A PETRI-NET-BASED COLLISION AND DEADLOCK AVOIDANCE SCHEME FOR FMS

Jie Wu
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431

Hanqi Zhuang
Department of Electrical Engineering
Florida Atlantic University
Boca Raton, FL 33431

Abstract

Automated manufacturing systems, including Flexible Manufacturing Systems (FMS's), belong to the class of discrete event dynamic systems. In such a system, potentially conflicting events may occur due to concurrency. The problem of collision and deadlock avoidance can be investigated using Petri nets which are powerful techniques suitable for modeling concurrent processes. A Petri net based approach is employed in this paper to model, detect and avoid collisions and deadlocks in FMS's. Unlike the existing Petri net based techniques, the proposed approach uses the concept of critical states to avoid the system from entering a state leading toward a deadlock state. A multirobot assembly example is used to illustrate the proposed scheme.

Key words: Concurrent processes, deadlock, Petri nets, flexible manufacturing systems.

1. Introduction

Automated manufacturing systems, including Flexible Manufacturing Systems (FMS's) belong to the class of discrete event dynamic systems [Visw90]. In a typical FMS raw materials enter the system at the discrete points of time and are processed concurrently, sharing a limited number of resources. In such a system, potentially conflicting events may occur due to concurrency. For example, a number of robots installed in an assembly line access common space at times [Schn87, Bana90], and the operation sequences of the robots must be coordinated to avoid collisions. An equally important issue is how to control these robots to avoid deadlocks.

There has been a growing interest in methods for modeling, scheduling and performance analysis of Flexible Manufacturing Systems in general, and for handling the collision and deadlock avoidance problem in FMS's in particular. For instance, Shaffer and Herb [Shaf92] presented a data structure and an update algorithm for a prototype real-time collision avoidance safety system simulating a multirobot workspace. The data structure is basically a variation of the octree. Manivannan [Mani93] proposed a knowledge based approach to handle dynamic changes in workcell configuration to identify potential obstacles and to determine a collision-free path.

Petri nets are a graphical and mathematical modeling tool applicable to many systems. As a graphical tool, Petri nets are easier to understand due to the graphical and precise nature of the representation scheme. As a mathematical tool, it is possible to set up state equations and analyze the behavior of the system. Their success can be attributed to their simplicity, formality, their graphical nature, and their analytical capabilities which allow one to reason about important properties such as reachability, liveness, and boundedness. Petri nets can be used to model both the static and dynamic properties. Static properties of systems are represented by the graphical part of a Petri net. Dynamic properties of a system can be determined by the Petri net graph, the initial marking, and the simulation rules.

Techniques based on the Petri net theory have been found to be suitable for modeling manufacturing systems (for instance, refer to [Nara85] and [Brun86]). They are gaining increasing recognition in industry as a practical tool for the design and analysis of flexible manufacturing systems. The problem of collision and deadlock avoidance can be investigated using Petri nets which are powerful techniques suitable for modeling concurrent processes. Viswanadham et. al. [Visw90] showed that prevention and avoidance of FMS deadlocks can be implemented using Petri net models. Babaszak and Kroch [Bana90] developed a Petri net model of concurrent job flow and dynamic resource allocation in an FMS. They also presented a deadlock avoidance algorithm that can effectively avoid the so-called restrictive deadlocks. A comprehensive survey was given by D'Souza and Khator [D'So94] regarding application of Petri nets in modeling controls of automated manufacturing systems to avoid system deadlock.

However, a majority of available collision-avoidance algorithms based on Petri nets are basically static [Bana90, Visw90]. In such an approach, all possible sequences of operations for a FMS are listed, and if there is a potential collision, some of the robots (or other machines) are removed from the work cell [Bana90]. Similar procedures have been proposed for deadlock prevention in an FMS. A deadlock free sequence is usually chosen off-line among all possible operation sequences [Visw90].

