# DEADLOCK AVOIDANCE IN AUTOMATED MANUFACTURING SYSTEMS

By

RALPH A. FERNANDES

Bachelor of Engineering

University of Bombay

Bombay, India

1992

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1995

OKLAHOMA STATE UNIVERSITY

**DEADLOCK AVOIDANCE IN AUTOMATED**

**MANUFACTURING SYSTEMS**

Thesis Approved:

Manjunath Kamath

Thesis Advisor

David B. Pratt

[illegible]

Thomas C. Collins

Dean of Graduate College

## ACKNOWLEDGMENT

This thesis represents a milestone in my career; a personal achievement on one hand, as well as the result of several individuals' efforts and encouragement. I extend my deepest appreciation to Dr. Manjunath Kamath for his consistent encouragement and belief in my capabilities as a student, for being a guide as well as a friend, as a true teacher should, and for supporting me as a research assistant at the Center for Computer Integrated Manufacturing. Many thanks are due to my other committee members, Dr. David Pratt, and Dr. Timothy Greene, who were always prepared to offer their assistance, and who taught me how to succeed within the classroom and beyond it.

Many thanks to my friends at the Center for CIM and elsewhere, for helping me feel at home in a new world.

Last but not least, my heartfelt gratitude must go to my family; my parents, sisters, and late uncle; their sacrifices and precious love have helped me reach this milestone, and will keep me going onwards.

TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

Figure Page

# CHAPTER I

# INTRODUCTION

## Background of the Problem

Automated manufacturing systems (AMSs) provide the flexibility needed for producing a wide variety of parts. These systems are characterized by limited resources which are shared among jobs. One of the problems arising in the control of such systems is the possibility of system deadlock. In an unmanned AMS, the problem of system deadlock becomes significant because a deadlocked system can remain in that state indefinitely, i.e. the system is crippled. Because of the complex sequence of operations in such systems, deadlocks are difficult to predict. It is clear that the nature of deadlock renders automated operation impossible. Once a deadlock occurs, human intervention is needed to clear buffers and machines and restore the system to a state that is known to produce deadlock free operation under normal conditions of operation. Designing and operating an AMS without deadlock considerations can lead to excessive manual intervention to resolve deadlocks and reset the system to a known manufacturing state [Wysk et al. 1994]. Such intervention violates the definition of unmanned (automated) operation, and the cost in terms of lost production and labor may be high.

The problem of deadlocks has been studied quite extensively in computer science as it is critical in multiprogramming environments. Here, several processes compete for a finite number of resources, e.g. the central processing unit and memory space. Coffman, Elphick, and Shoshani (1971) state four necessary and sufficient conditions which lead to a deadlock. These are: mutual exclusion, wait-for condition, no preemption, and circular wait. It follows that one of the ways in which a deadlock may be prevented from occurring is by ensuring that the four conditions listed above do not hold simultaneously.

While the four conditions given above were developed for computer systems, they are applicable to manufacturing systems as well. However, most of the detailed implementation of deadlock resolution strategies used for computer systems cannot be applied to manufacturing systems [Leung and Sheen 1993]. One reason is that deadlock resolution algorithms are dependent on the topology of the system and the way the system operates. Also, resolution procedures which are suited to computer systems may be inappropriate to manufacturing systems. For example, in a computer system, a program may be "trashed" in order to free space, while "trashing" a part in a manufacturing system is not desirable [Leung and Sheen 1993]. Most of the resolution procedures adopted for dealing with deadlocks in distributed systems are of the detection and recovery type. Once a deadlock is detected, recovery is effected by a roll back to a prevoius state. This is clearly not possible with AMSs. *However, Habermann's algorithm [Habermann 1969] which was the earliest detailed discussion on deadlock avoidance in the context of operating systems, forms an integral part of the research developed in this thesis. The adaptability of Habermann's theorem to manufacturing systems is promoted by the fundamental manner in which he approaches the problem.*

While deadlocks have been studied in the computer literature for quite some time, the same cannot be said of manufacturing. The problem of deadlocking in an AMS (which is more prone to deadlocks compared to conventional manufacturing topologies) has been ignored by most research in scheduling and control [Wysk et al. 1991].

Approaches to the deadlock problem in AMSs can be categorized by way of the strategies employed, or by way of the methodologies employed to resolve deadlocks. The related literature reveals essentially three different strategies to resolve deadlocks: (i) deadlock prevention, (ii) deadlock avoidance and recovery, and (iii) deadlock detection and recovery. Various methodologies are used for implementing these strategies, including: Petri net models to analyze the system under consideration or to execute these models in real time for controlling deadlocks, directed graphs to model part routings, and real time control algorithms. The three strategies employed for deadlock resolution are defined below.

## Strategies Employed in Deadlock Resolution

### Deadlock Prevention

Deadlock prevention is concerned with the formulation of policies for designing AMSs so that one or more of the necessary and sufficient conditions for deadlock hold false while the AMS is in operation. Since deadlock prevention is accomplished by imposing restrictions during the design stage itself, poor resource utilization results [Viswanadham et al. 1990].

## Deadlock Avoidance

Deadlock avoidance and recovery is concerned with avoiding potential deadlock situations in real time, and initiating automatic recovery procedures when a deadlock state could not be avoided. While deadlock prevention results in static resource allocation policies, deadlock avoidance results in dynamic resource allocation policies; consequently the resource utilization with this approach is expectedly higher.

## Deadlock Detection and Recovery

In this approach, deadlocks are allowed to occur, i.e. no attempt is made to avoid deadlocks using any type of look-ahead in real time. Once a deadlock state has been detected, a correction system is initiated to move one of the deadlocked parts into a buffer space reserved exclusively for use in such situations [Wysk et al. 1991]. The rest of the deadlocked parts are transferred sequentially to their next destination.

As will be seen in the literature review, some of the methodologies employed for resolving deadlocks guarantee deadlock free operation of the manufacturing system under consideration, however there are limitations on the types of manufacturing topologies that can be handled by these approaches. The other set of methodologies are general enough to be applied to a wide variety of manufacturing topologies; however, they cannot guarantee deadlock free operation of the system, or in doing so would be computationally expensive.

## The Problem Statement

At this point the idealized version of the problem may be stated: "To operate a fully automated manufacturing system so that (1) it is guaranteed to be deadlock free, (2) the control is non-conservative, i.e. only true unsafe states are avoided, (3) the algorithm is executable in real-time, and (4) no need for additional resources is imposed."

As it turns out, the general problem as stated above does not have a practical solution. The reasons are discussed in the following section. Instead, this research will be restricted to the following version of the general problem: "To operate a fully automated manufacturing system so that (1) it is guaranteed to be deadlock free, (2) the control algorithm has an acceptable level of conservativeness, (3) the algorithm has a polynomial time solution, (4) no need for additional resources is imposed."

## Deadlock Avoidance: An NP-Complete Problem

The class of NP-complete problems is a fairly recent discovery in the mathematical world. A practical explanation of an NP-complete problem is that no known polynomial time solution exists for that problem. The first NP-complete problem was discovered by Stephen Cook in 1970. Cook also discovered certain theorems to determine whether a problem is NP-complete or not. These theorems are among the most profound discoveries in the mathematical world [Dewdney 1993]. One of these theorems states that all NP-complete problems are equivalent in the sense that any one problem in that class can be reduced to any other in polynomial time. This means that if an efficient algorithm (i.e. an algorithm with polynomial time complexity) is found for any NP-complete problem, then every NP-complete problem can also be solved efficiently [Manber 89]. As of today, no one has found an efficient algorithm for any NP-complete problem, and it appears unlikely that such an algorithm will ever be found [Dewdney 1993]. It so happens that most real world problems either fall into the class of NP-complete problems or they do not. The latter class can generally be solved using low-degree polynomial time algorithms; hence their solutions are efficient in an ordinary sense. On the other hand, the former class of problems generally cannot be solved practically (leave alone a solution in polynomial time).

The general deadlock avoidance problem as stated above is NP-complete [Gold 1978]. Gold (1978) discusses various restricted classes of problems and classifies them as easy (solvable in polynomial time) or difficult (not solvable in polynomial time). The restrictions developed by him for the former class of problems are abstract, and difficult to justify in the context of real world manufacturing systems. Gold also provides proofs of the NP-complete nature of the general deadlock avoidance problem. In this author's opinion, the greatest use of Gold's paper is that by proving the NP-complete nature of deadlock avoidance for the general case, Gold has clearly delimited the solutions that can be expected for this problem. *In other words, the fact that deadlock avoidance is NP-complete in the general case, means that it would be impractical to strive for exact solutions to this problem.* Gold (1978) suggests three directions that research on this problem can proceed in: (1) fast solutions for restricted cases of this problem, (2) heuristic solutions for the general problem, which will usually solve it in polynomial time, even though exponential time will be required in the worst cases, and, (3) fast solutions for the general problem which will not always be correct, but will always err on the conservative side: all unsafe system states will be correctly classified, but some safe system states will be classified as unsafe. The research approach developed in this thesis takes the third direction.

# CHAPTER II

# REVIEW OF THE LITERATURE

The review of the related literature is presented under the headings of the methodologies used for deadlock resolution strategies, as discussed in the introduction chapter.

## Deadlock Prevention

Viswanadham et al. (1990) use Generalized Stochastic Petri nets (GSPNs) to model a hypothetical FMS. Petri nets are well suited to modeling systems which exhibit concurrency and conflict, which are characteristic of AMSs. The model is used for both prevention, and avoidance and recovery strategies. For the prevention strategy, a reachability tree is obtained from the GSPN model. The reachability tree depicts all the reachable markings from a given initial marking. Terminating states in the reachablility tree expose the deadlock states. An exhaustive path analysis of the reachability tree is then done to derive a set of resource allocation policies that prevent the occurrence of deadlocks. In arriving at these policies, one or more of the four necessary and sufficient conditions to deadlock are falsified. It is necessary to do such an analysis just once in order to devise the policies.

A major problem with the method outlined above is that the reachability tree grows explosively with the system size, thus practically ruling out this method for real-world systems.

Agerwala (1979) and Kamath and Viswanadham (1986) demonstrate the use of net invariants to determine whether a net has the potential for deadlocking. An efficient method of computing the invariants of a net during its synthesis is provided by Narahari and Viswanadham (1985). Once all the invariants are obtained, certain invariants are intuitively selected to analyze certain states for deadlocks. The disadvantages with such an approach are 1) the use of intuitive reasoning does not lend itself to the use of conventional computer algorithms, and 2) an exhaustive search would be needed in order to determine all possible deadlock states in the net. Since such a search process has an exponential time complexity, this approach essentially faces the same problem of state space explosion of a basic reachability graph analysis.

D'Souza (1994) developed a Petri net control model from a list of programmable logic controller events. The programmable logic controller (PLC) was developed for a simple automated cell in the Manufacturing Automation Laboratory at Rutgers University, comprising a CNC lathe, a pick-and-place robot, and an input buffer of raw parts. The Petri net control model is represented in a matrix definitional form (MDF). This is the graphical form of the Petri net recast in vector and matrix terms as:

$$\text{MDF} = (\, C^{+}, C^{-}, M_{o})$$

where $C^{+}$ = the output incidence matrix,

$C^{-}$ = the input incidence matrix,

$M_{o}$ = the initial marking.

The purpose of developing the Petri net model is to validate the programmable logic controller, i.e. to ensure that the controller does not permit the cell to enter a deadlock state. In validating the PLC, the Petri net model is used to generate all possible states that the automated cell is capable of entering. A deadlock detection algorithm checks for the presence of a terminating state ( i.e. a deadlock state). The algorithm defines a goal state which it attempts to reach from any given state by firing an appropriate sequence of transitions. The state equation defined below is used for this purpose.

$$M' = M_0 + u_0 \cdot C$$

where $M_0$ is the initial state,

$C$ is the incidence matrix; $C = C^+ + C^-$, and

$u_0$ is the control vector which defines the transitions to be fired.

The methodology discussed above is essentially identical to a reachability tree analysis, and hence suffers from the same major problem of state-space explosion. Since the controller developed for the system results in static resource allocation policies, poor resource utilization may result.

Another approach to deadlock prevention is the synthesis of a class of live nets using a bottom-up approach. Zhou and DiCesare (1991) use two types of elemental net structures to synthesize a live net. These structures are referred to as parallel mutual exclusion (PME) and sequential mutual exclusion (SME). A PME models a resource shared by distinct independent processes, while an SME models a sequential composition of PMEs. Sufficient conditions are obtained for synthesizing live nets from these structures. The restrictions imposed during the synthesis stage are translated into restrictions at the design stage. This means that scheduling policies designed independently of deadlock considerations may not be freely applied to the system.

## Deadlock Avoidance

Deadlock avoidance policies are by definition executed in real-time. Viswanadham et al. (1990) use the generalized stochastic Petri net referred to earlier and define a look-ahead of 'n' steps to construct a real-time controller for deadlock avoidance. When the state of the Petri net is defined by a vanishing marking (at least one immediate or zero-time transition is enabled), potential deadlock situations can be detected in advance. By selecting the appropriate immediate transition to fire, the deadlock can be avoided if the set of look-ahead markings had a non-deadlocked marking. Otherwise, if the look-ahead set had only deadlocked markings, then deadlock recovery procedures will be initiated in advance. Such a strategy would require the reservation of a buffer case for such occasions.

The number of look-ahead steps needed for complete deadlock avoidance cannot be computed. It can be seen that only infinite look-ahead can guarantee deadlock avoidance for general systems. This would result in an exponential time cost, which is to be expected given that avoidance in the general case is an NP-complete problem.

Banaszak and Krogh (1990) developed a framework for modeling flexible manufacturing systems using classical Petri nets. The net type developed is referred to as a Production Petri Net (PPN). The set of places in the PPN is divided into two disjoint sets; one representing the set of operations within a process plan, and referred to as operation places, and another for the resources of the system, referred to as resource places. The PPN models the production sequence for a product. The resource requirements are modeled separately. A deadlock state in the system is defined in terms places. The PPN models the production sequence for a product. The resource requirements are modeled separately. A deadlock state in the system is defined in terms of two types of transition enablings; process-enabled transitions and resource-enabled transitions. A process-enabled transition represents a job that is currently in the production step preceding the process-enabled transition. A resource is modeled using a pair of places, one to show the number of available units of that resource type, and another to show the number of busy units of that type. It is assumed that each production step in the sequence requires only one resource in the system. When a set of process-enabled transitions can never become resource-enabled, the set of transitions is said to be deadlocked.

The principle behind the authors' algorithm is that deadlocks are caused by shared resources (resources which are revisited in the course of a process plan) in the system. Hence their approach consists of restricting the number of parts that can enter into certain zones delineated by such shared resources. The production sequence is divided into a sequence of zones of the type

$$p_q = p_q(0)z_q^{1} \dots z_q^{n(p_q)} p_q(L_q+1)$$

where $p_q(0)$ represents a production order, $p_q(L_q+1)$ represents the completion of an order, $L_q$ is the length of the production sequence, and $n(p_q)$ is the number of unique sets of sub-zones of the form

$$z_q^{k} = s_q^{k} u_q^{k}, \quad k = 1,2, \dots, n(p_q)$$

The deadlock avoidance algorithm consists of two rules, DDA1 and DDA2. DDA1 states that a token can enter a new zone in the production sequence only when the capacity in the unshared subzone exceeds the number of jobs currently in the zone. Rule DDA2 requires that if a shared resource is being requested by the job, then all of the shared resources in the remainder of the zone must be available at the time. The authors then follow the deadlock avoidance algorithm with a proof to show that the restriction policy guarantees that restricted deadlock will not occur.

Although the algorithm guarantees that all deadlocks are avoided, there are many limitations imposed on the type of systems that can be modeled. A major limitation is that it restricts the definition of an FMS to those systems without alternative part routings, which is not in keeping with trends towards real-time scheduling and control of FMSs (thus permitting greater flexibility of the systems). Only sequential production processes are modeled; thus branching and merging cannot be included. Another assumption is that a production step requires only one resource in the system for a given part type. The algorithm is quite conservative. For example, when applied to the system used by the authors to demonstrate the algorithm, not more than 10 parts are permitted in one of the production zones, while up to 19 parts could be safely allowed (as this author determined). The algorithm becomes increasingly conservative as the proportion of shared resources in the system increases. The algorithm fails (does not allow the system to start) in the limiting case, when all the system resources are shared among processes. Given that it is not uncommon for all resources in an AMS to be shared, by virtue of the fact that they are "flexible" resources, it follows that the above limitation is a severe one.

Hsieh and Chang (1994, 1992) extend the methodology developed by Banaszak and Krogh (1990) to include processes holding multiple resources. Although this is a significant extension, the other major limitations remain, i.e. alternate routings or branching cannot be modeled. The algorithm is conservative, since it is based on a sufficient condition for liveness (rather than a necessary and sufficient condition). The authors claim that their algorithm permits high resource utilization levels; however, their algorithm is based on the same idea that Banaszak and Krogh use, and it follows that the algorithm will become increasingly conservative as the proportion of shared resources in the system increases.

Leung and Sheen (1993) developed a real-time control algorithm for avoiding deadlocks. The principle used in their algorithm is that if one of the four necessary and sufficient conditions for a deadlock (discussed earlier) is relaxed, then a deadlock situation can be avoided.

The authors claim that the algorithm is general enough to be applied to a wide variety of FMS's, however, they state that the algorithms cannot be proved to be optimal in any sense. Furthermore, the deadlock avoidance strategy requires buffer spaces to be specially reserved if all deadlocks are to be avoided.

## Deadlock Detection and Recovery

Wysk et al. (1991) developed a graph based method to detect deadlocks. Part routings are represented as directed graphs, where nodes represent machines and directed arcs represent flow sequences through the machines. Directed arcs are labeled with the part numbers contributing to the arc. The presence of a circuit in the graph is a necessary but not sufficient condition for deadlocking. Circuits in the graph are detected using string arithmetic. Additional inspection is then used to determine if the circuit satisfies the sufficiency conditions of a deadlock.

Given a graph G = (V, A), the following conditions are sufficient for a system deadlock.

a) There exists at least one circuit $C = (V_c, A_c)$.

b) The number of jobs contributing arcs to C must be equal to the number of arcs in C.

c) The number of machines in C must be equal to the number of arcs in C.

To implement the deadlock avoidance approach, a deadlock detection procedure is activated whenever a new part attempts to enter the system. Routing information for the new part as well as for remaining routes for all parts currently in the system is converted into a graph and examined for circuits. If a potential deadlock is detected, the part is not allowed to enter the system until such time that its entry does not cause a deadlock situation.

While the aim of this approach is to entirely avoid deadlocks (i.e. there should be no need for recovery procedures), this necessitates detecting combinations of circuits which could lead to deadlocks, while the individual circuits in themselves would not. A higher level circuit has lower level circuits as its nodes. For example, a second-level node is actually a circuit with machines as its nodes. It is clear then, that if deadlocks are to be entirely avoided, then all such higher-level circuits that could lead to a deadlock need to be examined. The level upto which such circuits would have to be examined is indeterminate.

As an alternative strategy, only the next immediate destination of parts in the system is used to form a graph. When a system deadlock is detected, a correction system is initiated to move one of the deadlocked parts (chosen at random) to a special reserved storage. The rest of the deadlocked parts are then transferred sequentially to their next destination. It is clear that routing beyond the immediate destination cannot produce a system deadlock, hence it is ignored in the graph construction. This approach corresponds to the deadlock detection and recovery strategy.

A major limitation in the graph-theoretic approach is that the part routing has to be fixed, i.e. alternate routings cannot be treated. Other limitations of this approach are that false deadlocks may be detected on the one hand, while true deadlock states may not be detected, on the other hand. Furthermore, a resolution from the deadlock state requires that a buffer space be reserved for such occasions.

A common limitation with all these strategies is that a complete resource allocation policy which optimizes resource utilization has not been defined, rather, the deadlock resolution algorithms have as their goal only the resolution of deadlocks. This limitation will be maintained in the development of this thesis.

