing circuit of the Petri net model, from the place where the token modeling the process is, until the idle state place. In consequence, after reaching a new safe state, we can recompute the *Max* matrix and take advantage of the knowledge about the processing sequences. This idea is based on the following fact: *for a process, we do not need to guarantee at each step that the maximum needs will be available during all of its working life, but only the maximum of the effective needs*, i.e., the number of copies of resources that can be needed from the actual state until the corresponding idle place.

Figure 7 depicts the reachability graph of the net in Figure 6. This net can reach some deadlocks (marking #9, for instance), and also some markings, as #16, which are not deadlocks but that will reach a deadlock in an inevitable way. In this graph, markings with a "–" mark are those states that are allowed once the classical Banker's algorithm is applied. These markings must be increased with those with "+" in order to get the markings allowed if the algorithm incorporates the improvement we have just introduced. In order to show the differences of the classical version and the first one presented here, let us consider state "#11": $c + e + 3 \cdot r2 + r4$ (idle places are not shown for the sake of brevity). In this state, there are two active processes: the tokens in places c and e. We use *Max1* to denote the *dynamic* version of the *Max* matrix (which corresponds to the new method): $Max1[i, j] = k$ means that process i may need until k copies of resource j, but considering only those states it can reach from the current state until completion (places of P_S in the paths from the place where the token is to the corresponding idle place). For example, $Max[e, r2] = 4$ and $Max1[e, r2] = 0$. This is so because the token would have needed four copies of $r2$ in the state modeled by e, which is not reachable anymore by the process modeled by the token in d. So, at the considered state the data are the following:

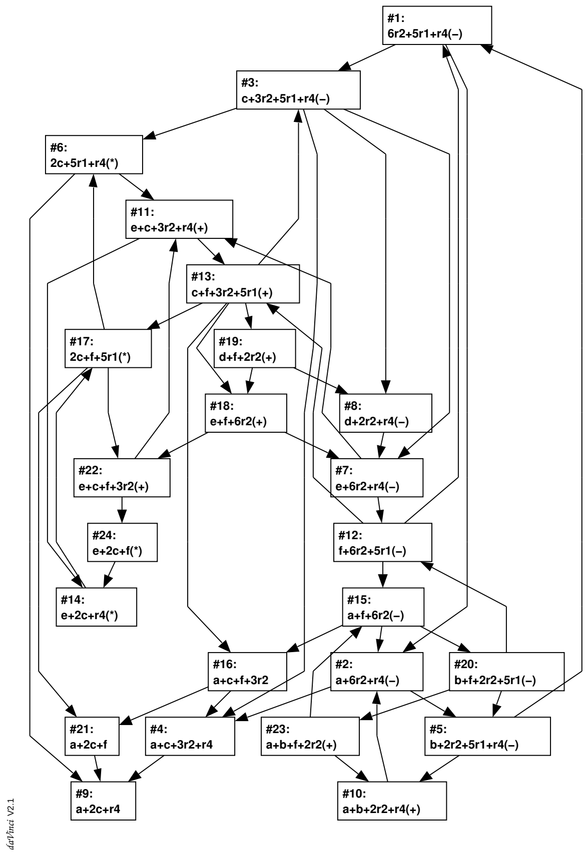

Fig. 7. Reachability graph of the net in Fig. 6.

	Max			Max1			Allocation			Available		
c	5	4	1	5	4	1	0	3	0	0	3	1
e	5	4	1	5	0	1	5	0	0			

It is easy to prove that this marking is not safe for the classical Banker's algorithm. Now, for our method, we have to find an ordered sequence. We can start this sequence with the process modeled with the token in e, since $[0,3,1]+[5,0,0] = [5,3,1] \geq [5,0,1]$ and now, we can finish the sequence with the process modeled with the token in c since $[0,3,1]+[5,0,0]+[0,3,0] = [5,6,1] \geq [5,4,1]$; so we can conclude that this state is safe for the improved method. This improvement can be considered as a generalization of the one presented in [7] (because in each processing state we can use a multi-set of resources).