A Petri net based approach is employed in this paper to model, detect and avoid collisions and deadlocks in FMS's. This approach is based on a novel idea of critical state to prevent the system from entering a state leading towards a deadlock state. We combine the task of collision avoidance with that of deadlock avoidance, resulting in an efficient algorithm that systematically generates critical states from a given system. First, we review some preliminaries of deadlock and Petri nets. We then present a collision and deadlock avoidance algorithm. A case study is also given to illustrate the applicability of the proposed algorithm.

This paper is organized as follows: Section 2 overviews several basic concepts, including FMS and Petri nets. A collision and deadlock avoidance algorithm is proposed in Section 3. In Section 4, an example of multi-robot flexible assembly cell is used to illustrate the proposed algorithm. Concluding remarks are given in Section 5.

2. Preliminaries

2.1 Flexible manufacturing system

An flexible manufacturing system is built to manufacture different types of products. For this purpose, raw parts of various types enter the FMS at a discrete points of time and are processed concurrently, sharing a limited number of resources such as numerically controlled machines, robots, material handling system fixtures, and buffers. By a resource we mean an element of the system that is able to hold a product (for transport, operation, storage, quality control) [EZPE95]. Every product follows a route through the set of system resources, according to a preestablished working plan. The sequence of operations performed in order to manufacture a product is termed a working process. Working processes in a FMS are executed concurrently, and therefore, they have to complete for common resources, which may cause deadlocks.

2.2 Deadlock

A deadlock occurs when a set of processes in a system is blocked waiting on resources that can never be satisfied. A process is a program in running and a resource can be memory, CPU and

I/O in a computer system or a part, a track and a robot in an FMS. These processes, while holding some resources, request access to resources held by the other processes in the same set. That is, the processes are involved in a circular wait. Deadlocks have been observed in FMS's when parts were processed concurrently at workstations. In general, deadlocks are difficult to predict in advance and they result in suboptimal performance.

Formally, a deadlock can arise if and only if the following four conditions hold simultaneously [Colf71, Islo80, Pete85]:

1. Mutual exclusion: no resource can be shared by more than one process at a time.
2. Hold and want: there must exist a process that is holding at least one resource and is waiting to acquire additional resources that are currently being held by other processes.
3. No preemption: a resource cannot be preempted.
4. Circular wait: there is a cycle in the wait-for graphs.

There are three strategies for handling deadlocks:

1. Deadlock prevention: prevents deadlocks by restraining how requests are made to ensure that at least one of the four deadlock conditions cannot occur.
2. Deadlock avoidance: dynamically grants a resource to a process if the resulting state is *safe*. A state is safe if there is at least one execution sequence that allows all processes to run to completion.
3. Deadlock detection and recovery: allows deadlocks to form and then finds and breaks them.

A deadlock situation may occur if, and only if, the four necessary conditions hold simultaneously in the system. To prevent deadlocks, one only needs to ensure that at least one of the necessary conditions never occurs. The problem of deadlock avoidance normally requires less stringent conditions than that of deadlock prevention, and it uses a priori information on how each process will utilize the resources. In general, deadlock detection and recovery is an overly optimistic approach for FMS's.

2.3 Petri nets

Petri nets [Ager79], [Pete81] are powerful tools to study the behavior of distributed computer systems. As a modeling tool Petri nets can be used to model both the static and dynamic properties of systems. On the other hand, as a graphically-oriented specification tool they appear to be one of the best approaches for enhancing interaction among users and specifiers and for easy human comprehension.

A Petri net is a five-tuple, $C = (P, T, I, O, \mu)$, where $P = \{p_1, p_2, ..., p_n\}$ $(n > 0)$ is a finite set of *places*, $T = \{t_1, t_2, ..., t_m\}$ $(m > 0)$, is a finite set of *transitions* and T and P are disjoint ($P \cap T \neq \Phi$). $I : T \rightarrow P$ is the input function, a mapping from transitions to *bags* (or multi-sets) of places. $O : T \rightarrow P$ is the output function of a similar type of mapping. Vector $u = (u_1, u_2, ..., u_n)$ gives, for each place, the number of tokens in that place and is called a *marking*. An example of a Petri net is given below.