# CHAPTER III

# STATEMENT OF THE RESEARCH

## Research Objectives

Objective I To develop an understanding of the various approaches to the resolution of deadlocks in unmanned flexible manufacturing systems and related systems, as presented in the literature.

Objective II To develop a new approach to the problem that *attempts* to (1) guarantee deadlock free operation of an AMS, while (2) maintaining an acceptable level of conservativeness; further, (3) it should be solvable in polynomial time, and (4) it should not require the reservation of resources as an aid to deadlock resolution. These criteria formed the structure of the practical problem statement. Given the NP-complete nature of the problem, we have opted to use the verb "attempts" in defining this objective.

Objective III To stimulate further research on the deadlock avoidance problem in unmanned flexible manufacturing systems, given the limited body of literature in the area.

## Research Scope And Assumptions

A widely studied problem in the flexible manufacturing systems literature is concerned with dynamically determining the part routing. By its very nature, an FMS is flexible, allowing for several alternate routings for a part type. In developing heuristics for part routings, the researchers have ignored the deadlock problem. This research does not propose to develop deadlock control strategies or a set of resource allocation policies that at the same time also result in an optimal or near optimal operation of the automated manufacturing system by design.

The following conditions will be assumed to hold for the automated manufacturing systems under consideration:

1. Mutual exclusion: A customer (part) requires the exclusive use of a resource.
2. Hold while waiting: A part holds on to resources while waiting for additional resources to become available.
3. No preemption: Parts holding resources determine when they will be released.
4. Availability of raw parts: For convenience in modeling, rather than as a necessity, it will be assumed that raw parts are always available, and consequently the availability of raw parts will not be modeled.
5. A machine that fails comes up in finite time.

The above conditions are not to be viewed as restrictions from the viewpoint of the deadlock problem. Conditions (1) through (3) are necessary for a system to deadlock, i.e. if any of these conditions were not true for a given system, then that system would not experience deadlocks in the sense that a recovery could be affected from any "deadlock" state.

## Performance Measures

The primary objective of this research is to study the efficacy of any approach developed by this author in avoiding deadlocks in the operation of the AMS under control. Since the scope of this research does not include development of optimal resource allocation policies, traditional performance measures like throughput and resource utilization are not considered. The performance metrics that are of relevance in this research include (1) the proportion of deadlocks avoided, (2) the time complexity of the algorithm, and (3) the conservativeness of the algorithm.

### Proportion of Deadlocks Avoided

The performance measure defined below will be used to determine the efficacy of the algorithms developed:

$$\text{Proportion of Deadlocks Avoided} = 1 - \frac{\text{No. of deadlock states that the AMS entered}}{\text{Total no. of deadlock states in the state space of the AMS}}$$

If the above performance measure is 1, then the algorithm has guaranteed deadlock free operation of the AMS under its control. If the performance measure is less than 1, then the algorithm did not guarantee deadlock free operation of the AMS. In this case the next logical step would be to compare the computational complexity of the algorithm (which affects the time needed to execute the algorithm in real-time), with that of another algorithm applied to the same system configuration. To make a proper comparison of the computational complexity, not only should the performance of the algorithm (i.e. the proportion of deadlocks avoided) in the control experiment be the same, but also the deadlock states avoided/visited should be the same. For example, identical avoidance performance measures can be achieved in the finite look-ahead algorithm of Viswanadham et al. (1990) by varying the look-ahead parameter, however this will not ensure that the deadlock states avoided/visited will be the same as with the proposed algorithm.

For this reason, the computational complexity of the algorithm will be estimated based on the number of computational steps to be executed in the algorithm. This approach will be preferred rather than using simulation to compare the execution speed of the algorithm with that of a comparable algorithm, because of the difficulty in obtaining identical results in the control experiment so that a meaningful comparison may be made, as discussed above.

Comparison of the computational complexities of the new algorithms with those of established algorithms would help in establishing guidelines on the use of the various approaches for different AMS types.

## Time Complexity

The time complexity of an algorithm is represented by the big 'O' notation, for example, a polynomial complexity is written as $O(n^b)$, where b is the degree of the polynomial. The relation between the big-O estimate and the time complexity f(n) is:

$f(n) = O(g(n)) \Rightarrow f(n) < C.g(n)$ when $n > k$; C and k are constants.

The table [Rosen 1988] below displays the time required for solving problems of various sizes using algorithms of varying complexities.

Bit operations used

| *Problem Size* | *Complexity* | | |
|---|---|---|---|
| n | $n^2$ | $2^n$ | n! |
| 10 | $10^{-7}$ sec. | $10^{-6}$ sec. | $3(10^{-3})$ sec. |
| $10^2$ | $10^{-5}$ sec. | $4(10^{13})$ yr. | $>10^{100}$ yr. |
| $10^4$ | $10^{-1}$ sec. | $>10^{100}$ yr. | $>10^{100}$ yr. |

Table 1: Bit operations used for various problems sizes and algorithmic complexities.

From the table, it is easy to see the great disparity between the time complexities of polynomial solutions versus exponential and factorial solutions, and also the reason why exponential and factorial solutions are impractical for large problems. One of the requirements of the problem statement is that the algorithm developed in this research have a polynomial time complexity.

## Conservativeness

The conservativeness of the algorithm determines how restrictive it is in execution. If an algorithm is too conservative, then it may result in poor resource utilization, and will interfere with the execution of scheduling policies when these are designed independently of deadlock considerations (as will usually be the case).

Let the state space of the system be S. Let the set of unsafe states in S be $S_u$. Let the state space of the system when controlled by the deadlock avoidance algorithm be $S_a$. Then a logical metric for conservativeness can be expressed as:

$$\text{Conservativeness} = 1 - \frac{|S_a|}{|S - S_u|}$$

Zero conservativeness means that the algorithm correctly avoids only true unsafe states. The metric defined above is appropriate only when the algorithm always errs on the safe side. If this were not true, then it is possible that some of the undetected unsafe states would count for some of the incorrectly avoided safe states. For such algorithms, the only way to establish the true conservativeness would be to compare every state in the original and controlled state space sets.

# CHAPTER IV

# RESEARCH APPROACH

## Design of the Test Systems

### Issues Related to the Design of the Test Systems

In order to test any algorithm that would be developed in the course of this thesis, two test systems were designed. System A was designed as an example of a system with concurrent processes, alternate process plans, and multiple resource holdings by a single process, which is modeled by the synchronization of a set of resources (which in turn can also be interpreted as assembly). System B was designed with a view to testing the algorithm for processes with alternate routings. Both these systems were designed prior to the development of the algorithms presented in this thesis; thus there could have been no bias towards any algorithm when the systems were designed.

The sizes of the test manufacturing systems were limited by the following: 1) the availability of appropriate software and hardware to develop the reachability tree corresponding to the state space of the system, and 2) the requirement that any algorithm developed would have to be tested over the entire state space to determine how conservative it was, and whether it would avoid all unsafe states. The first limitation is due to the fact that any software code (external) used to develop the reachability graph will have an exponential time and space complexity (due to the phenomenon of the state space explosion); thus there is a limit on the system size that can be handled when developing the reachability graph. The second limitation arises since the scope of this thesis does not include the implementation of the algorithm in a machine executable language. As will be seen later, the algorithm that this author developed was tested over the entire state space manually. The layout of System A is shown in Figure (1). The layout of System B is shown in Figure (3).

## Description of System A

The test System A has three types of resources, viz. machining centers M1 and M2, automated guided vehicles (AGVs), and fixtures A and B. There are two units of M1, one unit of M2, two AGVs, and two fixtures of each type (A and B). The routing of the AGV is determined by the process plan, which is shown in Figure (2).

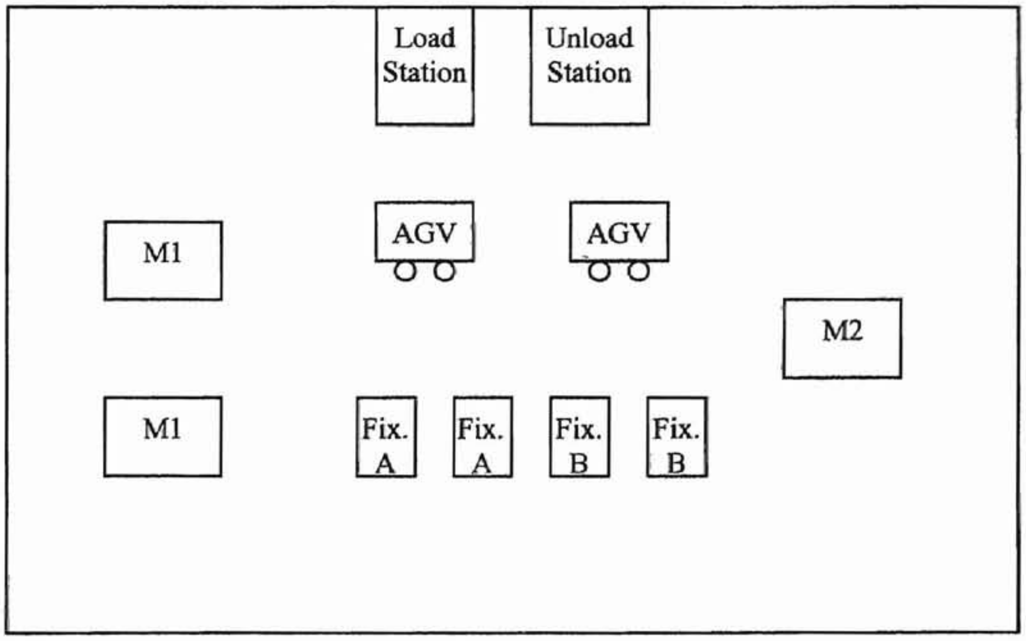


Figure (1): Automated Manufacturing System A

Two alternate process plans are modeled. The alternate process plans require that two machining operations be carried out on a part. Any operation may precede the other; however, each operation may be performed on only one type of machine. For e.g., M1 may perform drilling operations, while M2 may perform milling operations. The functions of the two machines cannot be exchanged, so that a part has to be routed through both machine types. A fixture of type A is used when the first operation is being performed on a part, while fixture B is used when the second operation is performed on the part.

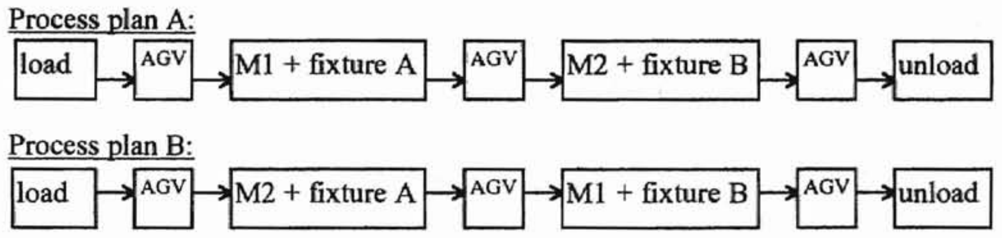


Figure (2): Alternative process plans for a raw part

The AGV is loaded with raw parts at the load station, carries them to either M1 or M2, where it awaits unloading. The AGV is unloaded only when both a fixture and machine are free, otherwise it remains blocked. The AGV also transports semi-finished parts to either machine M1 or M2, depending on which operation was completed on the semi-finished part. When a finished product (a part which has been processed on both M1 and M2) is available, the AGV can be assigned to carry it to the unload station.

## Description of System B

System B is shown in the layout diagram in Figure (3). A single process plan with two alternate routings is modeled in this system. The routing is shown in Figure

(4). There are two shared resources (resources which are revisited in the course of a process plan), M2 and the AGV.

Assumptions in the systems

The conditions defined earlier in the section "Research Scope and Assumptions" hold. In addition:

1. The unload station is assumed to have infinite capacity. (Such assumptions are necessary to define the boundary of the system model.)

Note that we have not explicitly assumed that the load station is of infinite capacity. This is not required, since the system boundary does not include the process of loading the load station. Hence, a load station filled to its capacity with raw parts would not have any impact on the potential of the system to enter into a deadlock. Having described the two systems, a framework is now presented for modeling the systems.

## Synthesis of a Blocking Restricted Petri Net

Deadlocks are defined by a circular blocked state, where each blocked resource in a set is awaiting another blocked resource in that set. A blocking operation can be viewed as a state in which a resource is waiting to be unloaded (by transferring its load on to another resource). From the definition of a deadlock as a circular blocked state, it follows that only the blocking operations are of relevance to the deadlock phenomenon. Hence it suffices to model the blocking operations only in the Petri Net (PN) model. When the scope of a PN model of the system is restricted thus, then the model is

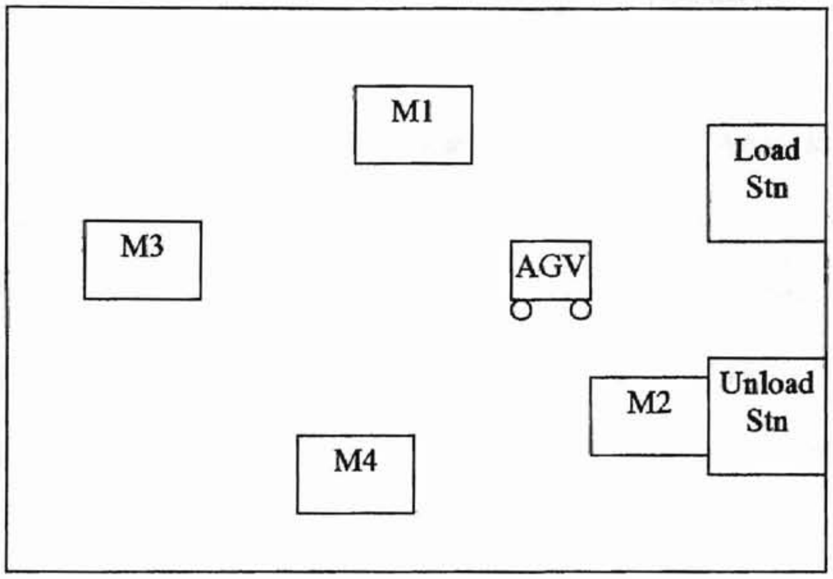


Figure (3): Layout for System B

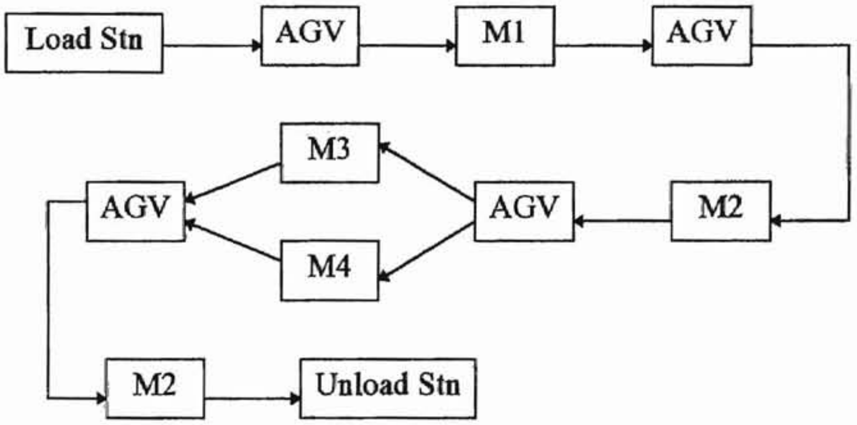
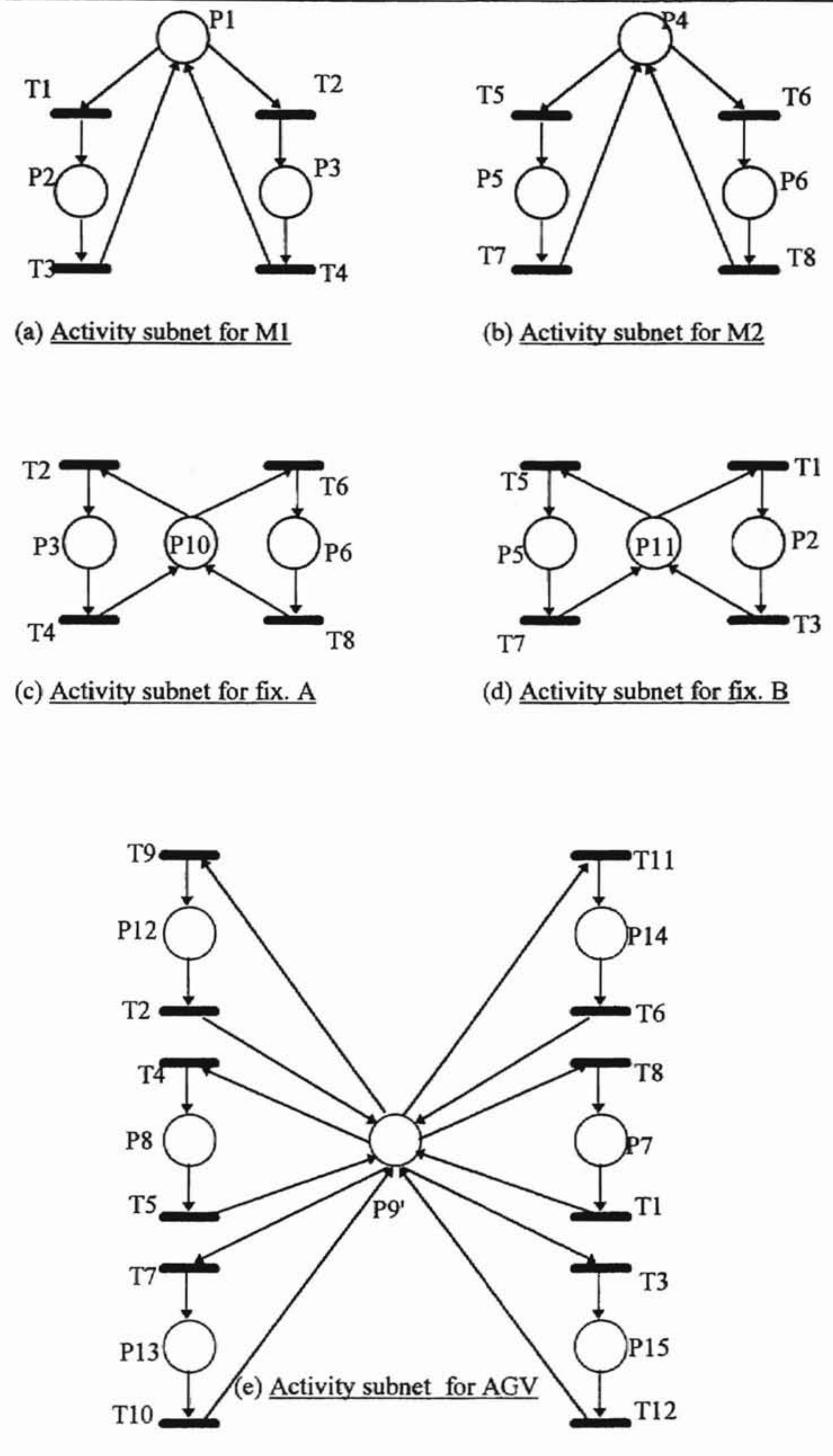


Figure (4): Routing for part in System B referred to as a Blocking Restricted Net (BRN). The BRN is synthesized from a union of subnets, where each subnet models the cycle of blocked activities that a resource may go through. These subnets are referred to as resource activity nets (RANs), and are shown in Figure (5) for the three types of resources in System A. The interpretation of the places is given in Table 2. The set of places can be classified into two types: (1) resource places, which model the availability of resources, and (2) operation places, which model the loading or unloading operations, i.e. the blocking operations of a process. The transitions have a dual interpretation, in that they represent the commencement or completion of a particular activity.

The synthesis of the complete BRN model is achieved through a union of these subnets, and is shown in Figure (6) for System A. The interpretation of places is given in Table 2. The union process essentially combines the identical places and transitions in the full PN model. The advantage of this bottom-up approach is clear: the modeler needs to deal with only one resource type at a time.

The synthesis of the BRN for System B is accomplished in a similar manner, and is shown in Figure (7). The interpretation of places is given in Table 3.