The second improvement that we are proposing looks in a more detailed way at the structure of the Petri net model. In the first improvement we have defined $Max1$ in such a way that each process eventually finish following no matter which path from its actual state to the completion of the processing. But it is enough to find an ordering in such a way that a given process can finish, which is equivalent to ensure that resources can be granted in such a way that *at least* one path from the actual state to the idle one can be followed: i.e., if the maximal needs differ depending on the processing path (completion of a processing path), we do not need to guarantee the maximal needs but only the adequate needs to finish following one of the processing paths. In the reachability graph of Figure 7 the states considered safe with this approach and not with the classical and the first improvement are marked (*). Let us concentrate on state $\#14 : 2 \cdot c + e + r4$. For each process in c there are two different maximal needs, depending on which path we choose for termination; for the path corresponding to transitions $\{t4, t5, t8, t9\}$, the maximal need is $n1 = [5, 4, 1]$, for the path corresponding to transitions $\{t4, t6, t7, t9\}$ is $n2 = [5, 3, 1]$. These two possibilities are reflected in matrix $Max2$. For the token in e the maximum needs are the same passing through all the paths: $[5, 0, 1]$. This information is sketched in table $Max2$:

	Max1			Max2			Allocation			Available		
c	5	4	1	n1 or n2			0	3	0	0	0	1
c	5	4	1	n1 or n2			0	3	0			
e	5	0	1	5	0	1	5	0	0			

It is easy to prove that this marking is not safe neither for the first improvement nor for the classical algorithm. Now, let us consider that for each process in c we can choose either $n1$ or $n2$ as the maximal needs vector. Let us consider $n2$ (in this case this decision is obvious, but there can be situations in which no obvious decision exists) for both cases. We have then $[5, 0, 0] + [0, 0, 1] = [5, 0, 1] \geq [5, 0, 1]$ and then, choosing one of the other two processes: $[5, 0, 0] + [0, 0, 1] + [0, 3, 0] = [5, 0, 1] + [0, 3, 0] = [5, 3, 1] \geq [5, 3, 1]$ and, in consequence, for the remaining process: $[5, 0, 0] + [0, 0, 1] + [0, 3, 0] + [0, 3, 0] = [5, 3, 1] + [0, 3, 0] = [5, 6, 1] \geq [5, 3, 1]$ that conducts us to consider this state as safe.

4.1 First improvement

Let M be a reachable state. For convenience, the relation between the set of active processes and the corresponding reachable marking will be established by means of a mapping $AP_M : \mathcal{A}_M = \{a_1, ...a_{|M|}\} \longrightarrow P_S$ where a_i is the i-th active process. This mapping is such that, given an active process $a_i \in \mathcal{A}_M$, $AP_M(a_i)$ is the place that contains the token modeling the process a_i.

Definition 5. *Let $M \in \mathcal{R}(\mathcal{N}, M_0)$. Let $\mathcal{A}_M = \{a_1, ...a_{|M|}\}$ be the associated set of active processes. We say that this state is* safe *if and only if there exists an ordering $\alpha_M : \{a_1, ..., a_{|M|}\} \longrightarrow \{1, ..., |M|\}$, such that for each $i \in \{1, ..., |M|\}$ $Available_M + Allocation(a_i) + \sum_{\substack{j \in \{1,...,|M|\} \\ \alpha_M(a_j) < \alpha_M(a_i)}} Allocation(a_j) \geq Max1(a_i)$ where $Max1(a_i) = PNR(AP_M(a_i))$.* □

Proposition 1 ([8]). *Let $M \in \mathcal{R}(\mathcal{N}, M_0)$ be a safe state according to Definition 5. Then, M_0 is reachable from M.* □

As a corollary of this proposition we can state that an S^4PR net controlled according to Definition 5 is a live system.