Example 1: University Semester System [WuJi90]:

$C = (P, T, I, O, \mu)$
$P = \{p_1, p_2, p_3, p_4\}$
where $p_1 =$ fall semester, $p_2 =$ spring semester,
$p_3 =$ summer semester, $p_4 =$ not summer semester.

$T = \{t_1, t_2, t_3\}$

where t_1 = start of spring semester,

t_2 = start of summer semester,

t_3 = start of fall semester.

$u = \{1, 0, 0, 1\}$

$O(t_1) = \{P_2\}$ $\quad I(t_1) = \{P_1\}$

$O(t_2) = \{P_3\}$ $\quad I(t_2) = \{P_2, P_4\}$

$O(t_3) = \{P_1, P_4\}$ $\quad I(t_3) = \{P_3\}$

Petri nets can be represented by graphs where a *circle* represents a place and a *bar* represents a transition. The input and output function are represented by directed arcs from the places to the transitions and from the transitions to the places. The Petri net graph for Example 1 is shown in Figure 1. In this example a university semester system consists of three semesters: fall, spring, and summer, each of which is represented by a place. The transitions between two semesters are represented by transitions in the Petri net.

Petri net execution is represented by the movement of token. Tokens move through a transition firing. When a transition is validated, i.e., each of its input places has at least one token, a transition can fire, which removes one token from each of its input places and deposits one token into each of its output places. The patterns of different token distributions represent systems states. The example of the university semester system is a repetition of three states shown in Figure 2 with the fall semester as the initial state.

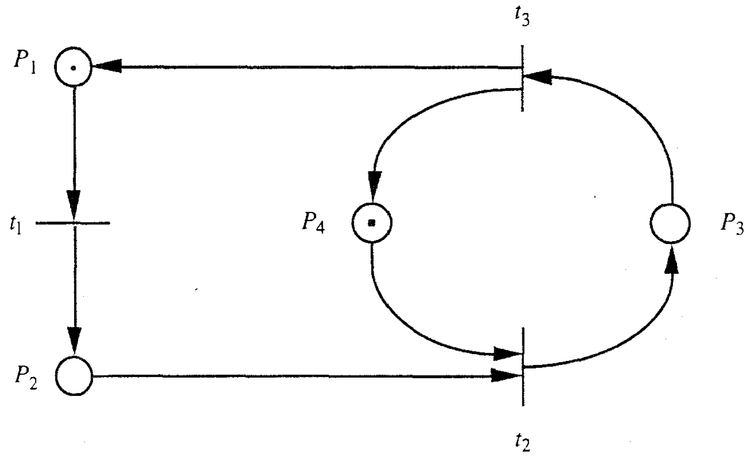

Figure 1: Petri net graph for a university semester system.

Figure 2 is also called a *reachability graph*. It consists of a tree whose nodes represent markings of the Petri net and whose arcs represent the possible changes in state resulting from the firing of transitions. A transition is *dead* in a marking if there is no sequence of transition firings that can enable it. A system is said to be in a deadlock state if all the transitions are dead.

In general, deadlock detection and avoidance algorithms developed for computer operating systems need to be modified for application in an FMS due to the method of specifying resource requirements. In the next section, we propose a deadlock avoidance algorithm for an FMS.

The reachability analysis can be extended to generalized Petri nets. Extensions to Petri nets can be classified into two groups: extensions that add time modeling capabilities, and extensions that add functional model capabilities. The inclusion of inhibitor arc is one of the fundamental extensions that add functional modeling capabilities. An inhibitor arc, represented by a dot attached at the end of the link that connects a place to a transition, enables the transition if its associated input place has a marking of zero tokens. A number of timed Petri net models has been proposed where timing aspects are incorporated. In Temporal Petri Nets [Merl76], each transition is associated with a time interval (t_{min}, t_{max}), where t_{min} is the minimum duration of sensitization of the transition and t_{max} is the time before which the transition must be fired. Another type of timed Petri net can be generated by associating each transition with a duration of time [Rama80]. When a transition is enabled, it is immediately fired and removes the enabling tokens from its input places. the token disappear and new tokens are created in the output places when the duration associated with the transition is elapsed. By associating each place with a duration of time, another type of timed Petri net is generated [Cool83]. In this model, a token created by a transition firing in a place becomes ready only after the delay associated with the place is elapsed. A transition fired immediately when it becomes enabled. It can be shown that the above three models can be simulated from one another; therefore they are equivalent. The problem of integrating the representation of time and functional aspects in high level Petri nest has also been studied [Ghez91].