Having obtained the BRN of the system, we can identify the properties that give it the potential for deadlocking. These are: 1) sequential mutual exclusion (SME), and 2) parallel mutual exclusion (PME) [Zhou and DiCesare 1991]. In ordinary words, these concepts can be explained through examples as follows. An AGV can either load M1, or it can unload M1, but it cannot concurrently do both operations. This is an example of



(a) Activity subnet for M1

(b) Activity subnet for M2

(c) Activity subnet for fix. A

(d) Activity subnet for fix. B

(e) Activity subnet for AGV

Figure (5): Resource activity subnets for system A

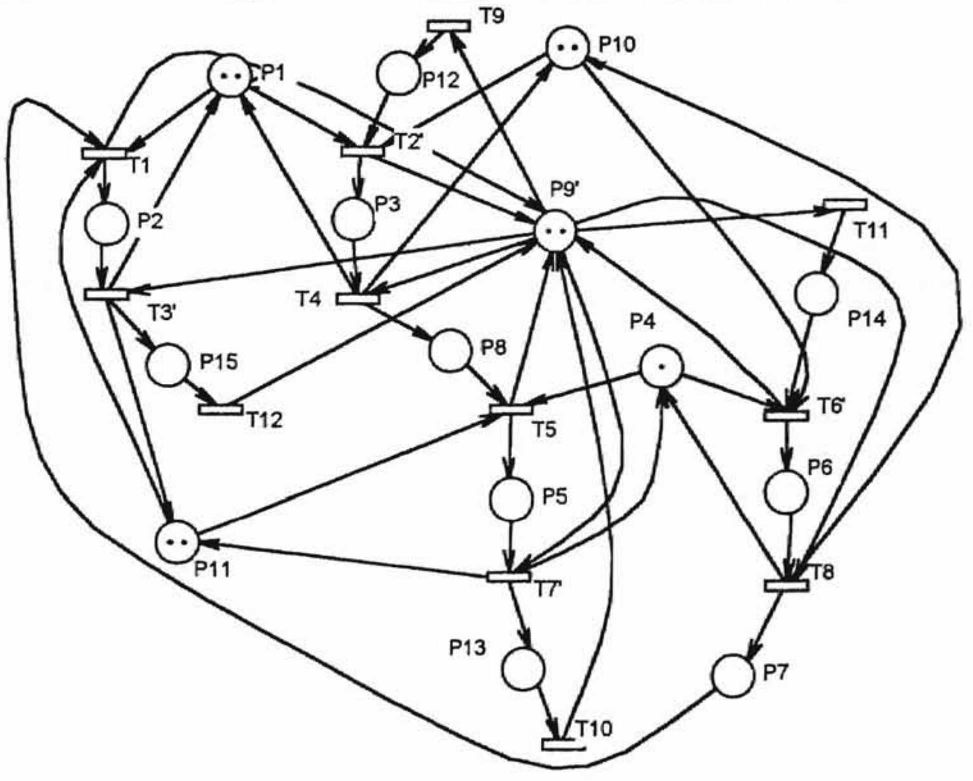


Figure (6). Synthesis of blocking restricted net (BRN) for System A from a union of resource activity subnets in Figure. (5)

| RESOURCE PLACE | INTERPRETATION |
|---|---|
| P1 | M1 available |
| P4 | M2 available |
| P9' | AGV available |
| P10 | Fixture A available |
| P11 | Fixture B available |
| OPERATION PLACE | INTERPRETATION |
| P2 | M1 processing semi-finished part on Fix. B |
| P3 | M1 processing raw part on Fix. A |
| P5 | M2 processing semi-finished part on Fix. B |
| P6 | M2 processing raw part on Fix. A |
| P7 | AGV with semi-finished part awaiting M1 |
| P8 | AGV with semi-finished part awaiting M2 |
| P12 | AGV with raw part awaiting M1 |
| P13 | AGV with finished product unloaded from M2 |
| P14 | AGV with raw part awaiting M2 |
| P15 | AGV with finished product unloaded from M1 |

Table 2: Interpretation of places for Figures (5) and (6).

| RESOURCE PLACE | INTERPRETATION |
|---|---|
| P1 | M1 available |
| P4' | AGV available |
| P6 | M2 available |
| P10 | M3 available |
| P11 | M4 available |
| OPERATION PLACE | INTERPRETATION |
| P4" | AGV with raw part |
| P2 | M1 busy |
| P3 | AGV carrying WIP from M1 |
| P5 | M2 busy |
| P7 | AGV carrying WIP from M2 |
| P7' | AGV moving WIP from M2 to M3 |
| P7" | AGV moving WIP from M2 to M4 |
| P8 | M3 busy |
| P9 | M4 busy |
| P12' | AGV moving WIP to M2 |
| P13 | M2 processing operation # 4. |

Table 3: Interpretation of places for BRN of System B in Figure (7)

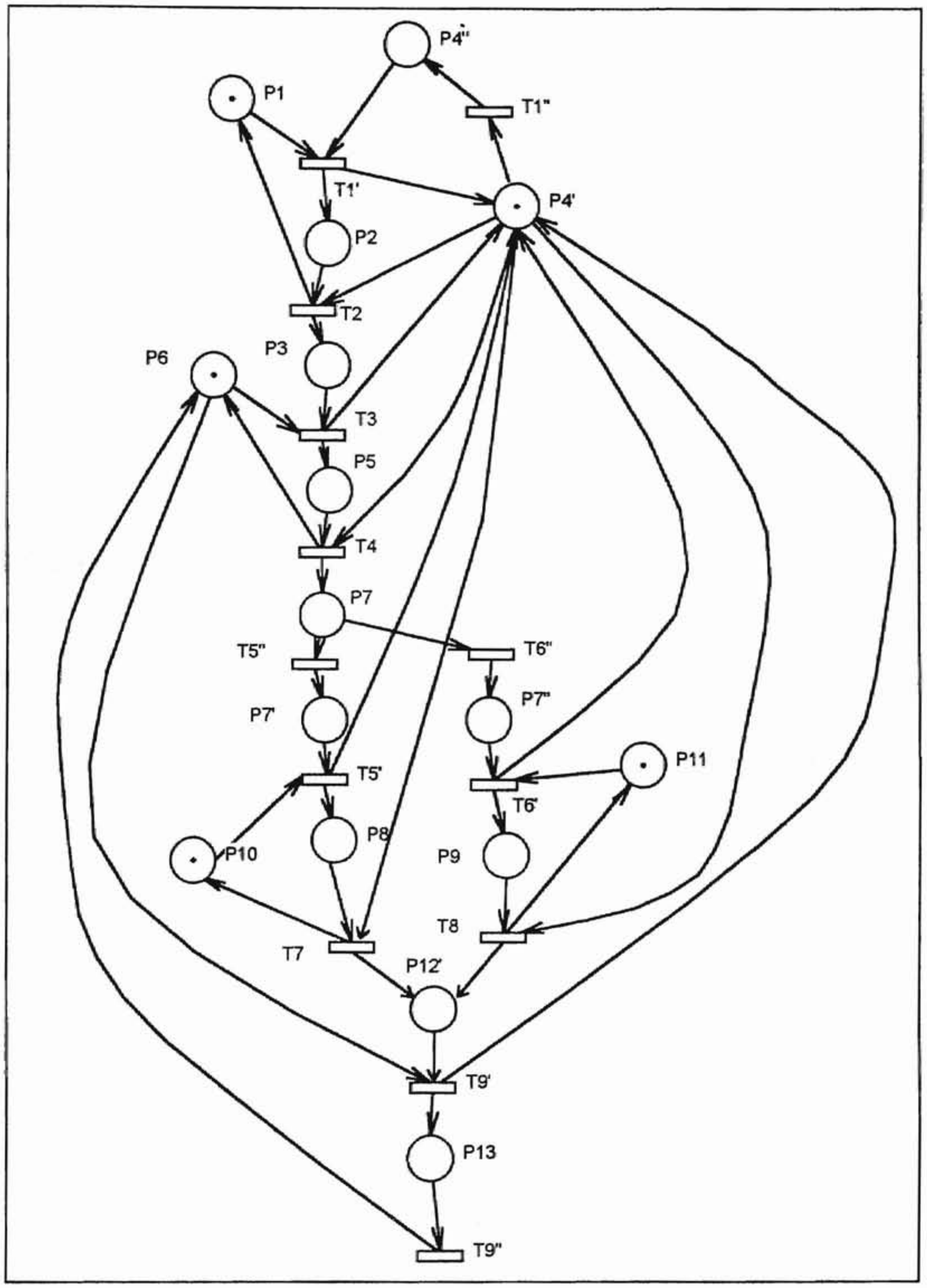


Figure (7): BRN for System B

SME. Second, an AGV can either load M1, or it can load M2; but it cannot do both concurrently. This is an example of PME. There are many other instances of SMEs and PMEs in this system.

It may be noted that although these properties are undesirable, no attempt is made to synthesize the net without these properties. An approach where this is done falls into the category of synthesis of live nets, and is critiqued in the literature review chapter.

## Development of the Research: A Preview

In an initial approach to the problem, this author developed a framework for modeling an automated manufacturing system from the perspective of the deadlock phenomenon. The framework uses classical Petri nets to model the blocking operations within the AMS, and was used to model two test systems in the preceding section. This framework is appropriate since any deadlock state can be defined in terms of a set of blocked resources, with each resource in the set waiting indefinitely for the other to become available. Petri nets was chosen as the modeling tool because of its suitability for modeling characteristics like concurrency and synchronization which are prevalent in an AMS. The resulting net is referred to as a Blocking Restricted Net (BRN). An example of synthesizing a BRN for System A was described in the preceding section.

Once a framework had been developed for modeling the deadlock phenomenon, the next step in the evolution of this research was the identification of a set of elementary reductions to be applied to the net model. The resulting reduced model is referred to as an e-BRN, for elementary blocking restricted net. The concept of reduction was motivated by the finite look-ahead algorithm of Viswanadham et al. (1990) (referred to as the VNJ algorithm). Such a concept was appealing because the state space of a Petri net is exponentially proportional to its size (number of nodes), except for simple nets such as marked graphs. Thus the state space of a reduced net would be significantly smaller than the state space of the original net. It follows that the average number of look-ahead steps needed to avoid the same subset of deadlocks as the VNJ algorithm (for the same system) would be lower.

Hence the improvements of this approach over the VNJ algorithm were achieved in two phases: (1) by restricting the scope of the Petri net model to only those operations of relevance to the deadlock phenomenon, a first reduction in the state space is implicit (the PN model used for the VNJ algorithm is not restricted to blocking operations), and (2) by further reducing the net model, the state space is further reduced. However, as was the case with the VNJ algorithm, all deadlock states were not avoided.

The next transition in the development of this research was the process of dynamic reduction. Such a reduction was performed on the e-BRN for every new marking that was entered. The concept behind dynamic reduction was the same as that behind elementary reduction, i.e. to reduce the state space of the net, and thereby reduce the average size of the look-ahead needed in order to avoid a certain proportion of deadlocks. With dynamic reduction, a greater number of deadlock states could be avoided than with elementary reduction alone. However, it still could not avoid all deadlocks. A study of the successful cases of deadlock avoidance revealed a parallel with the principle behind Habermann's algorithm [Habermann 1969].

This research culminated in the HS$^3$ algorithm, which used Habermann's algorithm to guarantee deadlock avoidance, and a heuristic to reduce the conservativeness of Habermann's algorithm when adapted to manufacturing systems. Habermann's algorithm essentially determines whether a process can be assigned its share of resources so that it may continue without interruption to termination. If all such processes can be sequentially cleared, then the state of the system as defined by the set of active processes and the resources allocated to that set is safe. If this is not true, then the system state as defined above is declared unsafe. Since an unsafe state may be falsely declared thus, it makes the algorithm conservative. The structure of the $HS^3$ algorithm is designed to reduce the conservativeness of Habermann's algorithm.

The following discussion is divided into sections corresponding to the evolutionary steps outlined above. A discussion of the elementary reduction (ER) approach is presented first, followed by a presentation of the dynamic reduction algorithm, and finally the $HS^3$ algorithm, which is presented as the solution to the pratical version of the deadlock problem. The elementary reduction approach is not a necessary step in the application of the $HS^3$ algorithm. However, such a reduction has the advantage of resulting in a significantly reduced state space for the given system; hence the frequency of invoking the $HS^3$ algorithm is reduced. Both the BRN models of the test systems were reduced to their e-BRN versions prior to testing the $HS^3$ algorithm over the state spaces (of the e-BRNs). The dynamic reduction approach does not form any part of the $HS^3$ algorithm. The purpose served by presenting the dynamic reduction algorithm in itself is that it provides the reader with a gist of the transition to the $HS^3$ algorithm from the elementary reduction approach, which was the first approach to be developed in this thesis.

## Elementary Reduction of the BRN Model

In the course of a preliminary study of the problem, the author had developed an elementary reduction (ER) approach that is demonstrated for the simple system shown in Figure (8). The system consists of a load-unload station, a single machine station, and an AGV. This system has been used as an example by Viswanadham et al. (1990) for their deadlock detection algorithm. The AGV transports raw parts from the load-unload station to the machine, as well as finished parts from the machine to the load-unload station. There are no buffers in the system. It is assumed that raw parts are always available. Two states of deadlock are evident in this system. In one such state, the AGV is waiting to load the raw part it has onto the machine, which has a finished part on it. The machine in turn is waiting for the AGV to carry off the finished part and load a fresh raw part on it. This represents the circular 'wait-for' state discussed in the literature.

The second deadlock state occurs when the empty AGV is assigned to carry a finished part from the machine, while the empty machine is waiting for the AGV to load a raw part onto it. This again represents a cycle of requests which cannot be fulfilled, i.e. the system is frozen.

The elementary reduction approach uses Petri nets to model the 'wait-for' or blocking relationships between the resources. Thus the scope of the modeling view is restricted to blocking phenomena only. This modeling scope differs from that adopted by Viswanadham et al. (1990), who use generalized stochastic Petri nets to develop a general purpose model of the system. The restricted scope model results in a

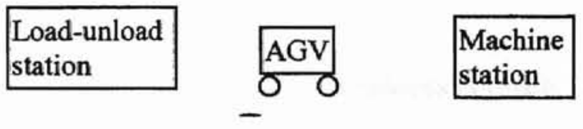


Figure (8). A one-machine-AGV system

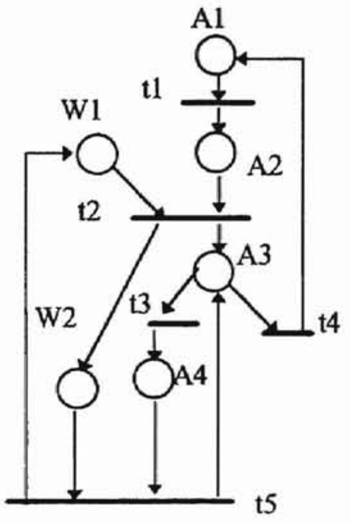


Figure (9). A blocking-restricted net for the system in Figure (8)

EXPLANATION OF PLACES

W1 - machine waiting for AGV to load raw part
W2 - machine waiting for AGV to unload finished part

A1 - AGV waiting for raw part
A2 - AGV waiting at machine to load raw part
A3 - AGV waiting for resource allocation decision
A4 - AGV waiting to unload finished part from machine

simpler net structure; consequently the modeling process is made simpler. The 'wait-for' or blocking-restricted Petri net model is shown in Figure (9).

The next step in the approach is to further reduce the blocking-restricted net model by eliminating those portions of the net which represent fixed, deterministic paths for the flow of tokens.

More formally, let G = (P, T, IN, OUT) be the blocking-restricted Petri net model of the system. Let $G_S = (P_S, T_S, IN_S, OUT_S)$ be a sub-net, $P_S \subseteq P$, $T_S \subseteq T$, such that:

$$\begin{array}{ll} \forall\, t \in T_S & |\,.t\,| = 1 \\ & |\,t.\,| = 1 \\ \forall\, p \in P_S & |\,.p\,| = 1 \\ & |\,p.\,| = 1 \end{array}$$

where:

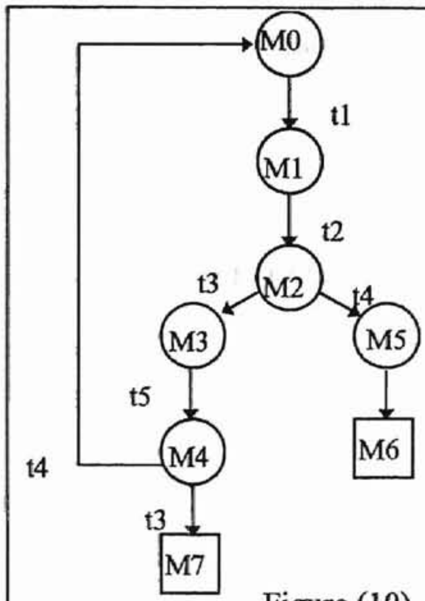
.p is set of input transitions to place p;

p. represents the set of output transitions of place p;

.t is the set of input places to transition t; and,

t. denotes the set of output places from transition t.

The net reduction procedure consists of removing all such sub-nets (see Figure (11)) from the original net. If the Petri net of Figure (9) is reduced in accordance with the method outlined above, then the net in Figure (12) is obtained. This will be referred to as the "controller-net", which will determine those transitions to be fired in order to avoid a deadlock state.



P = {W1, W2, A1, A2, A3, A4}
M0 = (1, 0, 1, 0, 0, 0)
M1 = (1, 0, 0, 1, 0, 0)
M2 = (0, 1, 0, 0, 1, 0)
M3 = (0, 1, 0, 0, 0, 1)
M4 = (1, 0, 0, 0, 1, 0)
M5 = (0, 1, 1, 0, 0, 0)
M6 = (0, 1, 0, 1, 0, 0)*
M7 = (1, 0, 0, 0, 0, 1)*

Figure (10). The reachability graph for Figure (9)

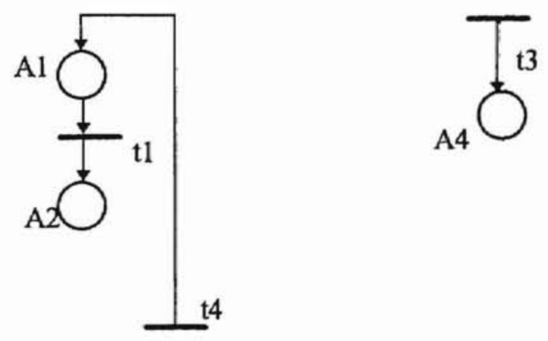
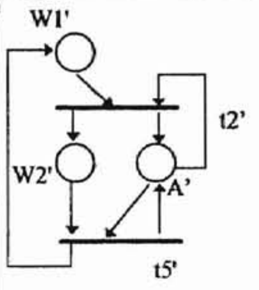


Figure (11). Removal of subnets for reducing the blocking restricted net

In the controller-net, place A' represents the set of places {A1, A2, A3, A4} in the original net. A token in places {A1, A2, A3, A4} in the original net puts a corresponding token in place A' in the controller net. The controller net is decision-free, even though place A' has two out-going arcs to transitions t2' and t5'. This gives the appearance of conflict, but as can be seen from the reachability tree of the controller net in Figure (13), there is no conflict. Furthermore, the net is reinitializable, i.e. there is no terminating state (or deadlock state). It can be seen from Figure (14) that if the controller net is interfaced with the original net model, then it will ensure that no transition is fired in the original net which will lead to a deadlock, thus ensuring deadlock free operation of the system.