Implementation hints We are going to present here some implementation details that are not essential for the correct understanding of the policy, but will help us in a better performance when implementing it. They are based on the idea that a token in a state place represents a process and that the relevant information for the process is provided by the place. First of all note that the $Max1$ matrix does not need to be re–computed for each state: we can compute the values of $Max1$ at the beginning for each place in P_S and each resource in P_R. When we have a process, we just need to check for the adequate values, using the information about in which place the token is. Then, $Max1$ is a $P_S \times P_R$ matrix, where the row $Max1(p)$ represents the maximal need of resources for any process in p in order to finish. So, for an active process $a_i \in \mathcal{A}$ in a reachable marking M, its maximal need corresponds to $Max1(AP_M(a_i))$. The second improvement is related to the fact that we can see as 'equivalent' all the processes that are represented by tokens in the same place: once you can guarantee that one of them can finish, obviously, all of them can finish one after the other, without need for more checking. Then, given a set of active processes only the checking for essentially 'different' ones is needed. This means that in the ordering we are looking for, we will have to "order" as much items as marked state places. Let us study the computational cost. The following algorithm computes the $Max1$ matrix in an efficient way.

Algorithm `Max1`
Input: $\langle \mathcal{N}, M_0 \rangle$, a S^4PR net
Output: the matrix Max1
Begin
  For_Each $\mathtt{q} \in {}^{\bullet\bullet}\mathtt{P}_0 \cap \mathtt{P_S}$
    $\mathtt{Max1(q)} := \mathtt{Y_R(q)}$
  End_For
  $\langle \mathtt{DONE}, \mathtt{NOT\_DONE} \rangle := \langle {}^{\bullet\bullet}\mathtt{P}_0 \cap \mathtt{P_S}, \mathtt{P_S} \setminus {}^{\bullet\bullet}\mathtt{P}_0 \rangle$
  While $\mathtt{NOT\_DONE} \neq \emptyset$
    Let $\mathtt{q} \in \mathtt{NOT\_DONE}$ s. t. $\mathtt{q}^{\bullet\bullet} \cap \mathtt{P_S} \subseteq \mathtt{DONE}$
    $\mathtt{Max1(q)} := \mathtt{max}\{\mathtt{Y_R(q)}, \{\mathtt{Max1(p)} \mid \mathtt{p} \in \mathtt{q}^{\bullet\bullet} \cap \mathtt{P_S}\}^4$
    $\langle \mathtt{DONE}, \mathtt{NOT\_DONE} \rangle :=$
      $\langle \mathtt{DONE} \cup \{\mathtt{q}\}, \mathtt{NOT\_DONE} \setminus \{\mathtt{q}\} \rangle$
  End_While
End

In this algorithm, the maximun of two "vectors" is a vector composed of the maximum of each component. Based on this we are going to evaluate the cost of the new approach:

Computation of the Max1 matrix: we need to check each place in the net in the adequate order, provided by the structure of the net. This need to be computed once and can be done in $O(card(P_S)^2 \times card(P_R))$.

Decision about the safety of a state: Let M be a reachable state. As proved in [9], the cost of deciding if the state M is safe is $O(m \times log(m) \times$

$card(P_R))$, being m the number of active processes. In the presented approach, the cost will be as much as the one of the traditional Banker's algorithm. However, taking into consideration the implementation hints presented above, in those cases in which more than one process are in the same state (tokens in the same place), we can have the answer with less computational effort: $O(K_M \times log(K_M) \times card(P_R))$, being K_M the number of state places marked at M. Notice that this number cannot be greater than the number of active processes.

4.2 Second improvement

In the second improvement we will take advantage of the idea that a process, in order to finish, does not need to be able to follow each possible path till the idle state, but only one of them. In this sense we will use a mapping β_M that associates to each active process a processing path (from the corresponding state place to the idle place).

Definition 6. *Let $M \in \mathcal{R}(\mathcal{N}, M_0)$. Let $\mathcal{A}_M = \{a_1, ...a_{|M|}\}$ be the associated active processes. We say that this state is* safe *if and only if there exits a bijection $\alpha_M : \{a_1, ..., a_{|M|}\} \longrightarrow \{1, ..., |M|\}$, and a mapping, $\beta_M : \{a_1, ..., a_{|M|}\} \longrightarrow \mathcal{T}$, such that, for each $i \in \{1, ..., m\}$*

$$Available_M + Allocation(a_i) + \sum_{\substack{j \in \{1,\dots,m\} \\ \alpha_M(a_j) < \alpha_M(a_i)}} Allocation(a_j) \geq Max2(a_i)$$

where $Max2(a_i) = PNR_{\beta_M(a_i)}(AP_M(a_i))$. □

Proposition 2 ([8]). *Let $M \in \mathcal{R}(\mathcal{N}, M_0)$ be a safe state following Definition 6. Then, M_0 is reachable from M.* □

So, we can establish the same corollary as in the previously presented improved policy. The same implementation details presented for the first improvement can also be applied now.