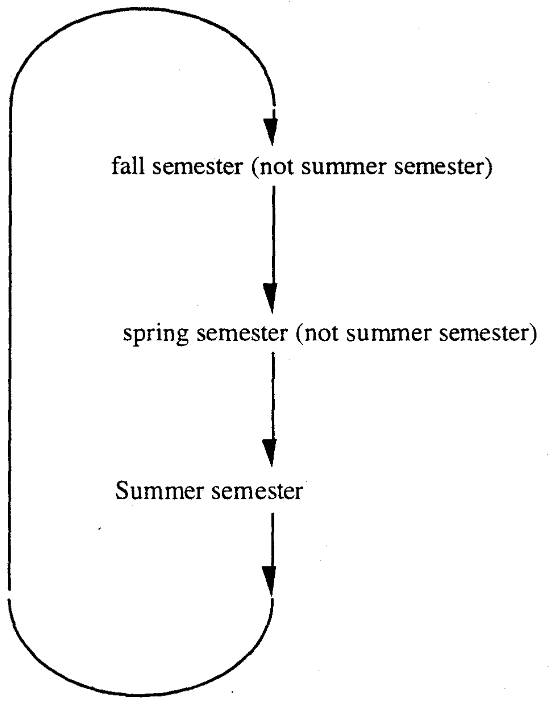
fall semester (not summer semester)
spring semester (not summer semester)
Summer semester

Figure 2: A state diagram of the university semester system.

The original Petri net model has limited modeling power but extensive decision power. The extended Petri net models, such as the timed Petri nets, have increased modeling power but limited decision power, i.e., the formal analysis in extended Petri net models is more complex than the original model. In our study here, we use the basic Petri net model to illustrate the concepts.

3. Collision and Deadlock Avoidance Algorithm

We combine collision avoidance with deadlock avoidance. In general, the collision and deadlock avoidance in a FMS using Petri nets requires the following steps:

1. Provide a specification of the FMS.

2. Obtain the Petri net model of the FMS (for Petri net modeling techniques, refer to [Pete81, WuJi90]).
3. Derive the corresponding reachability graph of the Petri nets.
4. Determine the collision and deadlock states from the reachability graph.
5. Find all the critical states together with their inhibited transitions based on the collision and deadlock states.

In general, a specification of the FMS can be either *model-oriented* or *property-oriented.* A model-oriented specification is an explicit system model constructed out out of abstract or concrete primitives. A property-oriented specification is given in terms of axioms which define the relationships between operations: no value or explicit constructions are defined in this type of specification. We use here the model-oriented specification which is similar to a Petri net specification in terms of their semantics.

Wu and Fernandez [WuJi90] proposed three models of Petri nets that describe a given system: *event-condition model, token-object association model,* and *place-object association model.* We will use the event-condition model to represent a FMS system. In this model, the notation of "condition" and "event" is used where a condition is considered as a passive component in the system while an event is considered as an active component. For each event there is a pre-condition and a post-condition. Normally, events are presented as transitions and conditions as places in a Petri net.