A' = {A1, A2, A3, A4, A5}
W1' = {W1}
W2' = {W2}
t2' = {t1, t2, t4}
t5' = {t3, t5}

Figure (12). Controller net for Figure 9 places = {W1', W2', A'}

$$\begin{tikzcd}
M0 = (1, 0, 1) \arrow[d, "t2'"] \\
M1 = (0, 1, 1) \arrow[d, "t5'"] \\
M0 = (1, 0, 1)
\end{tikzcd}$$

Figure (13). Reachability graph for Figure (11)

$$\begin{tikzcd}
M0 \arrow[d, "t1"] \\
M1 \arrow[d, "t2"] \\
M2 \arrow[d, "t3"] \\
M3 \arrow[d, "t5"] \\
M4 \arrow[uuuu, bend left=60, "t4"]
\end{tikzcd}$$

Figure (14). Deadlock free reachability tree for original net interfaced with controller net

To observe how a deadlock state is avoided using the controller net, consider the marking M2 for the original net (1 token in M2 and 1 token in A3). This marking enables two transitions, viz. t3 and t4. If transition t4 is fired, then the net will ultimately reach a deadlock state (corresponding to marking M6 in Figure (10)). Now, the marking M2 puts tokens in places W1' and A' in the controller net, resulting in marking M1' for the controller net. This marking enables transition t5', which represents the set of transitions {t3, t5} of the original net. This means that either transition t3 or t5 should be fired in the original net in order to avoid a deadlock state. Since t5 is not enabled in the original net, it cannot be fired. Thus t3 is fired, and the deadlock state of M6 is avoided. The elementary reduction approach is now described for the BRN of System A.

## Elementary Reduction of the BRN for System A

Let G be the 4 tuple (P, T, IN, OUT) representing a BRN. The process of reducing subnets of the type $G_c$ , where $G_c = (P_c, T_c, IN_c, OUT_c)$; $\forall p \in P_c$, $|.p| = |p.| = 1$; $\forall t \in T_c$, $|.t| = |t.| = 1$; $\forall (p,t) \in (P_c \text{ X } T_c)$, $IN_c = IN$; $OUT_c = OUT$; into macro places and macro transitions is referred to as an elementary reduction. The reduced net is referred to as an e-BRN (elementary Blocking Restricted Net).

There are two types of elementary reductions. A type I reduction involves a subnet consisting of a resource place followed by its operation place. For example, the subnet shown below consists of a resource place P9' and an operation place P12. This subnet is merged to form the macro place P9 = {P9', P12,...} and macro transition T2 =

{T2', T9}. The ellipses indicate that other places are also included in the macro place P9, through the merging of other subnets.

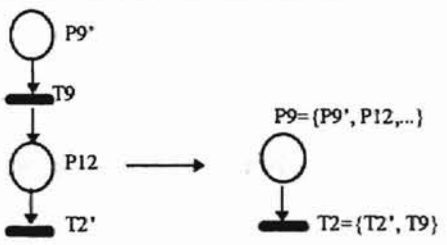


Figure (15): Type I reduction

A type II reduction involves a subnet consisting of an operation place followed by a resource place. For example, the subnet below consists of an operation place P15 and a resource place P9'. This subnet is merged to form the macro place P9 = {P9', P15, ...} and the macro transition T3 = {T3', T12}.

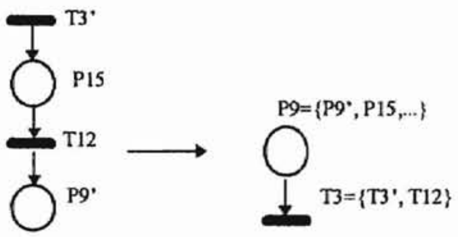


Figure (16): Type II reduction

Since the reduced net is used to control the original net, any disabled macro transition requires that any transition in the original net making up the macro transiton should not be fired, with an exception. The exception is that macro transitions derived from type II reductions are excluded from this rule. Consider a subnet corresponding to a type II reduction. A token in the operation place enables the transition preceding the resource place. Thus the only effect of firing the transition which has the operation place as its input place is to make a resource available. It is intuitive that when the only effect of an action is to increase the availability of a resource, then the resulting state cannot have progressed towards a deadlock. Therefore, although the macro transition resulting from a type II reduction may not be enabled, it is safe to fire the individual transitions comprising that macro transition.

The e-BRN for System A is shown in Figure (17). There are 369 markings in the original BRN, of which 61 are deadlock markings. In the e-BRN, there are 71 markings, of which 9 are deadlock markings. Hence, the elementary reduction algorithm does not avoid all deadlocks. The performance measure defined earlier can be applied

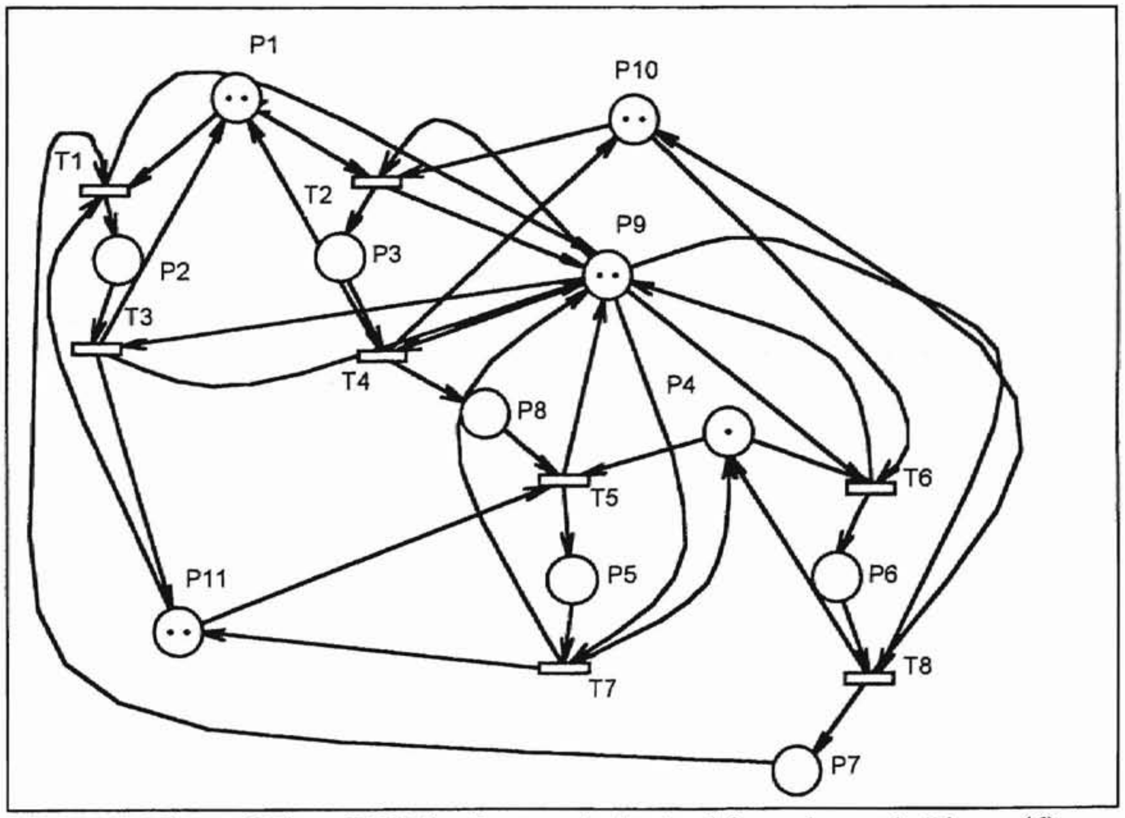


Figure (17): e-BRN for System A obtained from the net in Figure (6)

| RESOURCE PLACE | INTERPRETATION |
|---|---|
| P1 | M1 available |
| P4 | M2 available |
| P9 | AGV available |
| P10 | Fixture A available |
| P11 | Fixture B available |
| OPERATION PLACE | INTERPRETATION |
| P2 | M1 processing semi-finished part on Fix. B |
| P3 | M1 processing raw part on Fix. A |
| P5 | M2 processing semi-finished part on Fix. B |
| P6 | M2 processing raw part on Fix. A |
| P7 | AGV with semi-finished part awaiting M1 |
| P8 | AGV with semi-finished part awaiting M2 |

Table 4: Interpretation of places for e-BRN of System A

for this case. The value is:

$$1 - 9/61 = 0.85$$

i.e. 85% of the deadlock states are avoided. At this point, it would be pertinent to inquire why the reduction algorithm does not avoid all deadlock states. The reason can be found in the motivation behind the algorithm, which derives from the finite look ahead algorithm of Viswanadham et al. (1990). Just as a finite number of look-ahead steps cannot guarantee avoidance of all deadlock states, so also the finite look-ahead built into the reduced net cannot guarantee avoidance of all deadlock states. The advantage of building look-ahead in the net through reduction is that the state space of the reduced net is significantly reduced, hence the number of steps needed to check whether a state is safe (there is at least one sequence of transitions which can be fired from a safe state to initialize the system) will be less on the average. However, the number of steps needed for complete avoidance is still indeterminate.

This implies that look-ahead would still be needed to guide the reduced net away from potential deadlocks. Thus, the reduction algorithm with look-ahead would continue to have an exponential time complexity.

The e-BRN for System B is shown in Figure (18) and can be obtained in a similar manner by performing the elementary reduction on the BRN of System B. The e-BRN model of this system is only used as a model for the other approaches as a test whether the approaches are general enough to handle alternate routings. The interpretation of places in the e-BRN is given in Table 5.

| RESOURCE PLACE | INTERPRETATION |
|---|---|
| P1 | M1 available |
| P4 | AGV available |
| P6 | M2 available |
| P10 | M3 available |
| P11 | M4 available |
| OPERATION PLACE | INTERPRETATION |
| P2 | M1 busy |
| P3 | AGV carrying WIP from M1 |
| P5 | M2 busy |
| P7 | AGV carrying WIP from M2 |
| P8 | M3 busy |
| P9 | M4 busy |
| P12 | AGV moving WIP to M2 |
| P13 | M2 processing operation # 4. |

Table 5. Interpretation of places of e-BRN of System B

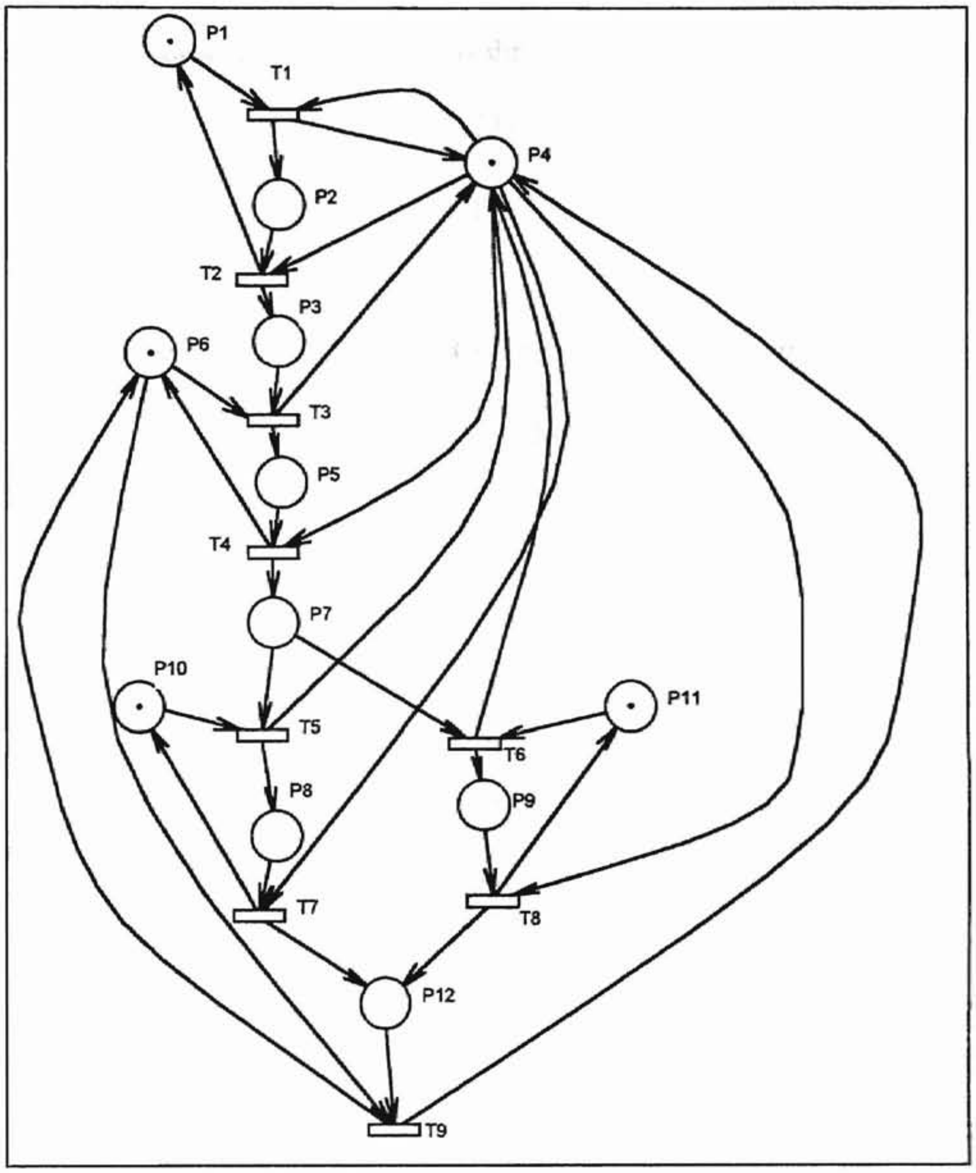


Figure (18). e-BRN of System B

## Dynamic Reduction

Dynamic Reduction of the BRN seemed to be a natural step in the evolution of this thesis. Henceforth, all development is directed at guaranteeing avoidance in the e-BRN. It follows that if any algorithm avoids the deadlock states in the e-BRN, then it would also accomplish the same for the original BRN, when used in conjunction with the elementary reduction algorithm.

In dynamic reduction, a reduced net is obtained every time a new marking is obtained, as opposed to a static elementary reduction. The subnets reduced in dynamic reduction depend on the marking. The dynamic reduction algorithm is presented below and is restricted to systems without alternate routings.

Dynamic Reduction Algorithm

1. Let S denote the current marking corresponding to the actual system state.
2. Let $Po_e$ be the set of empty operation places in S. Delete $p \in Po_e$ . Merge .p with p. Repeat $\forall\ p \in Po_e$ .
3. Fire a transition $t \in T_{de}$ , where $T_{de}$ represents the set of enabled macro transitions in the dynamically reduced net.

Testing of this algorithm showed that it avoided more deadlock states than an elementary reduction, however, it still would not avoid all deadlocks. In the special case where there is only one resource of each type, the dynamic reduction process avoided all deadlocks. However, there appeared to be a similarity in the way the algorithm was avoiding deadlocks with the theory developed by Habermann (1969), which is now discussed in detail.

## Habermann's Algorithm and its Adaptation to Manufacturing Systems

The process of dynamic reduction enables a macro-transition when all the resource places needed for thst macro trsnsition are marked. This reflects the principle behind Habermann's algorithm (also referred to as the H-algorithm). The H-algorithm was developed for computer operating systems; thus the algorithm does not directly apply to manufacturing systems. This section summarizes Habermann's algorithm and discusses the issues related to the adaptation of the algorithm to manufacturing.

The H-algorithm was intended for avoiding deadlocks due to improper resource allocation among processes that satisfy certain assumptions. The following assumptions are stated [Habermann 1969].

1. While a resource A is allocated to process $P_i$ , no other process $P_j$ can seize A .
2. An allocated resource is not released until it has fulfilled its task.

Both the above assumptions require that no resource in use by a process can be pre-empted by another resource. It may be recalled that this assumption corresponds to one of the four necesary conditions for a deadlock to occur [Coffman et al. 1971], i.e. the assumption of no preemption is not strictly necessary for the algorithm to work; however if a system did allow preemption of resources, then a recovery could be easily be effected from a deadlock state. Habermann further states the distinction between the conception of a process as opposed to a resource: A resource works by order of a process P or another resource, while a process does not. This distinction leads to the following definitions: An *independent* process is one 1) which releases all the resources allocated to it upon completion, and 2) whose termination does not require the commencement of another process. A *dependent* process, on the other hand, can be defined as one 1) which releases a subset of the resources allocated to it upon completion, and 2) whose termination requires the commencement of another process. This conceptualization of dependent versus independent processes is specifically relevant to manufacturing systems, as it determines the applicability of the H-algorithm to manufacturing systems. The following terms are defined in context of the algorithm.

1. The resource availability vector **a** states the availability of each type of resource when the system is initialized.
2. The claim vector **b** defines the minimum number of resources of each type that a process will need in order to terminate. The matrix of claim vectors is denoted by **B** and is called the claim matrix.
3. The allocation vector **c** defines the number of resources currently allocated to a process. The matrix of allocation vectors is called the allocation matrix and is denoted by **C**.
4. $\mathbf{r} = \mathbf{a} - \sum \mathbf{C}$ is the free resource vector.

   The following relations are defined:

5. R1: $\forall\, k$ , $\mathbf{b}_k \leq \mathbf{a}$, i.e. no process claims more resources than are initially available in the system.
6. R2: $\mathbf{C} \leq \mathbf{B}$, i.e. no process will try to seize more resources than it has claimed.
7. R3: $\sum \mathbf{C} \leq \mathbf{a}$, i.e. at most all resources are allocated.

8. R4: Let **S** be an ordered set of active processes. The state defined by **S** is safe if and only if: $\forall P_k \in \mathbf{S}, \mathbf{b_k} \le \mathbf{r} + \sum_{S(l) \le S(k)} \mathbf{C}$, where $P_k$ is a process in **S**; S(l) is the rank of the $l^{th}$ process $P_l$; **C** is the allocation matrix; **r** is the free resource vector; and $\mathbf{b_k}$ is the claim vector for $P_k$. Further details are provided in Habermann (1969).

The relations R1, R2, and R3 must hold for a state to be realizable. The relation R4 becomes a condition for a realizable state to be safe, when the set **S** includes all the participating processes in that state. Habermann derived three theorems in the course of arriving at his algorithm. These are reproduced below.

Theorem I

Theorem I states the necessary and sufficient condition for declaring a given state as safe. "When no process will release its resources until it has been allocated all its claimed resources, the process will not get into a deadlock if and only if the allocation state is safe."

Note that there is an explicit assumption in this theorem, which is later discussed in the context of the NP-complete nature of the generalized deadlock problem.

Theorem II

The most significant of the three theorems, theorem II makes possible the *exact* solution of the deadlock problem in polynomial time, given the restrictions mentioned in theorem I. "If the allocation state is safe and a subsequence S fulfills the condition for safeness (R4), then S can be extended into a full safe sequence."

In more logical terms, this theorem can be stated as follows. Let P be the assertion "The state S is safe". Let Q be the assertion "Any subsequence q fulfilling relation R4 can be extended into a full sequence". Then theorem II states that $P \rightarrow Q$. The converse of this theorem is equally important: $\neg Q \rightarrow \neg P$ ("not Q implies not P"). In words, the converse of theorem II states that if a subsequence cannot be extended into a safe sequence, then the given state is unsafe, and there is no need for backtracking. If n is the number of active processes in a given state, theorem II says that in the worst case, we do not have to check all n! possible sequences to find a safe sequence; instead we would need only $n(n+1)/2$ checks. This is a remarkable reduction in the worst case computational complexity.

Although a detailed proof of the theorem is reproduced in the Appendix III, the success of theorem II may be intuitively explained as follows. Whenever an independent process is terminated by allocating its claim to it, then it will release all the resources allocated to it. It is intuitive that if the only effect of an action is to make resources available, then that action could not be possibly constraining the system to move in the direction of a deadlock. It does not matter which process we chose to terminate from a set of active processes, since the only effect of any termination action is to make more resources available.

Note that although a state may be declared safe by an application of theorem II, there can still be transitions that take that state into a deadlock. However, theorem II does say that there is at least one sequence of transitions from the given state that can take the system back to its initial state.

Theorem III

This theorem states that any subsequence satisfying condition R4 and containing the most recently introduced process can be extended into a full safe sequence. "If a safe state is transformed by allocating resources to process $P_k$ and if any subsequence can be found containing $P_k$ and fulfilling condition R4, then the transformed state is also safe."

Having defined the concept of independent vs. dependent processes, we will now proceed to show why it is not possible to apply Habermann's theorems ( II and III ) to dependent processes.