Let us study the computational cost.

Computation of the Max2 matrix: The algorithm for the computation of this matrix is an adaptation of the one for $Max1$: for each place in P_S we must keep record not only of a unique vector of maximal needs, but of a list of such vectors (one vector for each possible processing path from the considered state place to the idle state place). Since such paths are determined by $\mathcal{T}(p_{0_1}) \dots \mathcal{T}(p_{0_m})$, the $Max2$ matrix can be computed in $O(card(P_S)^2 \times card(P_R) \times (card(\mathcal{T}(p_{0_1})) + \dots + card(\mathcal{T}(p_{0_m})))$

Decision about the safety of a state: Let us consider the reachable marking M, and let K_M be the number of state places marked at M. For each one of these places we can have a set of different paths to the corresponding idle state place; let p be one of such places, and L_p the number of paths related to p. In order to solve the problem, we must find, for each one of the marked state places, one path in such a way that all the chosen paths can be ordered as in the previous improvement of the Banker's algorithm. This means that, in the worst case, we could have to solve $L = \prod_{\{p \in P_S | M(p) \neq 0\}} L_p$ Banker's problems. Then, according to the first improvement presented, the decision cost of this second one is $O(K_M \times log(K_M) \times card(P_R) \times L)$. Notice that the number L can be non-polynomial (considering the number of places and transitions as the size of the problem).

5 Conclusions

In this work we have presented a class of Petri net models appearing in the modelling and control of production systems. Among the wide set of problems that appear in this class of systems (due to its concurrent nature) we have concentrated on the deadlock problem, presenting three different algorithms for the control of such systems. Once controlled, it can be assured that each production order, once started, can finish. This allows the system to run in an automatic way.

References

1. F. Tricas, J. Colom, and J. Ezpeleta, "A solution to the problem of deadlocks in concurrent systems using Petri nets and integer linear programming," in *Proc. of the 11th European Simulation Symposium* (G. Horton, D. Moller, and U. Rude, eds.), (Erlangen, Germany), pp. 542–546, The society for Computer Simulation Int., Oct. 1999.
2. M. Fanti, B. Maione, S. Mascolo, and B. Turchiano, "Event–based feedback control for deadlock avoidance in flexible production systems," *IEEE Trans. on Robotics and Automation*, vol. 13, pp. 347–363, June 1997.
3. F. Tricas, J. Colom, and J. Ezpeleta, "Some improvements to the Banker's algorithm based on the process structure," in *To appear in Proc. of IEEE Int. Conf. on Robotics and Automation*, (San Francisco, USA), 2000.
4. T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, pp. 541–580, Apr. 1989.
5. M. Silva, *Las Redes de Petri en la Automática y la Informática*. Madrid: Editorial AC, 1985.
6. K. Xing, B. Hu, and H. Chen, "Deadlock avoidance policy for Petri–net modeling of flexible manufacturing systems with shared resources," *IEEE Trans. on Automatic Control*, vol. 41, pp. 289–295, Feb. 1996.
7. 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. Journal of Flexible Manufacturing Systems*, no. 10, pp. 73–100, 1998.
8. F. Tricas, J. Colom, and J. Ezpeleta, "Some improvements to the Banker's algorithm based on the process structure," Tech. Rep. GISI–RR-97-19, Dept. de Informática e Ingeniería de Sistemas, Univ. de Zaragoza, Mar. 2000.
9. E. M. Gold, "Deadlock prediction: Easy and difficult cases," *SIAM J. Comput.*, vol. 7, pp. 320–336, Aug 1978.