There are standard techniques for deriving the reachability tree of a given Petri net and there are several reduction methods which transform an infinite reachability graph to a finite one without losing important information. Fortunately, most FMS's have finite number of states, therefore there is no need for reduction. Note that most FMS's repeat a predefined sequence of actions over and over again, states are also repeated. As mentioned in the previous section, a deadlock state is identified by pinpointing dead transitions. A state (place) is *critical* if it is the closest state to a deadlock state that can still reach other states that do not lead to a deadlock state. Critical states can be identified by traversing the *reversed reachability graph* from all the deadlock states. Once all the critical states are determined, the corresponding transitions that lead toward deadlocks are also identified. These transitions are called *inhibited transitions.*

Note that the concept of critical state is similar to that of *unsafe* state used in the normal deadlock avoidance algorithms. Recall that a state is *safe* if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock. Any other states are call unsafe states. The difference between an critical state and an unsafe state is that any next state of an unsafe state may or may not lead to a deadlock and most often these two situations cannot be determined. On the other hand, the next stage of a critical state is either in a non-critical state (or safe state) or in a state that leads to a deadlock. Therefore, the critical state provides more (accurate) information than an unsafe state. In general, a critical state is an unsafe state while an unsafe state may or may not be a critical state.

It is easy to see that the proposed approach can be extended using a generalized Petri net, such as a net with inhibitor arcs. In this case, reachability graph can be derived following the normal procedure. Note that an inhibitor arc allows a transition to fire if its associated input place has a marking of zero tokens. Once a reachability graph is obtained, subsequent analysis is the same as the one employing a regular Petri net. Sometimes, if we are only interested in the status of a particular subsystem (for instance, we want to check if a particular subsystem is in a deadlock status) a color Petri net can be used, in which case each token associated with a specific color representing state information of a subsystem. If there is no token movement of a particular color, it then corresponds to a deadlock of the subsystem.

4. A Multirobot Flexible Assembly Cell Example

The following example of a multirobot flexible assembly cell shows how our method works. The system (Figure 3) consists of two robots performing various pick-and-place operations, accessing common space at times to obtain and transfer parts. It is assumed that each robot always holds a resource (one of the discrete regions in space), and the current resource cannot be relinquished until the next resource in the production sequence becomes available.

The two paths defined in the workspace of Figure 3 correspond to the production sequences of these two robots. When both robots enter the shadowed regions a collision will occur. A deadlock situation involves two types of entities: active entities called processes (the robots in the flexible assembly cell example) and passive entities called resources (the discrete regions in the same example).

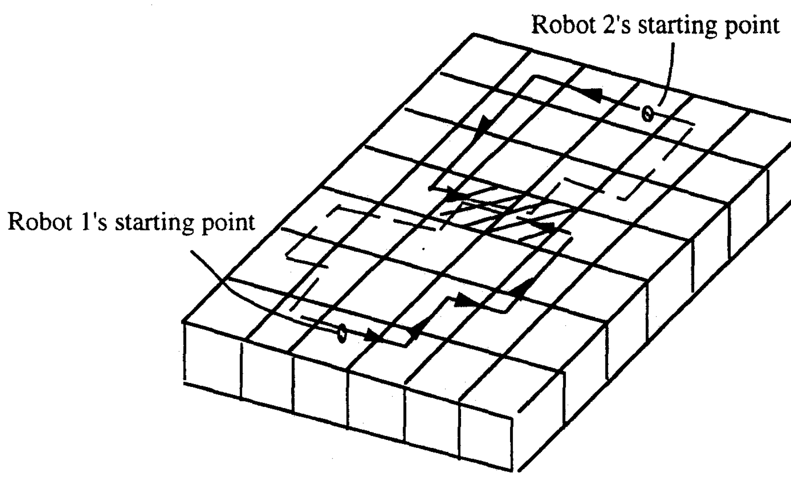

Figure 3: The flexible assembly cell with two robots.

The collision and deadlock situation can be depicted by the Petri net of Figure 4. The place and transitions in this figure have the following interpretations:

Transitions:

t_{i1} $(i = 1, 2)$:	Robot i's production sequence before entering the shadow regions sr_1 and sr_2.
t_{i4} $(i = 1, 2)$:	Robot i's production sequence after entering the shadow regions.
t_{12}, t_{23}:	Requests for sr_1.
t_{22}, t_{13}:	Requests for sr_2.
t_c:	Collision.