Consider the e-BRN in Figure (17). At a first inspection, there are two ways to define a process:

1) The places P2, P3, P5, P6, P7, P8 may be considered as distinct processes which need certain sets of resources to terminate. For example, the process P3 needs M1, Fixture A, and an AGV to terminate; this is its *claim*. By virtue of being active, it will already possess one unit each of M1 and Fixture A. Thus it would require just one unit of an AGV to terminate. However, the manner in which we have defined P3 as a process makes it a dependent process. This is because the termination of P3 requires the initiation of the process P8. Further, P3 releases only M1 and Fixture A upon termination, while the AGV is assigned to P8.

2) The sets of places $P_1 =$ {P3, P8, P5}, $P_2 =$ {P6, P7, P2} $P_3 =$ {P8, P5}, $P_4 =$ {P7, P2}, $P_5 =$ {P5}, and $P_6 =$ {P2} are defined as processes. The manner in which the above processes are identified corresponds to the definition of independent processes. For example, $P_1$ needs the following resources (one each) to proceed to termination: {M1,

Fixture A, AGV, M2, Fixture B}. It is obvious that upon termination of $P_1$, this set of resources would be available for use. Further, the termination of $P_1$ does not imply the initiation of a subsequent process, since no process follows from $P_1$. Similarly, the definition of all the other processes $P_2, \ldots, P_6$ can be seen to correspond to independent processes. The resource vector **a** and the claim matrix of $P_1, \ldots, P_6$ can be written as:

$\mathbf{a} = [P1\ P4\ P9\ P10\ P11]^T = [2\ 1\ 2\ 2\ 2]^T$

$\mathbf{B} = [\mathbf{b}_1\ \mathbf{b}_2\ \mathbf{b}_3\ \mathbf{b}_4\ \mathbf{b}_5\ \mathbf{b}_6]$

$\mathbf{b}_1 = [1\ 1\ 1\ 1\ 1]^T$

$\mathbf{b}_2 = [1\ 1\ 1\ 1\ 1]^T$

$\mathbf{b}_3 = [0\ 1\ 1\ 0\ 1]^T$

$\mathbf{b}_4 = [1\ 0\ 1\ 0\ 1]^T$

$\mathbf{b}_5 = [0\ 1\ 1\ 0\ 1]^T$

$\mathbf{b}_6 = [1\ 0\ 1\ 0\ 1]^T$

Note: Henceforth, the transpose symbol 'T' will be left out for convenience.

If Habermann's algorithm could be applied to dependent processes, then an exact solution of the above system would be possible in polynomial time (and the class of NP-complete problems solved!). Unfortunately, this is not possible, the reason being that theorem II breaks down when applied to dependent processes. Since theorem III depends on theorem II, the former breaks down as well. We have provided a mathematical explanation showing precisely how theorem II degenerates in Appendix III along with the proof.

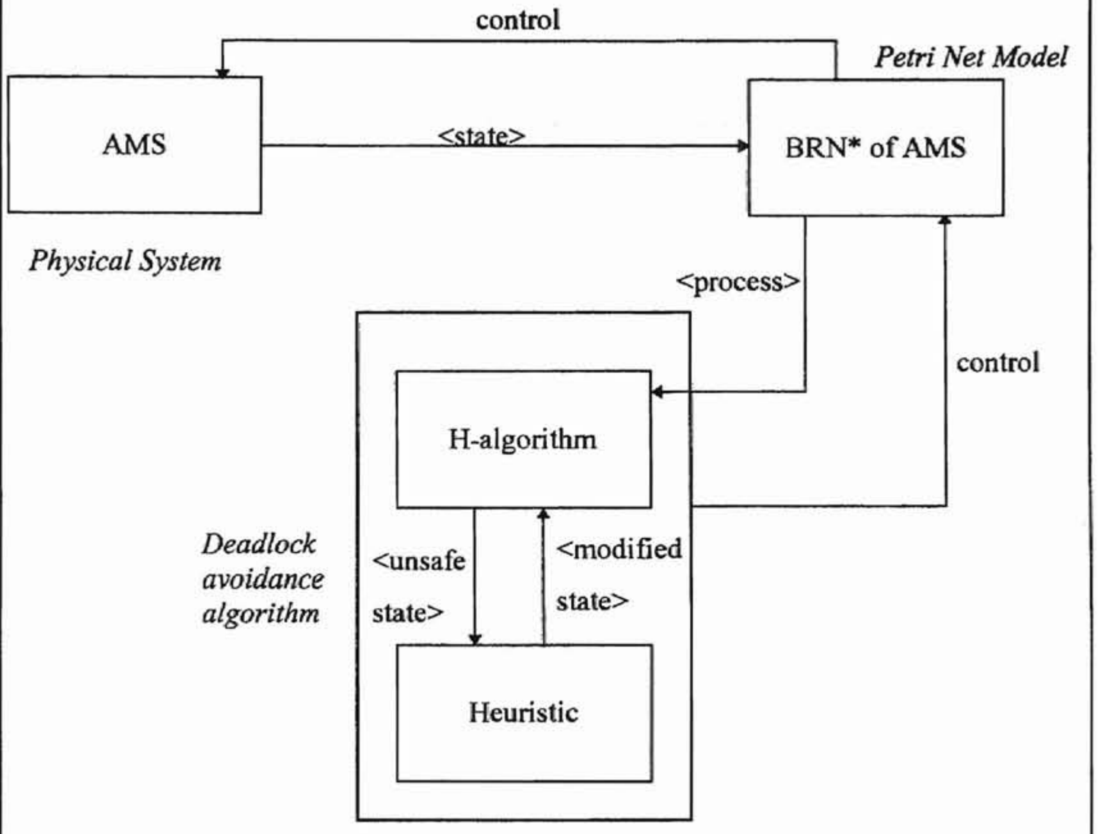
Having set the stage for the application for the H-algorithm in a manufacturing setting, the sub-problems arising from such an application are now discussed. The above definition of an independent process is not in concordance with the restriction in theorem I. This restriction requires that a process not release any of its allocated resources until it has been allocated its full claim of resources. The independent processes defined here do release some of their allocated resources at intermediate steps, i.e. it is not necessary that an independent process be allocated its full claim before releasing some if the resources it holds. For example, $P_1$ releases M1 and Fixture A when it has been allocated an AGV, although its full claim requires that it be allocated M2 and Fixture B in addition to the AGV. The consequence of applying the H-algorithm to independent processes is that the algorithm now becomes conservative, i.e. the solution is not exact, but whenever the algorithm errs, it will always err on the conservative side, i.e. it will declare some safe states as unsafe. Hence, the H-algorithm can still be used to guarantee avoidance of all deadlocks. A metric for conservativeness was defined in the section titled "Performance Measures" (chapter III). The sub-problem is: "given that an algorithm conservatively guarantees that all deadlock states in a system will be avoided, how can the conservativeness be reduced?".

This sub-problem completely transforms the focus of the original problem, which was to guarantee avoidance of deadlocks. With the application of the H-algorithm, the focus is now on reducing its conservativeness when applied to manufacturing systems. It may be noted that in defining the sub-problem, we have used the word "reduced", and not "eliminated". To eliminate the conservativeness (with a polynomial time solution) would be to solve the class of NP-complete problems. Given that a solution to NP-completeness is improbable [Dewdney 1993], the sub-problem is appropriately stated as above.

## The $HS^3$ Algorithm

At this juncture, the algorithmic structure that we have conceptualized for solving the deadlock problem can be depicted as in Figure (19), in the larger context of a deadlock avoidance controller scheme, and in the high level scheme of Figure (20). This structure was based on the discovery that if the system state is allowed to advance after being incorrectly classified as unsafe by the H-algorithm, then it is likely that the advanced state will be correctly recognized as safe. Note that the H-algorithm always correctly declares a state to be safe (SAFE), while it may falsely declare a state as unsafe (labeled as UNSAFE?).

In the figure (19), the elementary Blocking Restricted Net (e-BRN) model of the Automated Manufacturing System determines the control points at which the deadlock avoidance algorithm will be invoked. The avoidance algorithm has two procedures. The first procedure is the H-algorithm, which determines whether a process can be safely introduced in addition to the current set of active processes. The H-algorithm will declare the given state as safe or unsafe. If the state is declared as safe, then the process is allowed. If the state is declared unsafe, then the second procedure, viz. the $S^3$ heuristic is invoked. The objective of the $S^3$ heuristic is to reduce the conservativeness of the avoidance algorithm. It achieves this by advancing the given unsafe state to a future state, which is tested for safeness by the H-algorithm. If the original state was incorrectly classified as unsafe, then it is possible that the H-algorithm will correctly



Note: *BRN = Blocking Restricted Net

Terms in angular brackets represent input variables

Figure (19): Deadlock avoidance control scheme

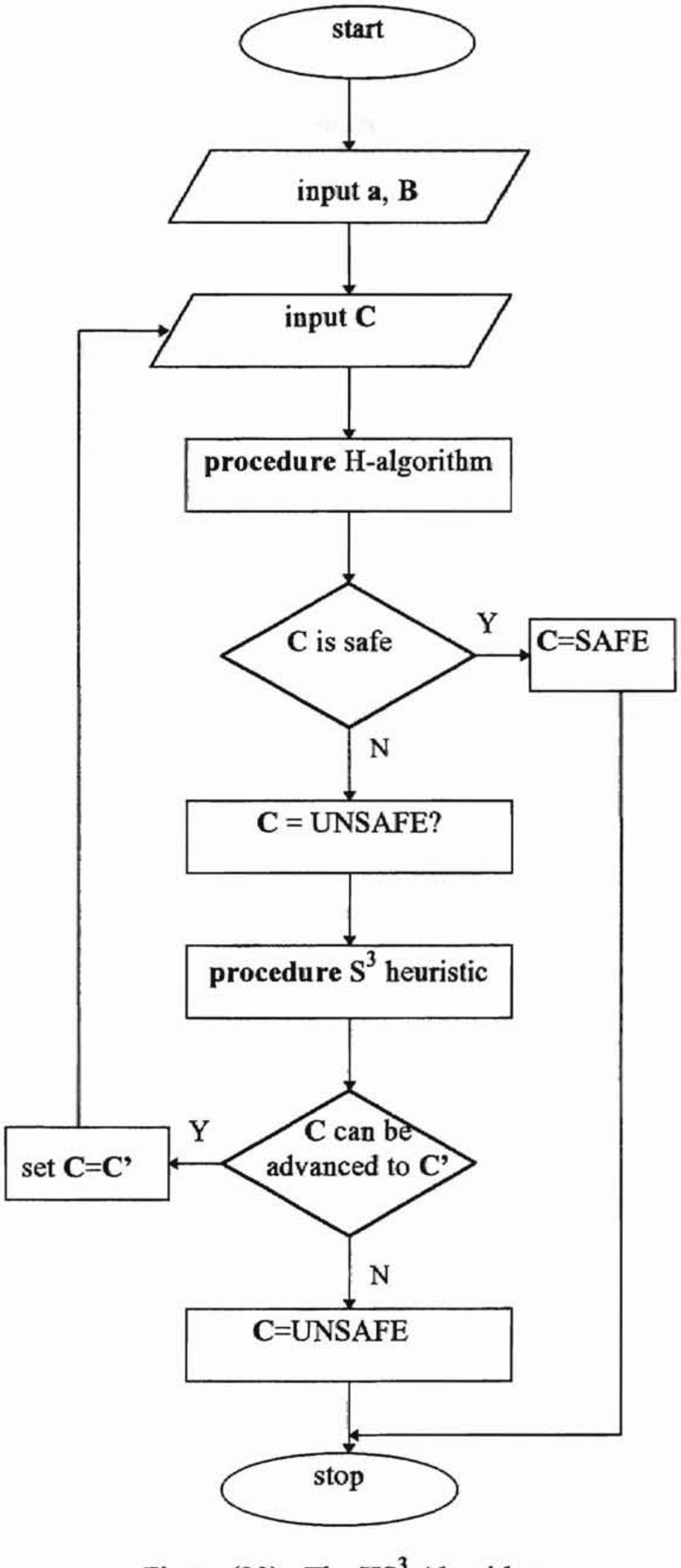


Figure (20): The HS$^3$ Algorithm

classify the derived state as safe. When the state can no longer be advanced by the heuristic, then the state is classified as UNSAFE.

The concept behind the algorithm can be explained with the help of Figure (21). Suppose that Ma is a safe state which is incorrectly classified as unsafe by the H-algorithm. Then the $S^3$ heuristic advances Ma to Mb, where it is again checked by the H-algorithm. It is possible that the H-algorithm now recognizes Mb as a safe state. Then theorem IV (a simple proof and explanation are provided later) says that Ma is also safe. It is also possible that the states Mb, Mc are not recognized as safe. Suppose that the $S^3$ heuristic advances Mc to $M_D$ instead of taking the left path. Then Ma would be incorrectly classified as unsafe, and the heuristic exited. The H-algorithm is presented below in pseudocode, and the structure of the $S^3$ heuristic is later discussed.

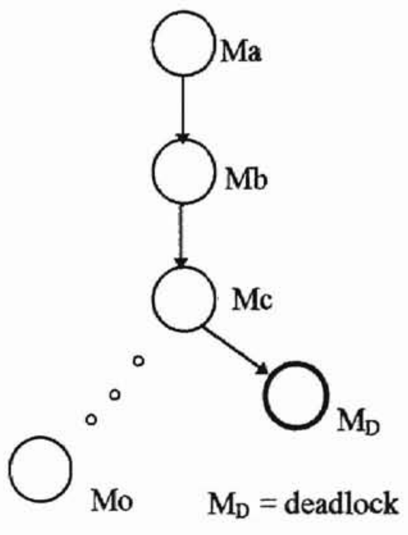


Figure (21): Concept behind heuristic

Pseudocode for Habermann's Algorithm [adapted from Habermann (1969)]

In the algorithm below, $S^*$ represents the complement of S, and $P_k$ is the proposed process.

```
Initialize
    $S = \Phi$.
    $T = S^*$.
while not ( $P_k$ in S or $T = \Phi$ ) do
    begin if $P_k$ in T
        then $P_{candidate} = P_k$
        else $P_{candidate}$ = 1st member of T; $T = T - \{P_{candidate}\}$
            if b[candidate] - c[candidate] $\leq r + \sum_{P_i \in S} c[i]$ then
                begin $S = S \cup \{P_{candidate}\}$ ; $T = S^*$ end
    end
SAFE = $P_k$ in S.
```

## Safe State Seeking Heuristic

The Safe State Seeking Heuristic ($S^3$ heuristic) was developed by this author as an embellishment to the H-algorithm developed by Habermann. The function of the $S^3$ heuristic is to reduce the conservativeness of the H-algorithm. It achieves this by evolving a future state from a given state which is proclaimed unsafe by the H-algorithm.

Thus, the $S^3$ heuristic is executed only when the H-algorithm declares a given state as unsafe. If the new state thus generated is safe, then the given state is also safe.

To achieve its purpose, the $S^3$ heuristic first determines the unavailable resource requirements of the shortest process (measured in terms of remaining resource requirements). Then it attempts to move the load of one of the busy resources needed by the shortest process into an available resource (thereby seizing that resource), preferably one that is not needed by the shortest process. If a transfer is accomplished, then the resultant state is checked for safeness by the H-algorithm. If this state is safe, then it is intuitive that the predecessor state is also safe. A simple proof is provided in theorem IV. If this state is not safe, then the $S^3$ heuristic is executed again. This process is repeated till the $S^3$ heuristic cannot evolve a state any further. The $S^3$ heuristic is written in algorithmic form below.

Algorithm for $S^3$ Heuristic

Note: Use of existing notation is consistent with its use in the section on the H-algorithm.

1. Let $\mathbf{C_g}$ be the given state (as the allocation matrix), where $\mathbf{C_g}$ = UNSAFE? (as declared by the H-algorithm alone). Let $\mathbf{C} = \mathbf{C_g}$.
2. Let {P} be the set of active processes in **C**. Select the shortest active process say $P_s$, in terms of the size of $|\mathbf{b_s} - \mathbf{c_s}|$, where $\mathbf{b_s}$ and $\mathbf{c_s}$ are treated as sets.

3. Let the set of resources needed to finish $P_s$ be $\mathbf{b_s}$. Let the set of free resources be $\mathbf{r}$. Let $\mathbf{u_s}$ be the set of unavailable resources needed by $P_s$. Try to advance some process $P_i$ where $P_i$ satisfies the following: (1) $P_i \neq P_s$, (2) $\mathbf{c_i} \cap \mathbf{u_s} \neq \Phi$, (3) the resource set held by $P_i$ is released upon seizing another resource set $\mathbf{w}$, where (i) $\mathbf{w} \subseteq \mathbf{r}$, and (ii) $\mathbf{w} \cap \mathbf{b_s} = \Phi$. If there is no $\mathbf{w}$ such that $\mathbf{w} \cap \mathbf{b_s} = \Phi$, then only condition (i) need be satisfied. If there is no $P_i$ satisfying the above conditions, then exit the heuristic, setting $\mathbf{C}$ = UNSAFE. If a transformation is achieved, let the new state be $\mathbf{C}$.
4. Check $\mathbf{C}$ for safeness using the H-algorithm.
5. If $\mathbf{C}$ = UNSAFE?, go back to step 2.
6. If $\mathbf{C}$ = SAFE, then $\mathbf{C_g}$ is also safe, i.e. $\mathbf{C_g}$ = SAFE.

Theorem IV

Let M be the current marking (state). If there exists a sequence of enabled transitions (firing vector) $\sigma$ such that $M [ \sigma > M'$, and M' is a safe state, then M is also safe.

Proof: Since M' is safe, it follows from the definition of a safe state that there exists a firing vector $\sigma'$ such that $M' [ \sigma' > M_o$, where $M_o$ is the initial marking corresponding to the initial system state where all resources are available.

Let $\sigma^* = \sigma + \sigma'$; where the operator '+' means "extend the operand $\sigma$ by the argument $\sigma'$ ". Then it follows that:

$M [ \sigma^* > M_o$. Hence M is safe.

## A Demonstration of the $S^3$ Heuristic

The utility of the $S^3$ heuristic is demonstrated in the following situation. Consider two processes represented as a routing of workstations, $P_1 = \{W1, W2, W3, W4\}$ and $P_2 = \{W4, W3, W1\}$. $P_1$ is currently allocated workstation W1, while $P_2$ is currently allocated workstation W4. When this state is checked for safeness by the H-algorithm, it will be classified as unsafe (UNSAFE?). The $S^3$ heuristic is now invoked. The shortest process is $P_2$, since it needs two resources to finish, while $P_1$ needs three. In keeping with the notation used in the heuristic, $P_s = P_2$; $P_i = P_1$; $\mathbf{r} = \{W2, W3\}$; $\mathbf{u_s} = \{W1\}$; $\mathbf{b_s} = \{W1, W3, W4\}$; $\mathbf{c_s} = \{W4\}$; $\mathbf{b_i} = \{W1, W2, W3, W4\}$; $\mathbf{c_i} = \{W1\}$. The current state is $[\mathbf{c_1}, \mathbf{c_2}] = [\mathbf{c_i}, \mathbf{c_s}]$. In accordance with the heuristic, $P_1$ is selected to be advanced ($P_i \neq P_s$; $\mathbf{c_i} \cap \mathbf{u_s} \neq \Phi$). To advance $P_1$, the resource set $\mathbf{w} = \{W2\}$ needs to be acquired. The set $\mathbf{w}$ satisfies the condition $\mathbf{w} \cap \mathbf{b_s} = \Phi$ in this case. After advancement, the transformed state is $P_1' = \{W2, W3, W4\}$; $\mathbf{c_1}' = \{W2\}$; $\mathbf{c_2} = \{W4\}$. The new state defined by $[\mathbf{c_1}', \mathbf{c_2}]$ is correctly detected as a safe state. Hence the original state $[\mathbf{c_1}, \mathbf{c_2}]$ is also declared safe.