Places:

p_{i1} $(i = 1, 2)$:	Robot i's initial state.
p_{i2} $(i = 1, 2)$:	Robot i's state before entering the first shadow region.
p_{i3} $(i = 1, 2)$:	Robot i's state before entering the second shadow region.
p_{i4} $(i = 1, 2)$:	Robot i's state after leaving the shadow regions.
sr_1, sr_2:	Shadow regions.
rs_i $(i = 1, 2)$:	Presence of Robot i in the shadow regions.
co:	Destruction state.

The approach to analyze Petri nets is to use the reachability graph. The nodes (or states) of the reachability graph of a Petri net represent the reachable markings of the net. Figure 5 shows the reachability graph of the flexible assembly cell example.

In general, deadlock avoidance is a stronger requirement than collision avoidance. In the flexible assembly cell example, the deadlock avoidance problem includes the collision avoidance problem. A deadlock situation is equivalent to a state in the reachability tree which has no firable transition (such as the state $p_{13}p_{23}rs_1rs_2$ where the only firable transition is the destruction state). To avoid deadlock (including collision) in the flexible assembly cell example, we only need to prohibit transition t_{22} at the state $p_{13}p_{22}rs_1sr_2$ and t_{12} at the state $p_{12}p_{23}sr_1rs_2$. States $p_{13}p_{22}rs_1sr_2$ and $p_{12}p_{23}sr_1rs_2$ are termed *critical states*. The way to implement restrictions on certain transitions depends largely on applications. The problem could become complicated when autonomous robots are used, in which case each robot is required to keep a global state (the state in the reachability graph) in order to avoid the occurrence of deadlocks.

5. Conclusions

An approach for deadlock and collision avoidance in a flexible manufacturing system has been proposed in this paper. The concept of critical/non-critical states in deadlock avoidance has been shown as an enhancement of the traditional safe/unsafe concept. The proposed scheme has been shown to be effective by using the multirobot assembly cell example.

The strategy proposed in this paper can be applied to more general cases in which more robots are used in a FMS can also be extended to a model using a generalized Petri net such as a net with inhibitor arcs and color nets. Other possible applications include FMS buffer allocation and Automated Guided Vehicle coordination problems.

References

[Bana90] Banaszak, Z. A. and B. Krogh, "Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows," *IEEE Trans. Robot. Automat.*, Vol. 6, No. 6, Dec. 1990, pp. 724-734.

[Brun86] Bruno, G. and G. Marchetto, "Process translatable Petri nets for the rapid prototyping of process control systems," *IEEE Trans.Software Eng.*, Vol. 12, 1986, pp. 346-357.

[Coff71] Coffman, E. G., M. Elphick, and A. Shoshani, "System deadlocks," *Comput. Surveys*, Vol. 3, No. 2, June 1971, pp. 67-78.

[Cool83] Coolahan, J.E. and N. Roussopoulos, "Timing requirements for time driven systems using augumented Petri nets," *IEEE Trans On Software Engineering*, Vol. 9, No. 9, Sept. 1983.

[D'So94] D'Souza, K. A. and S. K. Khator, "A survey of Petri net applications in modeling controls for automated manufacturing systems," *Computers in Industry*, Vol. 24, 1994, pp. 5-6.

[Ezpe95] Ezpeleta, J., J.M. Colom, and J. Martinex, "A petri net based deadlock prevention policy for flexible manufacturing systems," *EEE Trans. Robot. Automat.*, Vol. 11, No. 2, 1995, pp. 173-184.

[Ghez91] Ghezzi, C. D. Mandrioli, and M. Pezze, "A unified high-level Petri net formalism for time-critical systems," *IEEE Trans On Software Engineering*, Vol. 17, No. 2, Feb. 1991, pp. 160-172.

[Islo80] Isloor, S. S. and T.A. Marsland, "The deadlock problem: an overview," *Computer*, Vol. 13, No. 9, Sept. 1980, pp. 58-77.