## Application of the $HS^3$ Algorithm to System A

The $HS^3$ algorithm successfully avoids *all* deadlock states in the state space of the e-BRN model of System A in Figure (17). Further, it does so with *zero* conservativeness, i.e. only true unsafe states of the e-BRN are avoided. Since the e-BRN can be controlled to avoid all deadlocks, it can be interfaced with the original BRN to avoid all deadlocks in the larger state space of the original BRN, i.e. the physical system can run indefinitely (as long as any resources that fail in the course of a processing operation comes up in finite time).

That the $HS^3$ algorithm avoids all deadlock states is to be expected, since it is based on the H-algorithm, which in turn ensures that the necessary and sufficient condition for safeness is always satisfied. With regards to the conservativeness metric, the $HS^3$ algorithm could not have performed better (mathematically and literally speaking). Later, the $HS^3$ algorithm is applied to System B, which includes alternate routings. For System B, the $HS^3$ algorithm again performs with zero conservativeness (of course, all true unsafe states are avoided). The application of the $HS^3$ algorithm will be demonstrated below for two states, a safe state and an unsafe one.

Consider marking M46 in Table A1 (Appendix II). The fact that this state is active implies that it was found to be safe by $HS^3$. Suppose that we want to verify whether the state M47 is safe, where M47 is derived from M46 by firing transition T6. The marking of M47 = [0 2 0 0 0 1 1 0 1 1 0], where all places are ordered according as P1, $P_2$, ..., P11. There are 4 active processes in M47, viz. $P_6$, $P_6$, $P_4$, and $P_2$ (note: two of these processes are identically defined). These processes were earlier defined in the section "Habermann's Algorithm and its Adaptation to Manufacturing Systems"). The claim matrices, allocation matrices, and free resource vector are:

$\mathbf{b}_6 = [1\ 0\ 1\ 0\ 1]$

$\mathbf{b_4} = [1\ 0\ 1\ 0\ 1]$

$\mathbf{b_2} = [1\ 1\ 1\ 1\ 1]$

$\mathbf{C} = [\mathbf{c_6}\ \mathbf{c_6}\ \mathbf{c_4}\ \mathbf{c_2}]$

$\mathbf{c_6} = [1\ 0\ 0\ 0\ 1]$

$\mathbf{c_4} = [0\ 0\ 1\ 0\ 0]$

$\mathbf{c_2} = [0\ 1\ 0\ 1\ 0]$

$\mathbf{r} = \mathbf{a} - \sum\mathbf{C} = [2\ 1\ 2\ 2\ 2] - [2\ 1\ 1\ 1\ 2] = [0\ 0\ 1\ 1\ 0]$

According to the H-algorithm procedure in $HS^3$, the most recently introduced process is first checked to see whether it can be terminated. The most recently introduced process in this case is $P_2$, by virtue of firing T6 from M46. It can be seen that relation R4 is not satisfied, i.e. $\mathbf{b_2} - \mathbf{c_2} = [1\ 0\ 1\ 0\ 1]$ is not less than or equal to $\mathbf{r}$ (note: this does not necessarily mean that $\mathbf{b_2} - \mathbf{c_2} > \mathbf{r}$). The next step is to check whether some other process can satisfy R4. It can be seen that $P_6$ satisfies R4, and can be deallocated. The new value of r is:

$\mathbf{r'} = [1\ 0\ 1\ 1\ 1]$

Now, the relation R4 holds for $P_2$, and the state is SAFE. Note that there is no need to check whether the other two remaining processes $P_6$ and $P_4$ can be terminated; this is by virtue of Habermann's third theorem.

Consider the second state M70 = [0 2 0 1 0 0 2 0 0 2 0]. This state is reached from M47 by firing T8. There are 4 active processes in this state, viz. $P_6$, $P_6$, $P_4$, $P_4$. The claim and allocation vectors for these processes are given above. The value of the free resource vector is recomputed

$\mathbf{r} = \mathbf{a} - \sum\mathbf{C} = [2\ 1\ 2\ 2\ 2] - [2\ 0\ 2\ 0\ 2] = [0\ 1\ 0\ 2\ 0]$

It can be seen that no process satisfies R4. Hence the state M70 is labeled UNSAFE?. This invokes the $S^3$ heuristic. The shortest process is $P_6$ in terms of the size of its remaining resource requirements (it needs only the AGV to terminate). Hence, $S^3$ will search for a process which can release an AGV (viz. place P9). The resources P9 are held by the two processes $P_4$, $P_4$. Neither of these can release the AGV (P9) till they obtain M1 (P1) and fixture B (P11). Since $S^3$ is unsuccessful in releasing P9, the state is labeled UNSAFE by $HS^3$. Indeed, M70 is a deadlock marking as can be seen from the marking set in Table A2 in Appendix II.

In the following section, a systematic method of computing the claim vector **b** for a process is described.

## A Procedure for Obtaining the Minimal Requirements for a Process

A systematic way of obtaining the minimal resource requirements for an independent process is developed in this section. Initially, only processes without alternate routings are considered. Later, the procedure is extended to include processes with alternate routings.

Processes without Alternate Routings

A blocking restricted net of a process without alternate routings is a marked graph, the transitions of which form a unique T-invariant [Desrochers and Al-Jaar 1995] corresponding to that process. When the T-invariant is ordered according to the appearance of its transitions in the elementary path, the T-invariant is referred to as an ordered T-invariant. The procedure developed here is first described with the help of an example. Consider the independent process $P_1$ = {P3, P8, P5} in the e-BRN in Figure (17). The elementary path describing $P_1$ is {T2, P3, T4, P8, T5, P5, T7}. The ordered T-invariant corresponding to $P_1$ is T2T4T5T7. In the example, the minimal requirement of the AGV (represented by P9) in $P_1$ is computed.

| T-invariant | P9 (2) |
|---|---|
| T2 | -1 + 1 (1, 2) |
| T4 | -1 (1) |
| T5 | +1 (2) |
| T7 | -1 + 1 (1, 2) |

In the table above, the first column contains the transitions of the ordered T-invariant. In the second column, the numbers in brackets represent the number of resources available after firing the transition in that row. If a transition is associated with a self loop, then there will be two numbers in the brackets; the first represents the quantity available upon firing the transition and ignoring the output arc to the resource place; the second quantity represents the quantity available upon firing the transition with all arcs preserved. The signed numbers represent the change in availability of the resource. The minimal resource requirement, denoted as $\min(P_i, R_k)$, where $R_k$ is the $k^{th}$ resource type, is:

$\min(P_1, P9) = |\min(\text{change vector})| = |\min(-1, 1, -1, 1, -1, 1)| = 1$.

Let the ordered T-invariant of P be written as $T = u_1u_2, ..., u_{l(p)}$, where $u_i$ represents the ith transition in T, and l(p) is the length of the T-invariant. Let $A^-$ be the output incidence matrix corresponding to the ordered T-invariant of an independent process P and a given resource place. Let $A^+$ represent the input incidence matrix corresponding to the ordered T-invariant of an independent process P and the given resource place. The pseudocode for obtaining the minimal resource requirement is given below.

Algorithm for Computing min(P, R)

In the algorithm below, $T = u_1, u_2, ..., u_{l(p)}$ is the ordered T-invariant, and the $1 \times l(p)$ row unit vector $e_i$ corresponding to transition $u_i$ is defined as:

$e_i = [z_1, z_2, ..., z_{l(p)}]$; $z_j = 0$ for $j \neq i$ ; $z_j = 1$ for $j = i$

```
Input: P, T, R (set of system resources types)
Initialize: $j = 1$
**do while** $(R \neq \Phi)$
        $A^-[j] = A^-$ ( T-invariant of P X resource place representing $R_j$ )
        $A^+[j] = A^+$ ( T-invariant of P X resource place representing $R_j$ )
        Initialize: $c_j = x = 0; i = 1$
        **do while** $(i \leq l(p))$
          **begin**   $c_j = c_j + e_i . A^-[j]$
                **If** $c_j < x$ **then** $x = c_j$
                $c_j = c_j + e_i . A^+[j]$
                $i = i + 1$
          **end**
        $\min(P, R_j) = |x|$
        $R = R - \{R_j\}$
        $j = j + 1$
**end**
```

The vector min(P, R) is the claim vector for P, i.e. for process $P_k$ , we can write $b_k$ = min($P_k$ , R).

## Incorporating Processes with Alternate Routings into the Claim Matrix

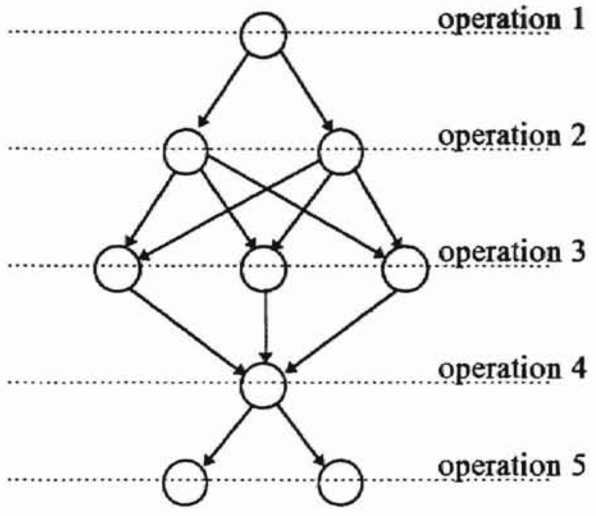


Figure (22): A process plan with alternate routes represented as an OR digraph

Alternate routings in a process may be represented by the OR digraph in Figure (22) [Wysk 1995]. Note that the digrapgh is strictly OR; an AND digraph would represent alternate process plans [Wysk 1995], i.e. alternate processes. The OR digraph is not an integral part of the approaches developed below; rather, it is used as a scheme to clearly present the concept of a process with alternate routes. We can see that the process represented in Figure (22) above can take 1.2.3.1.2 = 12 different routes.

Let P be a process defined as the ordered set of operations in the e-BRN of the given system, i.e. $P = \{p_1, p_2, ..., p_{l(p)}\}$, where $p_{l(p)}$ represents the last operation in P, and $l(p)$ is the number of distinct operations needed to produce the end product. Hence $p_i$ can be considered as a set of one or more operation places in the e-BRN of P. Each $p_i$ can be mapped onto one or more operation places in the e-BRN. If $p_i$ can be mapped on to two or more operation places, then it is interpreted as an operation with alternate sets of concurrently needed resources available for its processing, and represents alternate routes in P. Let $P_a = \{p_i\}$ be such an ordered set of operations (with alternate sets of concurrently needed resources available for its processing) in P. The distinction between an operation (e.g. a drilling operation) and an operation place representing that operation will be maintained in the exposition below.

Define a function $f : O \rightarrow P$, where O is the set of operation places that define P in its e-BRN. Define the set $U = \{ o_i \mid f(o_i) \neq f(o_j) ; i \neq j \}$. Define the set $V = O - U$. The function f maps every operation place in O to an operation p in P. The set U is the set of operation places corresponding to operations in P without alternate resources available for processing that operation. Thus if the domain of f is restricted to $\{o \mid o \in U \}$, then f is a one-to-one function, since every operation place in U can be mapped on to a unique operation in P, and vice-versa. The set V is the set of operation places corresponding to operations in P with alternate resources available for processing that operation. When the domain of f is restricted to the set $\{ o \mid o \in V \}$, then f is not a one-to-one function. For example, refer to the e-BRN of System B in Figure (18). There is a single process with two alternate routes. The process can be defined as $P = \{p_1, p_2, p_3, p_4, p_5, p_6\}$. The set of operation places that define P in the e-BRN shown is O = {P2, P3, P5, P7, P8, P9, P12}. We have $f(P8) = f(P9) = p_5$. The set U = {P2, P3, P5, P7, P12}. The set V = {P8, P9}.

Further, define $z_k = \{ o_i \mid f(o_i) = f(o_j) = p_l ; i \neq j , p_l \in P_a \}$, where $k = s(p_l)$, and $s(p_l)$ is the rank of $p_l$ in $P_a$. $z_k$ defines the set of operation places that map on to a single operation p in $P_a$. It can be seen that $V = \cup z_k$ over all k. In the above example, $z_1 = \{P8, P9\}$.

Having defined the terms above, we will proceed to develop two approaches to incorporating processes with alternate routes into the claim matrix. The first approach does not impose any further restrictions on the system being modeled. The second approach is more refined, however it requires a further restriction on the system types allowed.

Approach I

This approach computes one claim vector for each route that a process can take in the course of its exeution. For the OR digraph shown in Figure (22), there are 12 claim vectors for the process represented by the digraph. For the process modeled in Figure (18), there are 2 claim vectors. In general, a claim vector must be computed for each route that a process can take. For a process P, let $P_a$ be defined as above. Let $v = |P_a|$. Let c be the number of claim vectors for P. Then c = number of routes for P = $|z_1| \times |z_2| .... |z_V|$ . The method of obtaining the claim vector corresponding to a route of P is the same as obtaining the claim vector by treating that route as the only route for P. Let $b_k = [ b_k^1,..., b_k^c ]$ be the claim sub-matrix thus obtained for P. Then the H-algorithm is modified as follows.

Pseudocode for the H-Algorithm, Extended to Processes with Alternate Routings.

```
Step 1.

Input a, B, C, $P_k$ (most recently introduced process)

Step 2.

Initialize

    S = $\Phi$.

    T = S *.  // S* is the complement of S

Step 3.

while not ( $P_k$ in S or T = $\Phi$ ) do

    begin if $P_k$ in T

        then $P_{candidate} = P_k$

        else $P_{candidate}$ = 1st member of T; T  = T - $\{P_{candidate}\}$

        j = 1

            while (j $\leq$ c) do // c is the no. of routes that $P_k$ can take, as computed earlier

            b[candidate] = $b^j$ [candidate]

            begin if    b[candidate] - c[candidate] $\leq$ r + $\sum_{P_i \in S}$ c[i]

                then begin S = S $\cup$ $\{P_{candidate}\}$ ; T = S* ; j = c + 1 end // set j = c+1 to exit

                else j = j + 1

            end

    end

SAFE = $P_k$ in S.  UNSAFE? = $P_k$ not in S.
```

Approach II

The concept behind this approach is to define an equivalent (hypothetical) resource as a logical (as in the use of logic operators OR, AND) combination of the alternate resource sets that are available for performing an operation. The claim vector thus computed is refered to as an equivalent claim vector. The size of the claim vector is increased by the number of equivalent resources defined. This approach results in having to define a single claim vector for a process with alternate routings, as against 'c' different claim vectors as defined in approach I.

For a process P with alternate routings, let O, U and V be sets as defined above. Let R(O), R(U) and R(V) be the set of resource places needed to initialize the operation places in O, U and V respectively. In this discussion, to "initialize" an operation place means to begin the operation corresponding to that operation place. In the Petri net model, this would be translated as the placement of a token into the operation place. In order to develop this approach, the following restriction is needed: the maximum claim by P of a resource type belonging to the set R(V) is limited to one unit. There is no restriction on the set of distinct resource types that a process can concurrently claim, as long as the maximum claim for each resource type belonging to R(V) is one unit. Many realistic AMSs can satisfy this restriction.

Consider a process P with alternate routings and confirming to the restriction above. Let the set $z_i$ be defined as above. $z_i$ represents an operation in P which can be performed on $|z_i|$ alternate resource sets. We can write $z_i = \{z_i^1, \ldots, z_i^{|z_i|}\}$, where $z_i^k$

represents the kth operation place in $z_i$. Define $R(z_i^k)$ as the set of resources needed to initialize $z_i^k$. The procedure for obtaining the claim matrix is as described below.

Procedure For Obtaining The Equivalent Claim Vector

The minimal claim for a resource r by process P can be written as min(P, $r \in R \cup R_{eq}$), where R is the set of system resources, $R_{eq}$ is the set of equivalent resources defined as follows:

1. $\min(P, r) = 0 \; \forall \, r \in R - R(O)$
2. $\min(P, r) = \min(P, r)$ as computed using algorithm for processes without alternate routings, for any route of P; $\forall \, r \in R(U)$
3. $\min(P, r) = 0 \; \forall \, r \in R(V) - \{R(U) \cap R(V)\}$
4. $\min(p, r_{eq}) = 1 \; \forall \, r_{eq} \in R_{eq}$, where $R_{eq} = \{ R_{eq}^1, \ldots, R_{eq}^v\}$, where v is the number of operations in P with alternate resource sets to choose from, and $R_{eq}$ is defined below.

$R_{eq}^i(P) = \underset{z_i}{\mathrm{OR}}[\underset{R(z_i^k)}{\mathrm{AND}} R(z_i^k)]$, where the operators OR and AND are logical operators with their conventional definitions. As the above expression shows, the AND operator is applied over the set $R(z_i^k)$, and represents the fact that for the ith alternative, the operation place must have one unit of each type of resource contained in the set $R(z_i^k)$. The OR operator is applied over the set $z_i$, and represents the choice between the sets $\{R(z_i^k)\}$ to perform the operation. For example, consider the net in Figure (18). Here, $z_1 = \{P8, P9\}$, i.e. $z_1^1 = P8$ and $z_1^2 = P9$. $R(z_1^1) = \{P10\}$, $R(z_1^2) = \{P11\}$. Then $R_{eq}^1(P) =$

P10 OR P11. Thus $R_{eq}^{1}$ is TRUE (boolean value of 1) when P10 OR P11 is TRUE, i.e. when there is at least one unit of P10/P11 available (or both P10 and P11 available). In this example, the AND operator is not used, since both $|R(z_1^{1})| = 1$ and $|R(z_1^{2})| = 1$. To see the use of the AND operator, suppose that an additional resource type represented by a resource place P14 is needed for transition T5 (refer Figure (18)) to fire. Then $R(z_1^{1}) = \{P10, P14\}$, and $R_{eq}^{1}(P)$ = (P10 AND P14) OR (P11). In words, $R_{eq}^{1}(P) = 1$ when P10 and P11 are both available, or when P11 is available, or all three resource types are available. However, if only P10 is available, or if only P14 is available, the expression does not evaluate to 1, i.e. the claim $\min(P, R_{eq}^{1}(P))$ is not satisfied.

In determining the allocation vector **c** for a process, all resources that are physically allocated to the process are entered as usual into the vector. All equivalent resources that hold true (i.e. evaluate to 1) for the process are also entered as such into the allocation vector for the process.

In determining the **r**, free resource vector, we first compute the free resource vector ignoring the set of equivalent resources; let this partial vector be **r'**. Next, using the partially computed free resource vector, we obtain the free equivalent resource vector, say $\mathbf{r}_{eq}$ by evaluating the expressions for the equivalent resources. The combination of **r'** and $\mathbf{r}_{eq}$ gives the free resource vector.

## Application Of $HS^3$ To System B

System B is represented by its e-BRN in Figure (18). A single process plan with two alternate routes is modeled. The $HS^3$ algorithm will be applied to this system using the second approach developed for process plans with alternate routings to compute the claim matrix. The sets O, U, V, R, R(O), R(U), R(V) are listed below.

O = {P2, P3, P5, P7, P8, P9, P12}

U = {P2, P3, P5, P7, P12}

V = {P8, P9}

$z_1$ = {P8, P9}

R = {P1, P4, P6, P10, P11}

R(O) = R

R(U) = {P1, P4, P6}

R(V) = {P10, P11}

R(U) - R(V) = {P1, P4, P6}; min(P, P1) = min(P, P4) = min(P, P6) = 1

Since there is only one operation with alternate resource sets to choose from, there is correspondingly only one equivalent resource defined:

$R_{eq}^1$ = P10 OR P11.

The above expression for $R_{eq}^1$ is derived in the discussion on approach II for processes wiith alternte routings. The minimal requirement for all equivalent resources is one unit, i.e. min(P, $R_{eq}^1$) = 1.

Consider marking M12 for this example. Let the set of places be ordered as (P1, P2, ..., P12). The marking M12 = [0 1 0 0 0 1 1 1 0 0 1 0], i.e. there is a token in each of the places P2, P6, P7, P8, and P11.