[Mani93] Manivannan, "Robot collision avoidance in a flexible assembly cell using a dynamic knowledge base," *IEEE Trans. Systems, Man and Cybernetics.*, Vol. 23, No. 3, May 1993, pp. 766-782.

[Merl76] Merlin, P.M. and D.J. Stolzy, "Recoverability of communication protocols-implications of a theoretical study," *IEEE Trans. on Communication*, Vol. 24, No. 6, June 1976, pp. 614-621.

[Nara85] Narahari, Y. and N. Vishwanadham, "A Petri net approach to the modeling and analysis of flexible manufacturing systems," *Ann. Operations Res.*, 1985, pp. 449-472.

[Pete81] Peteson, J.L., *Petri net theory and the modeling of systems*, Prentice Hall, 1981.
[Rama80] Ramamoorthy, C.V. and G.S. Ho, "Performance evaluation of a synchronous concurrent systems using Petri nets," *IEEE Trans. on Computers*, Vol. 6, No. 9, Sept. 1980.
[Schn87] Schneider, F., "Automated material handling in FMS and FAS," in *Automated Guided Vehicle Systems*, edited by R.H. Hollier, Springer-Verlag, 1987, pp. 171-191.
[Shaf92] Shaffer, C. A., "A real-time robot arm colision avoidance system," *IEEE Trans. Robot. Automat.*, Vol. 8, No. 2, April. 1992, pp. 149-160.
[Visw90] Viswanadham, N., 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, No. 6, Dec. 1990, pp. 713-723.
[WuJi90] Wu, J. and E.B. Fernandez, "Petri nets modeling techniques and their applications," in *Modeling and Simulation*, edited by W.G. Vogt and M.H. Mickle, Vol. 21, No. 3, 1989, pp. 1311-1317.