At this state, three active independent processes can be defined. The three processes can be defined by their header operation places, viz. P2, P8, and P7, and will be referred to as $P_1$, $P_2$, and $P_3$. The most recently introduced process was that headed by place P7. This can be seen from the reachability graph in Figure (23), where M10 [ T4 > M12, and the firing of T4 initializes P7, which implies the initiation of $P_3$. In applying the $HS^3$ algorithm to see whether the allocation state C is safe, what is checked for is whether the claim of the process $P_3$ can be satisfied.

The resource vector **a** and the claim vector **B** are shown below. The allocation vector **C** and free resource vector **r** for marking M12 are also shown.

$\mathbf{a} = [P1, P4, P6, P10, P11, R_{eq}^{1}] = [1\ 1\ 1\ 1\ 1\ 1]$.

$\mathbf{B} = [\mathbf{b_1}\ \mathbf{b_2}\ \mathbf{b_3}]$

$\mathbf{b_1} = [1\ 1\ 1\ 0\ 0\ 1]$

$\mathbf{b_2} = [0\ 1\ 1\ 0\ 0\ 1]$

$\mathbf{b_3} = [0\ 1\ 1\ 0\ 0\ 1]$

$\mathbf{C} = [\mathbf{c_1}\ \mathbf{c_2}\ \mathbf{c_3}]$

$\mathbf{c_1} = [1\ 0\ 0\ 0\ 0\ 0]$

$\mathbf{c_2} = [0\ 0\ 0\ 1\ 0\ 1]$

$\mathbf{c_3} = [0\ 1\ 0\ 0\ 0\ 1]$

$\mathbf{r} = \mathbf{a} - \sum \mathbf{C}$, where the summation over **C** is a vector summation,

$= \mathbf{r}' + \mathbf{r}_{eq}$; where r' is the free resource vector for the physical resources, and $\mathbf{r}_{eq}$ is the free resource vector for the equivalent hypothetical resources.

$\mathbf{r}' = [1\ 1\ 1\ 1\ 1\ ] - [1\ 1\ 0\ 1\ 0\ ]$

$= [0\ 0\ 1\ 0\ 1\ ]$

$\mathbf{r}_{eq} = R_{eq}{}^{1}$ = P10 OR P11 = 1 (since P11 is available)

$\mathbf{r} = [0\ 0\ 1\ 0\ 1\ 1]$

According to the HS$^3$ algorithm, we first use the H-algorithm to label a state as SAFE or UNSAFE?, and if the label is UNSAFE?, then the S$^3$ heuristic is applied to modify the allocation state. The cycle is then repeated till the state is either declared SAFE or no further modification is possible, when the state is labeled UNSAFE.

In the H-algorithm procedure, the candidate process is $P_3$ (most recently introduced process):

**b**[candidate] = $\mathbf{b}_3$; **c**[candidate] = $\mathbf{c}_3$.

$\mathbf{b}_3 - \mathbf{c}_3 = [0\ 1\ 1\ 0\ 0\ 1] - [0\ 1\ 0\ 0\ 0\ 0] = [0\ 0\ 1\ 0\ 0\ 1] \leq [0\ 0\ 1\ 0\ 1\ 1]$

Since the condition **b**[candidate] - **c**[candidate] $\leq$ **r** is true, the allocation state is SAFE.

The HS$^3$ algorithm will now be tested on marking M14, which is unsafe, i.e. every path from M14 leads to a deadlock marking. Using a straightforward look-ahead approach, the look-ahead parameter would have to be a minimum of 4 to avoid M14, and several states would have to be checked before concluding that M14 is an unsafe state.

The marking M14 = [1 0 1 0 0 1 0 1 1 0 0 0]. There are three active processes viz. $P_1$, $P_2$, $P_3$ in this marking, and are defined by their header operation places P3, P8, and P9. The resource vector was defined above. The claim vectors for these processes are:

$\mathbf{b}_1$ = [0 1 1 1 1 1]

$\mathbf{b}_2$ = [0 1 1 0 0 1]

$\mathbf{b}_3$ = [0 1 1 0 0 1]

The allocation vectors are:

$\mathbf{c}_1$ = [0 1 0 0 0 0]

$\mathbf{c}_2$ = [0 0 0 1 0 1]

$\mathbf{c}_3$ = [0 0 0 0 1 1]

The free resource vector is:

$\mathbf{r}$ = [1 0 1 0 0 0] (as computed from $\mathbf{r}' + \mathbf{r}_{eq}$)

It can be seen that condition R4 is not satisfied in the H-algorithm, hence the state is labeled UNSAFE?. This invokes the $S^3$ heuristic, which selects $P_2$ as the shortest process, and in accordance with the heuristic, modifies the allocation state $\mathbf{c}_1$ as follows:

$\mathbf{c}_1'$ = [0 0 1 0 0 0]

$\mathbf{b}_1'$ = [0 1 1 1 1 1]

$\mathbf{r}$ = [1 1 0 0 0 0]

The modification is actually a transformation of one independent process into another shorter independent process. The new allocation state is again UNSAFE?. The $S^3$ heuristic modifies $c_1'$ to $c_1''$ as follows:

$\mathbf{c_1''} = [0\ 1\ 0\ 0\ 0\ 0]$

$\mathbf{b_1''} = [0\ 1\ 1\ 1\ 1\ 1]$

$\mathbf{r} = [1\ 0\ 1\ 0\ 0\ 0]$

The new allocation state is UNSAFE?. At this point, the $S^3$ heuristic cannot modify the allocation state any further, hence the state is UNSAFE.

*Application of the $HS^3$ algorithm to this system results in the avoidance of all deadlock states, with zero conservativeness, i.e. only true unsafe states are avoided.*

# CHAPTER V

# CONTRIBUTIONS, FUTURE RESEARCH, AND SUMMARY

## Research Contributions

The primary contribution of this research is the development of an algorithm that (1) meets the criteria stated in the practical version of the problem (the statement can be found in the introduction), and (2) can be applied to real world systems. As the first statement implies, the $HS^3$ algorithm (1) guarantees deadlock avoidance, (2) has an acceptably low conservativeness (for the systems tested, the conservativeness metric was zero), (3) has a low order polynomial complexity, of the order of $O(n(n+1)/2)$, and (4) does not require the reservation of resources for the purpose of avoiding deadlocks.

The processes that can be modeled include: concurrently active processes, alternate process plans, alternate routings, processes holding multiple units of one or more resource types (including finite buffers), resources that are revisited, and assembly operations. The broad spectrum of processes that can be dealt with is a consequence of the fundamental manner in which the deadlock problem is approached. Failures are permitted on the assumption that a machine comes up in finite time. The $HS^3$ algorithm was not designed for a restricted class of systems; rather, it evolved by conceptualizing the notion of an independent process, and adapting Habermann's algorithm to the formalized scheme of that concept. The formalism used to represent the concept of an independent process was a Blocking Restricted Petri net model of the physical and control elements that define the automated manufacturing system. An elementary reduction can be performed on the net in order to reduce the frequency of checking whether a given system state is safe or unsafe. However, it is not necessary that the BRN be reduced to the e-BRN in order to apply the $HS^3$ algorithm. This author chose to apply the $HS^3$ algorithm to the e-BRN simply because the concept of reduction was an integral part of his early work on the deadlock problem; therefore it was natural to continue the development of the research from that point on.

Table 6 summarizes the contributions of this research. The other research that is selected for this comparison is restricted to the area of avoidance, and does not include detection and recovery or prevention strategies. The $HS^3$ algorithm satisfies all nine criteria. The reason is that no assumptions were made regarding the type of system that could be handled by the algorithm, with the exception of the four necessary and sufficient conditions for deadlock (without which the system would not experience deadlocks), and the assumption that any machine that failed would come up in finite time. The $HS^3$ algorithm is based on Habermann's algorithm which guarantees avoidance of deadlocks in polynomial time. The $S^3$ heuristic was designed for reducing the conservativeness of Habermann's algorithm. The conservativeness for the systems tested was zero for both cases. Alternate process plans, which are modeled in System A, are a special case of concurrent processes, and hence can be handled without specially adapting Habermann's algorithm for these cases. Habermann's algorithm was adapted for alternate routings, and using this adaptation the $HS^3$ algorithm was successfully tested on System B, which modeled a process plan with two alternate routes. No restrictions were made regarding the number of resources that a process could concurrently seize.

The legend for the criteria used in this table is given below.

A. Guarantees avoidance of deadlocks

B. Acceptable conservativeness

C. Polynomial time complexity (worst case)

D. Concurrent processes

E. Alternate process plans

F. Multiple resources can be allocated to a process

G. Alternate routings

H. Assembly operations

I. All resources in system can be shared

| Algorithm | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| Viswanadham's finite lookahead (1990) | N | * | N | Y | Y | Y | Y | Y | Y |
| Banaszak & Krogh's algorithm (1990) | Y | N | Y | Y | Y | N | N | N | N |
| Heish & Chang's algorithm (1994, 1992) | Y | Y | Y | Y | Y | Y | N | Y | N |
| *$HS^3$ algorithm* | Y | Y | Y | Y | Y | Y | Y | Y | Y |

Table 6: Comparison of $HS^3$ algorithm to other published avoidance algorithms

Notes:

(1) N = no; Y = yes

(2) (*) The conservativeness of this algorithm depends on the degree of lookahead, and cannot be judged for this algorithm. In general, the term "acceptable conservativeness" is open to question. In this comparison, the authors' examples were used for judging whether their algorithm was "acceptably conservative". Banaszak and Krogh's algorithm was pronounced "No" for this metric; the reasons were discussed in the literature review chapter.

Since the modeling of concurrent resource acquisitions and assembly are very similar from the view point of a Petri net model (both are modeled through synchronization of multiple resources), it follows that assembly operations can also be treated by the algorithm. System A demonstrated the modeling of concurrent resource acquisitions (for example, M1 and fixture A are concurrently acquired). Finally, the $HS^3$ algorithm does not require separating the set of resources into shared and unshared resources, hence all resources in the system can be shared.

## Future Research

Three future research directions were suggested by Gold (1979) upon proving that the exact solution of the deadlock avoidance problem was NP-complete: (1) fast solutions for restricted cases of the problem; (2) heuristic solutions for the general problem which will usually solve it in polynomial time, even though exponential time will be required in the worst case; and (3) fast algorithms for the general problem which will not always be correct, but will always err on the conservative side: all unsafe states will be correctly detected, but some safe states will be incorrectly classified as unsafe.

A review of the important algorithms within the area of manufacturing systems shows that not all of the algorithms fit into any single category. The finite lookahead algorithm presented by Viswanadham et al. (1990) is best classified in the second category. The elementary reduction procedure developed in the early part of this thesis also fits into the second category. The algorithms by Banaszak and Krogh (1990) and Heish and Chang (1992, 1994) are fast solutions for restricted cases (category 1), but also err on the conservative side (category 3).

The $HS^3$ algorithm developed here clearly fits into the third category, and in fact was motivated by this classification. Hence, it is logical that any future research development of this algorithm will proceed in the third direction. The future research directions that are recommended are:

1. The $HS^3$ algorithm needs further testing to be able to more accurately judge the conservativeness metric.
2. The $S^3$ heuristic needs to be refined on the following points: (a) When more than two processes meet the criteria of a shortest process the tie is broken arbitrarily. A selection rule needs to be developed which breaks the tie. (b) The current criteria for deciding that a process is the shortest is based on the total number of remaining resources that is needed to complete the process. There is no distinction between two resources of different types, and two resources of the same type.
3. Other adaptations of Habermann's algorithm within the realm of computer science need to be explored for their possible adaptation to manufacturing systems. Gold (1979) adapts Habermann's algorithm to the abstract notion of linearly ordered processes, while Hansen (1973) adapts the algorithm to hierarchically ordered processes. This author found it difficult to adapt these notions to real-world manufacturing systems, nevertheless, further research along these lines is recommended.
4. Other developments on deadlock avoidance algorithms as applied to operating systems need to be reviewed. Such algorithms could be used in lieu of the H-

algorithm procedure of the $HS^3$ algorithm if they are less conservative than the H-algorithm.

5. The refined approach developed for incorporating processes with alternate routings incurred a restriction. Future research could be directed towards developing other approaches which do not incur this restriction.

## Summary

The $HS^3$ algorithm developed in this thesis meets all the criteria in the statement of the practical version of the problem. The structure of the $HS^3$ algorithm comprises two procedures: (1) Habermann's algorithm, and (2) the $S^3$ heuristic. The H-algorithm guarantees that all deadlocks are avioded. In doing so, it can incorrectly classify some safe states as unsafe. This makes it a conservative algorithm. In order to reduce the conservativeness, the structure of the $HS^3$ algorithm was conceptualized. This structure was based on the discovery that if the system state is allowed to advance after being incorrectly classified as unsafe by the H-algorithm, then it is likely that the advanced state will be correctly recognized as safe. The $S^3$ heuristic is used for the purpose of advancing the system state whenever the H-algorithm declares it as unsafe. Hence the function of the $S^3$ heuristic is to reduce the conservativeness of the H-algorithm. Since the algorithm is applied to general systems (no restrictions on the system type are necessary for applying the algorithm), it is not possible to eliminate conservativeness and guarantee avoidance at the same time. The reason is that deadlock avoidance for the general case is NP-complete.

When applied to the e-BRN models of the two systems (A and B), the HS$^3$ algorithm avoids all deadlock states while preserving all the safe states (Fernandes 1995). Hence, for these systems, the conservativeness metric is zero. The systems were developed before the HS$^3$ algorithm was conceived, hence there could have been no bias towards the algorithm while designing the systems. The systems A and B possessed the characteristics of real world systems, i.e. concurrent processes, alternate plans and routings, multiple resource allocations (the modeling of which is similar to the modeling of assembly processes); thus the HS$^3$ algorithm has the elements of a realistic solution to the problem.

# BIBLIOGRAPHY

Agerwala T. (1979), "Putting Petri Nets to Work," *Computer*, v12n12, pp. 85-94.

Banaszak, Z.A. and B.H. Krogh (1990), "Deadlock Avoidance in Flexible Manufacturing Systems with Concurrently Competing Process Flows," *IEEE Transactions on Robotics and Automation*, v6n6, pp. 724-734.

Coffman, G., M.J. Elphick, and A. Shoshani (1971), "System Deadlocks," *Computing Surveys*, v3n2, pp. 67-78.

D'Souza, K.A. (1994), "A Control Model For Detecting Deadlocks in an Automated Machining Cell," *Computers and Industrial Engineering*, v26n1, pp. 133-139.

Desrochers, A.A. and R.Y. Al-Jaar (1995), "Analysis of Petri Nets," In *Applications of Petri Nets in Manufacturing Systems*, IEEE Press, New York, NY, pp. 116-117.

Dewdney, A.K. (1993), "NP-Completeness," In *The New Turing Omnibus*, Computer Science Press, NY, pp. 276-281.

Fernandes, R.A. (1995), "Application of the $HS^3$ Algorithm to Manufacturing System Examples," *Technical Report, CIM-TRS-95-RF1*, Center for Computer Integrated Manufacturing, Oklahoma State University, Stillwater, OK.

Gold, M.E. (1978), "Deadlock Prediction: Easy and Difficult Cases," *SIAM Journal of Computing*, v7n3, pp. 320-336.

Habermann, A.N. (1969), "Prevention of System Deadlocks," *Communications of the ACM*, v12n7, pp. 373-385.

Hansen, P.B. (1973), "Deadlocks," In *Operating System Principles*, Prentice Hall Inc., Englewood Cliffs, NJ, pp. 122-129.

Hsieh, F.S. and S.C. Chang (1992), "Deadlock Avoidance Controller Synthesis for Flexible Manufacturing Systems," In *Proceedings of the Third International Conference on Computer Integrated Manufacturing, IEEE,* Troy, NY, pp. 252-261.

Hsieh, F.S. and S.C. Chang (1994), "Dispatching Driven Deadlock Avoidance Controller Synthesis for Flexible Manufacturing Systems," *IEEE Transactions on Robotics and Automation*, v10n2, pp. 196-208.

Kamath, M. and N. Viswanadham (1986), "Applications of Petri Net Based Models in the Modeling and Analysis of Flexible Manufacturing Systems," *Proceedings of the 1986 IEEE International Conference on Robotics and Automation*, pp. 312-317.

Leung, Y.T. and G. Sheen (1993), "Resolving Deadlocks in Flexible Manufacturing Cells," *Journal of Manufacturing Systems,* v12n4, pp. 291-301.

Manber, U. (1989), "NP-Completeness," In *Introduction to Algorithms*, Addison Wesley Publishing Company Inc., Reading, MA, pp. 341-374.

Narahari, Y. and N. Viswanadham (1985), "A Petri Net Approach to Modelling and Analysis of Flexible Manufacturing Systems," *Annals of Operations Research,* v3, pp. 330-345.

Rosen, K.H. (1988), "Complexity of Algorithms," In *Discrete Mathematics and its Applications*, Random House, NY, pp. 84-90.

Viswanadham, N., Y. Narahari, and T.L. Johnson (1990), "Deadlock Prevention and Deadlock Avoidance in Flexible Manufacturing Systems Using Petri Net Models," *IEEE Transactions on Robotics and Automation*, v6n6, pp. 713-722.

Wysk, R.A. and J.S. Smith (1995), "A Formal Functional Characterization of Shop Floor Control," *Computers and Industrial Engineering*, v28n3, pp. 631-643.

Wysk, R.A., N.S. Yang, and S. Joshi (1991), "Detection of Deadlocks in Flexible Manufacturing Cells," *IEEE Transactions on Robotics and Automation*, v7n6, pp. 853-859.

Wysk, R.A., N.S. Yang, and S. Joshi (1994), "Resolution of Deadlocks in Flexible Manufacturing Systems: Avoidance and Recovery Approaches," *Journal of Manufacturing Systems*, v13n2, pp. 128-138.

Zhou, M. and F. DiCesare (1991), "Parallel and Sequential Mutual Exclusions for Petri Net Modeling of Manufacturing Systems with Shared Resources," *IEEE Transactions on Robotics and Automation*, v7n4, pp. 515-526.

# APPENDICES

## APPENDIX I: GLOSSARY

### Classical Petri nets

A Petri net is a 4-tuple (P, T, IN, OUT) where

$P = \{p_1, p_2, \ldots, p_n\}$ is a set of places

$T = \{t_1, t_2, \ldots, t_m\}$ is a set of transitions

$P \cup T \neq \phi, \quad P \cap T = \phi$

IN : (P X T) $\rightarrow$ N is an input function that defines directed arcs from places to transitions, and

OUT : (P X T) $\rightarrow$ N is an output function that defines directed arcs from transitions to places, and N is the set of non-negative integers.

### Input and output places

Given a transition t, the set of input places of t is denoted by .t, and the set of output places of t is denoted by t., where

$.t = \{p \in P : IN(p,t) \neq \phi\}$

$t. = \{p \in P : OUT(p,t) \neq \phi\}$

### Input and output transitions

Given a place p, the set of input transitions of p is denoted by .p, and the set of output transitions of p is denoted by p., where

$.p = \{t \in T : OUT(p, t) \neq \phi\}$

$p. = \{t \in T : IN(p,t) \neq \phi\}$

Marking of a Petri net

A marking of a Petri net is a function $M : P \rightarrow N$. A marking of a Petri net with n places associates with each place a certain number of tokens represented by dots, and represents a state of the Petri net. An initial marking $M_o$ is always associated with a Petri net. The terms state and marking may be used interchangeably.

Reachability set

The set of all markings reachable from the initial marking $M_o$. It is denoted by $R[M_o]$.

Enabling of a transition

A transition t is said to be enabled in a marking M if:

$M(p_i) \geq IN(p_i, t_j) \forall p_i \in .t_j$

Blocked Marking

A marking $M \in R[M_o]$ is said to be blocked if there exists a transition such that: 1) t has two or more input places, i.e. $|.t| > 1$, and 2) there exists a $p \in .t$ such that $M(p) \geq IN(p, t)$, and, 3) t is disabled in M.

Deadlocked marking

A marking in which no transition is enabled is said to be a deadlocked marking.

Safe marking

A marking which is neither deadlocked nor blocked is called a safe marking.

Immediate transition

A transition with no time associated with it, i.e. a transition that takes zero time to execute.

Vanishing marking

A marking in which at least one immediate transition is enabled.

Timed transition

A transition which takes a finite amount of time to fire.

Tangible marking

A marking in which only timed transitions are enabled.

Net invariant

A net invariant is a set of places I such that $\sum M(p)$, $p \in I$, is a constant for each reachable marking M, and I does not have any proper sub-sets that are invariants.

Safe State

A state from which there is at least one sequence of transitions that leads to the initial state.

Unsafe State

A state from which all sequences of transitions will lead to a terminating state, i.e a deadlock. An unsafe state may be a deadlock state itself, or it may be live in the sense that limited further evolutions are possible from that state.

Allocation State

A state defined by the sets of resources allocated to a number of concurrent processes.

Safe Sequence

An ordered set of processes which can be completed sequentially in the given order.

Claim of a Process

The set of resources that an initiated process would require in order to terminate without interruption.

Process Plan

A sequence of material transformation operations (including assembly with another part) needed to make a product, in accordance with a set of precedence constraints.

Alternate Process Plans

A choice of two or more sequences of material transformation operations necessary for producing a product, with the sequences satisfying all the precedence constraints for the set of operations for that product.

Alternate Routing

A choice of an alternative resource set for performing a material transformation operation.

Blocking operation

An operation in which a resource is waiting to transfer its load on to an available resource which can hold or further process that load.

Resource Place

A place which represents the availability of a resource. It is distinguished from an *operation* place by an initial store of one or more tokens.

Operation Place

A place which represents a resource activity. It is distinguished from a resource place by not being initially marked.

## APPENDIX II: TABLES OF MARKINGS

| | p1 | p2 | p3 | p4 | p5 | p6 | p7 | p8 | p9 | p10 | p11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| M0 | 2 | | | 1 | | | | | 2 | 2 | 2 |
| M1 | 1 | | 1 | 1 | | | | | 2 | 1 | 2 |
| M2 | | | 2 | 1 | | | | | 2 | | 2 |
| M3 | 1 | | 1 | 1 | | | | 1 | 1 | 1 | 2 |
| M4 | | | 2 | 1 | | | | 1 | 1 | | 2 |
| M5 | 1 | | 1 | 1 | | | | 2 | | 1 | 2 |
| M6 | 1 | | 1 | | 1 | | | 1 | 1 | 1 | 1 |
| M7 | | | 2 | | 1 | | | 1 | 1 | | 1 |
| M8 | 1 | | 1 | | 1 | | | 2 | | 1 | 1 |
| M9 | 2 | | | | 1 | | | 2 | | 2 | 1 |
| M10 | | | 2 | | 1 | | | | 2 | | 1 |
| M11 | 2 | | | 1 | | | | 2 | | 2 | 2 |
| M12 | 2 | | | | 1 | | | 1 | 1 | 2 | 1 |
| M13 | 2 | | | 1 | | | | 1 | 1 | 2 | 2 |
| M14 | 2 | | | | 1 | | | | 2 | 2 | 1 |
| M15 | 1 | | 1 | | 1 | | | | 2 | 1 | 1 |
| M16 | 2 | | | | | 1 | | 1 | 1 | 1 | 2 |
| M17 | 1 | | 1 | | | 1 | | 1 | 1 | | 2 |
| M18 | 2 | | | | | 1 | | 2 | | 1 | 2 |
| M19 | 1 | | 1 | 1 | | | 1 | 1 | | 1 | 2 |
| M20 | | 1 | 1 | 1 | | | | 1 | 1 | 1 | 1 |
| M21 | 1 | 1 | | 1 | | | | 2 | | 2 | 1 |
| M22 | 1 | 1 | | | 1 | | | 1 | 1 | 2 | |
| M23 | | 1 | 1 | | 1 | | | 1 | 1 | 1 | |
| M24 | 1 | 1 | | | 1 | | | 2 | | 2 | |
| M25 | 1 | 1 | | 1 | | | | 1 | 1 | 2 | 1 |
| M26 | 1 | 1 | | | 1 | | | | 2 | 2 | |
| M27 | | 1 | 1 | | 1 | | | | 2 | 1 | |
| M28 | | 1 | 1 | 1 | | | | | 2 | 1 | 1 |
| M29 | | 1 | 1 | | | 1 | | | 2 | | 1 |
| M30 | 1 | | 1 | | | 1 | | | 2 | | 2 |
| M31 | 1 | | 1 | 1 | | | 1 | | 1 | 1 | 2 |
| M32 | | | 2 | 1 | | | 1 | | 1 | | 2 |
| M33 | 2 | | | 1 | | | 1 | 1 | | 2 | 2 |
| M34 | 2 | | | | 1 | | 1 | | 1 | 2 | 1 |
| M35 | 1 | | 1 | | 1 | | 1 | | 1 | 1 | 1 |
| M36 | | | 2 | | 1 | | 1 | | 1 | | 1 |
| M37 | 1 | | 1 | | 1 | | 1 | 1 | | 1 | 1 |
| M38 | 2 | | | | 1 | | 1 | 1 | | 2 | 1 |
| M39 | 2 | | | 1 | | | 1 | | 1 | 2 | 2 |
| M40 | 1 | 1 | | 1 | | | | | 2 | 2 | 1 |
| M41 | 1 | 1 | | | | 1 | | | 2 | 1 | 1 |
| M42 | 2 | | | | | 1 | | | 2 | 1 | 2 |
| M43 | 1 | 1 | | 1 | | | 1 | | 1 | 2 | 1 |
| M44 | | 2 | | 1 | | | | | 2 | 2 | |
| M45 | | 2 | | | | 1 | | | 2 | 1 | |
| M46 | | 2 | | 1 | | | 1 | | 1 | 2 | |
| M47 | | 2 | | | | 1 | 1 | | 1 | 1 | |
| M48 | 1 | 1 | | | | 1 | 1 | | 1 | 1 | 1 |
| M49 | | 1 | 1 | | | 1 | 1 | | 1 | | 1 |
| M50 | 1 | | 1 | | | 1 | 1 | | 1 | | 2 |
| M51 | 2 | | | | | 1 | 1 | 1 | | 1 | 2 |
| M52 | 1 | 1 | | | | 1 | | 1 | 1 | 1 | 1 |
| M53 | | 1 | 1 | | | 1 | | 1 | 1 | | 1 |
| M54 | 1 | 1 | | | | 1 | | 2 | | 1 | 1 |
| M55 | | 1 | 1 | 1 | | | 1 | 1 | | 1 | 1 |
| M56 | | 1 | 1 | | 1 | | 1 | | 1 | 1 | |
| M57 | 1 | 1 | | | 1 | | 1 | 1 | | 2 | |
| M58 | | 1 | 1 | 1 | | | 1 | | 1 | 1 | 1 |
| M59 | 1 | 1 | | 1 | | | 1 | 1 | | 2 | 1 |
| M60 | | 2 | | 1 | | | | 1 | 1 | 2 | |
| M61 | | 2 | | | | 1 | | 1 | 1 | 1 | |
| M62 | | 2 | | 1 | | | 1 | 1 | | 2 | |
| M63 | 1 | 1 | | | 1 | | 1 | | 1 | 2 | |
| M64 | 1 | | 1 | 1 | | | 2 | | | 1 | 2 |
| M65 | 1 | 1 | | | | 1 | 1 | 1 | | 1 | 1 |
| M66 | | 1 | 1 | 1 | | | 2 | | | 1 | 1 |
| M67 | 2 | | | | | 1 | 1 | | 1 | 1 | 2 |
| M68 | 2 | | | 1 | | | 2 | | | 2 | 2 |
| M69 | 1 | 1 | | 1 | | | 2 | | | 2 | 1 |
| M70 | | 2 | | 1 | | | 2 | | | 2 | |

Table A1: Markings for e-BRN Model of System A in Figure (17)

Note: The author acknowledges the use of PESIM, a shareware package, for generating the reachability sets and graphs for the Petri net models in this thesis.

Table A2: Deadlock Markings for e-BRN Model of System A in Figure (17)

| | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 | P11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| M8 | 1 | | 1 | | 1 | | | 1 | 1 | | 1 |
| M9 | 2 | | | | 1 | | | | 2 | | 1 |
| M18 | 2 | | | | | 1 | | 2 | | 1 | 2 |
| M24 | 1 | 1 | | | 1 | | | 2 | | 2 | 1 |
| M54 | 1 | 1 | | | | 1 | | 2 | | 1 | 1 |
| M57 | 1 | 1 | | | 1 | | 1 | 1 | | 2 | |
| M62 | | 2 | | 1 | | | 1 | 1 | | 2 | |
| M66 | | 1 | 1 | 1 | | | 2 | | | 1 | 1 |
| M70 | | 2 | | 1 | | | 2 | | | 2 | |

| | p1 | p2 | p3 | p4 | p5 | p6 | p7 | p8 | p9 | p10 | p11 | p12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| M0 | 1 | | | 1 | | 1 | | | | 1 | 1 | |
| M1 | | 1 | | 1 | | 1 | | | | 1 | 1 | |
| M2 | 1 | | 1 | | | 1 | | | | 1 | 1 | |
| M3 | 1 | | | 1 | 1 | | | | | 1 | 1 | |
| M4 | | 1 | | 1 | 1 | | | | | 1 | 1 | |
| M5 | 1 | | 1 | | 1 | | | | | 1 | 1 | |
| M6 | | 1 | | | | 1 | 1 | | | 1 | 1 | |
| M7 | | 1 | | 1 | | 1 | | 1 | | | 1 | |
| M8 | 1 | | 1 | | | 1 | | 1 | | | 1 | |
| M9 | 1 | | | 1 | 1 | | | 1 | | | 1 | |
| M10 | | 1 | | 1 | 1 | | | 1 | | | 1 | |
| M11 | 1 | | 1 | | 1 | | | 1 | | | 1 | |
| M12 | | 1 | | | | 1 | 1 | 1 | | | 1 | |
| M13 | | 1 | | 1 | | 1 | | 1 | 1 | | | |
| M14 | 1 | | 1 | | | 1 | | 1 | 1 | | | |
| M15 | 1 | | | 1 | 1 | | | 1 | 1 | | | |
| M16 | | 1 | | 1 | 1 | | | 1 | 1 | | | |
| M17 | 1 | | 1 | | 1 | | | 1 | 1 | | | |
| M18 | | 1 | | | | 1 | 1 | 1 | 1 | | | |
| M19 | | 1 | | | 1 | | | | 1 | 1 | | 1 |
| M20 | | 1 | | | 1 | | | 1 | | | 1 | 1 |
| M21 | 1 | | | | | 1 | 1 | 1 | 1 | | | |
| M22 | 1 | | | | 1 | | | | 1 | 1 | | 1 |
| M23 | 1 | | | | 1 | | | 1 | | | 1 | 1 |
| M24 | | 1 | | | | 1 | | | 1 | 1 | | 1 |
| M25 | | 1 | | 1 | | 1 | | | 1 | 1 | | |
| M26 | 1 | | 1 | | | 1 | | | 1 | 1 | | |
| M27 | 1 | | | 1 | 1 | | | | 1 | 1 | | |
| M28 | | 1 | | 1 | 1 | | | | 1 | 1 | | |
| M29 | 1 | | 1 | | 1 | | | | 1 | 1 | | |
| M30 | | 1 | | | | 1 | 1 | | 1 | 1 | | |
| M31 | | 1 | | | 1 | | | | | 1 | 1 | 1 |
| M32 | 1 | | | | | 1 | 1 | | 1 | 1 | | |
| M33 | 1 | | | 1 | | 1 | | 1 | 1 | | | |
| M34 | 1 | | | | | 1 | | | 1 | 1 | | 1 |
| M35 | 1 | | | 1 | | 1 | | | 1 | 1 | | |
| M36 | 1 | | | | | 1 | | | | 1 | 1 | 1 |
| M37 | 1 | | | | | 1 | | 1 | | | 1 | 1 |
| M38 | 1 | | | 1 | | 1 | | 1 | | | 1 | |
| M39 | 1 | | | | 1 | | | | | 1 | 1 | 1 |
| M40 | | 1 | | | | 1 | | | | 1 | 1 | 1 |
| M41 | | 1 | | | | 1 | | 1 | | | 1 | 1 |
| M42 | 1 | | | | | 1 | 1 | 1 | | | 1 | |
| M43 | 1 | | | | | 1 | 1 | | | 1 | 1 | |

Table A3: Markings for e-BRN Model of System B in Figure (18)

Table A4: Deadlock Markings for e-BRN Model of System B in Figure (18)

| | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 | P11 | P12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| M5 | 1 | | 1 | | 1 | | | | | 1 | 1 | |
| M11 | 1 | | 1 | | 1 | | | 1 | | | 1 | |
| M17 | 1 | | 1 | | 1 | | | 1 | 1 | | | |
| M18 | | 1 | | | | 1 | 1 | 1 | 1 | | | |
| M19 | | 1 | | | 1 | | | | 1 | 1 | | 1 |
| M20 | | 1 | | | 1 | | | 1 | | | 1 | 1 |
| M21 | 1 | | | | | 1 | 1 | 1 | 1 | | | |
| M22 | 1 | | | | 1 | | | | 1 | 1 | | 1 |
| M23 | 1 | | | | 1 | | | 1 | | | 1 | 1 |
| M29 | 1 | | 1 | | 1 | | | | 1 | 1 | | |
| M31 | | 1 | | | 1 | | | | | 1 | 1 | 1 |
| M39 | 1 | | | | 1 | | | | | 1 | 1 | 1 |

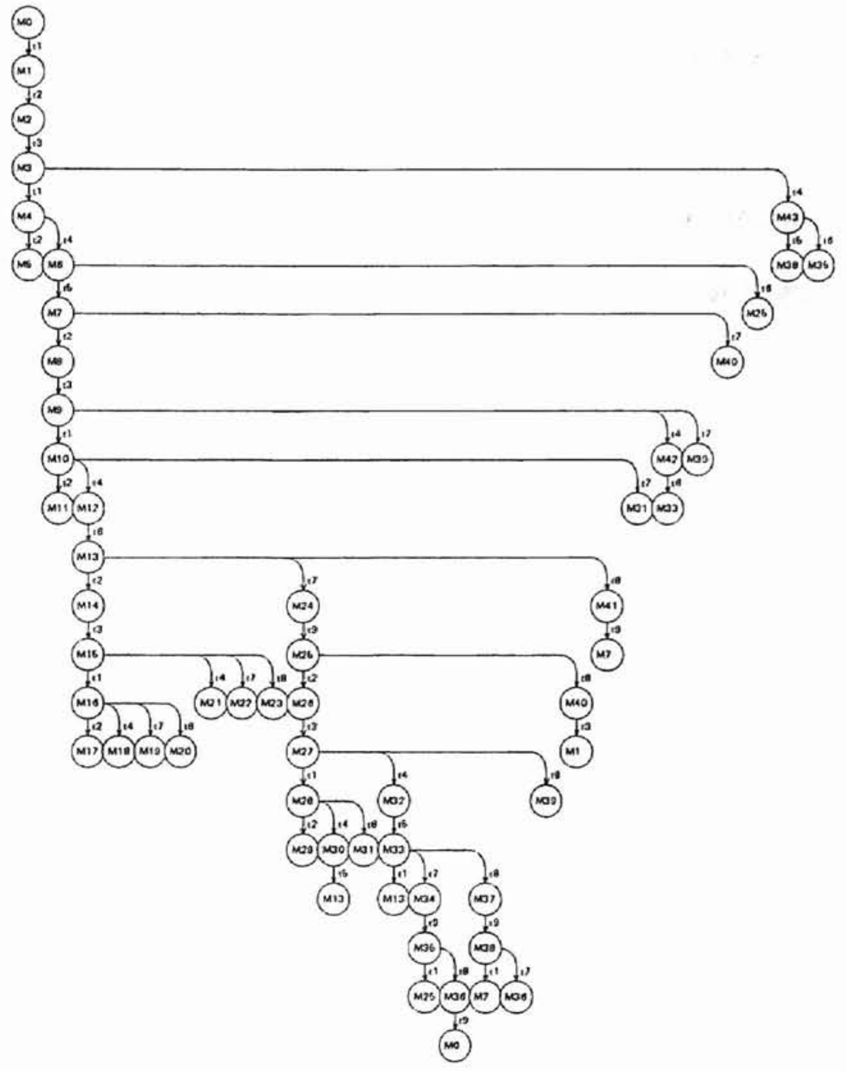


Figure A1: Reachability Graph for e-BRN for System B in Figure (18)

## APPENDIX III: PROOF OF HABERMANN'S SECOND THEOREM (1969)

*(and its incompatibility with dependent processes)*

A state is safe when all the participating processes in that state can be sequentially completed in the "worst case", i.e. when each process $P_k$ asks for all the resources it has claimed, and does not release any resources until it has been allocated all its claimed resources. Theorem II tells us that in such a case, there is no possibility of making the mistake of incorrectly ordering the processes for termination. Thus the worst case complexity of n! (there are n! ways in which we can rank a set of n processes) is reduced to n(n+1)/2.

Note: The notation used below is consistent with that used elsewhere, unless explicitly stated.

Let Q be a safe sequence, i.e. Q is an ordered set of processes which can be sequentially completed, then condition R4 holds for Q, i.e.

$$\forall_{P_k \in Q}\ \mathbf{b}_k \leq \mathbf{r} + \sum_{q(l) \leq q(k)} \mathbf{c}_l \text{ ...........(1)}$$

Let S be a subsequence, which fulfills R4, i.e.

$$\forall_{P_k \in S}\ \mathbf{b}_k \leq \mathbf{r} + \sum_{s(l) \leq s(k)} \mathbf{c}_l \text{ .............(2)}$$

i.e., S is a partial ordering of processes which can be sequentially completed.

Define S' as follows:

(a) $s'(k) = s(k)$ for $P_k \in S$;

(b) $q(l) \leq q(k) \rightarrow s'(l) \leq s'(k)$ for $P_k, P_l \in Q - S$;

This ordering implies that for $P_k \in Q - S$,

$$\sum_{q(l) \leq q(k)} c_l \leq \sum_{s'(l) \leq s'(k)} c_l \text{ ..................(3)}$$

since every $P_l$ that precedes $P_k$ in Q precedes also $P_k$ in S'.

From (1) and (3) it follows that

$$\forall_{P_k \in Q-S}\ b_k \leq r + \sum_{s'(l) \leq s'(k)} c_l \text{ .............(4)}$$

From (2) and (4) it follows that S' is a safe sequence. Since S' is defined as an extension of S, and S was any subsequence satisfying R4, the theorem follows. To prove that the theorem is not true when applied to dependent processes, we re-write expression (3) as follows:

$$\sum_{q(l) > q(k)} c_l > \sum_{s'(l) > s'(k)} c_l \text{ ...........(5)}$$

The above expression holds for independent processes since the right hand term is non-increasing. Since the completion of a dependent process implies the initiation of a subsequent process, it follows that the right hand term is not non-increasing, i.e. it is possible that $\sum_{q(l) > q(k)} c_l < \sum_{s'(l) > s'(k)} c_l$. Hence the theorem does not hold for dependent processes.

2

## VITA

Ralph A. Fernandes

Candidate for the Degree of

Master of Science

Thesis: DEADLOCK AVOIDANCE IN AUTOMATED MANUFACTURING SYSTEMS

Major Field: Industrial Engineering and Management

Biographical:

Education: Graduated from St. Xavier's College of Arts and Sciences, Bombay, India, in June 1988; received a Bachelor of Engineering degree in Automobile Engineering from the University of Bombay, Bombay, India, in July 1992. Completed the requirements for the Master of Science degree in Industrial Engineering and Management in December 1995.

Experience: Employed as a research assistant at the Center for Computer Integrated Manufacturing, Oklahoma State University, June 1994 to December 1995.

Professional Memberships: Alpha Pi Mu, Institute of Industrial Engineers.