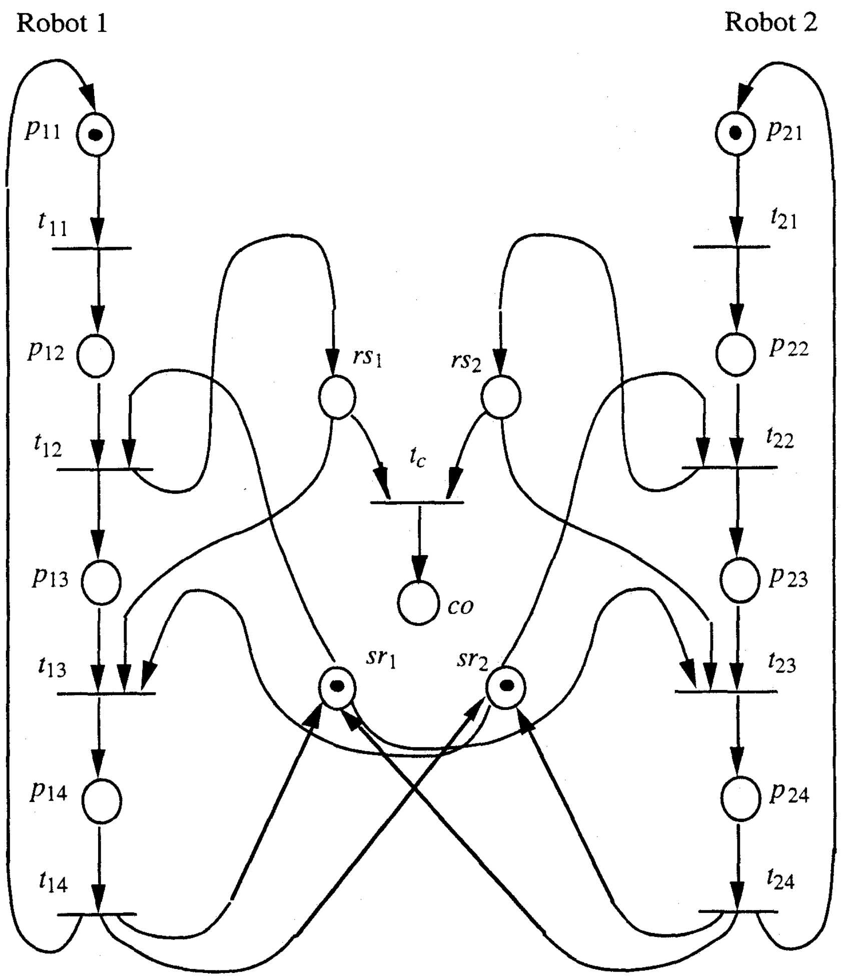

Figure 4: The Petri net representation of the flexible assembly cell example.

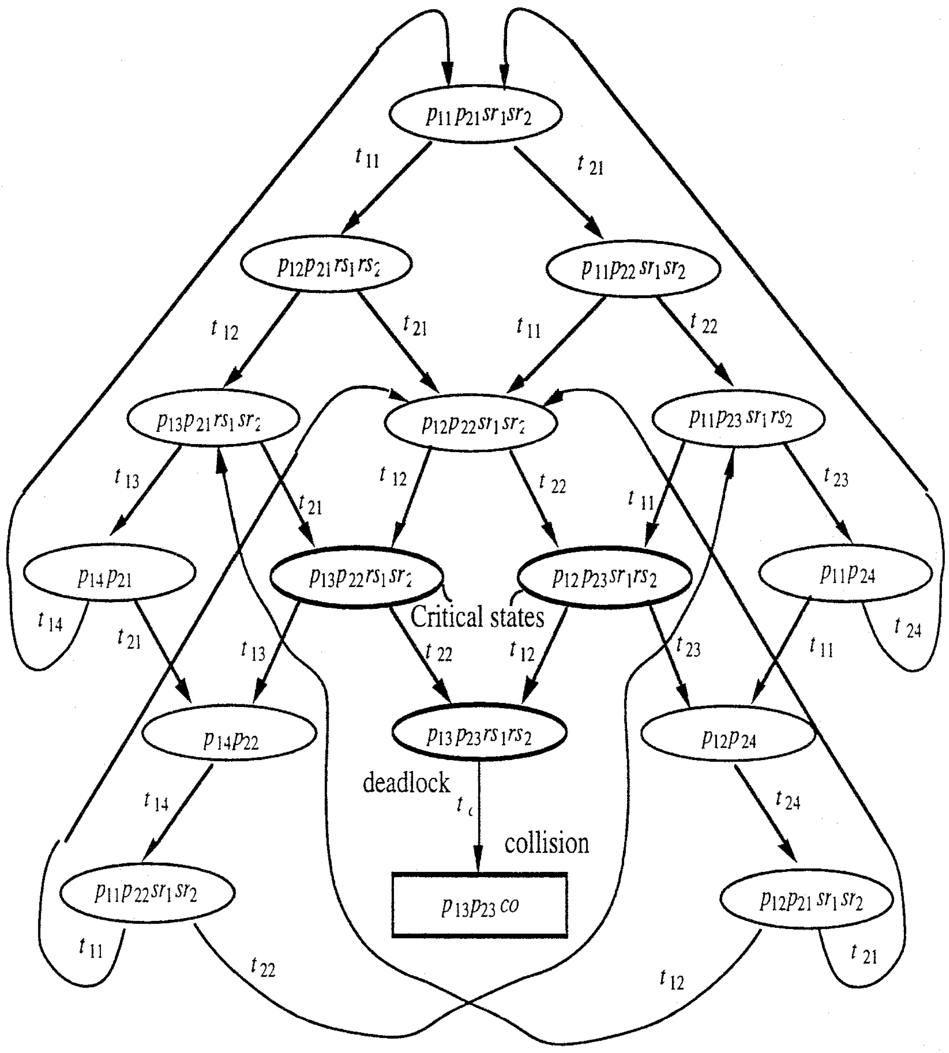

Figure 5: The reachability graph of the flexible assembly cell example.