ANALYSIS OF ASYNCHRONOUS CONCURRENT SYSTEMS BY PETRI NETS

Chander Ramchandani

Massachusetts Institute of Technology

Prepared for:

Office of Naval Research
National Science Foundation
Advanced Research Projects Agency

February 1974

AD - 775618

BIBLIOGRAPHIC DATA SHEET	1. Report Nos: GJ34671 + N00014-70-A-0362-0001 MAC TR-120	2.	3. Recipient's Accession No.

4. Title and Subtitle

Analysis of Asynchronous Concurrent Systems by Timed Petri Nets

5. Report Date: Issued February 1974

6.

7. Author(s)

Chander Ramchandani

8. Performing Organization Rept. No. MAC TR-120

9. Performing Organization Name and Address

PROJECT MAC, MASSACHUSETTS INSTITUTE OF TECHNOLOGY:
545 Technology Square, Cambridge, Massachusetts 02139

10. Project/Task/Work Unit No.

11. Contract/Grant Nos: GJ34671
N00014-70-A-0362-0001

12. Sponsoring Organization Name and Address:

Office of Naval Research
Department of the Navy
Information Systems Program
Arlington, Va 22217

Associate Program Director
Office of Computing Activities
National Science Foundation
Washington, D. C. 20550

13. Type of Report & Period Covered: Interim Scientific Report

14.

15. Supplementary Notes

Ph.D. Thesis, M.I.T., Department of Electrical Engineering, July 1973

16. Abstracts

Asynchronous concurrent systems are systems of interacting components whose operation is not governed by a central clock. This thesis presents a model for the analysis of such systems. In particular, the model used enables one to place bounds on the computation rate of a large class of asynchronous systems. The results obtained have wide applicability and several examples are given, drawn from areas such as computer systems modelling and operations research.

17. Key Words and Document Analysis. 17a. Descriptors

Petri Nets
Timed Petri Nets
Pipelined Computers
Parallel Computation
Computation Rate
Performance Analysis
Distributed Computation Systems
Concurrent Systems

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Unlimited Distribution Write Project MAC Publications	UNCLASSIFIED	221
	20. Security Class (This Page) UNCLASSIFIED	22. Price $5.75/1.45

MAC TR-120

ANALYSIS OF ASYNCHRONOUS CONCURRENT SYSTEMS BY PETRI NETS

Chander Ramchandani

February 1974

This research was supported in part by the National Science Foundation under research grant GJ 34671, and in part by the Advanced Research Projects Agency of the Department of Defense under ARPA Order No. 433 which was monitored by ONR Contract No. N00014-70-A-0362-0001.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
PROJECT MAC

CAMBRIDGE MASSACHUSETTS 02139

ANALYSIS OF ASYNCHRONOUS CONCURRENT SYSTEMS BY TIMED PETRI NETS

by

CHANDER RAMCHANDANI

Submitted to the Department of Electrical Engineering
on July 3, 1973, in partial fulfillment of
the requirements for the Degree of Doctor of Philosophy

ABSTRACT

This thesis is concerned with the modelling and performance analysis of systems which consist of concurrently acting components, an example of which is an asynchronous pipelined processor. The work is divided into two parts.

In the first part, a suitable model is developed for describing the structure of asynchronous concurrent systems. In conventional automata theory, the finite-state machine model is used to describe the behavior of systems; the problem with this is that a large number of states results when practical systems are modelled. In this thesis, each system component is modelled as a finite-state machine, and a system is viewed as an ensemble of interconnected finite-state machines.This has the advantage that the size of a system model grows linearly rather than exponentially with the number of system components. A subclass of Petri nets known as SMD (State Machine Decomposable) Petri nets is identified in order to formalize the notions of finite-state machines and their interconnection. For convenience, systems of interest are divided into two broad categories:

(a) Deterministic, or decision-free.
(b) Non-deterministic, or systems with decisions.

SMD Petri nets are used to model both classes of systems; in addition, a subclass of Petri nets known as LSP Petri nets is used to model those deterministic systems that cannot be modelled by SMD Petri nets.

The second part of the thesis is concerned with finding the computation rate of activities in real-world asynchronous concurrent systems. Practical systems are constructed from devices which have a finite speed of operation. Since Petri nets do not have time parameters as part of their definition, they can model the structure of systems but cannot be used to study their computation rate. The definition of Petri nets is augmented to model the speed of operation of a device in a system by assuming that the corresponding activity in the Petri net has a finite, non-zero time duration. The resulting nets are termed timed Petri nets, and methods are given for finding the computation rate of activities in timed SMD and LSP Petri nets. The results are applied to the analysis of several asynchronous systems drawn from areas within and outside the domain of computer systems .

THESIS SUPERVISOR: Jack B. Dennis
TITLE: Professor of Electrical Engineering

ACKNOWLEDGMENT

I would like to thank my thesis supervisor, Professor Jack Dennis for his support and encouragement during my years of graduate study. The readers of this thesis, Professors Robert Gallager and Robert Fano also deserve sincere thanks for their reading of the manuscript and their incisive comments that have served to improve the quality of the presentation.

My colleagues in the Fundamental Studies Group are to be thanked for creating a highly stimulating and enjoyable research environment. In particular, Henry Baker, Michael Hack, Richard Steiger and Fred Furtek are to be thanked for much valuable interaction.

I would also like to thank Professors Suhas Patil and Dave Liu (now at University of Illinois) for being extremely helpful graduate counsellors.

I greatly appreciate the patience of my wife Jayshree for enduring me during the completion of this thesis. Finally, thank you Gloria Marshall, for doing a splendid job of typing this thesis.

The work done in this thesis was done at Project MAC, MIT, and was supported in part by the National Science Foundation under research grant GJ-34671, and in part by the Advanced Research Projects Agency, Department of Defense, under Naval Research Contract N00014-70-A-0362-0001.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Background

The philosophy of this thesis can be described very effectively by the following quotation from T.C. Chen [C2]:

> In the quest for performance above and beyond that deliverable by hardware componentry, two alternative multiprocessing approaches to computer design can be taken. One is to subdivide each oncoming job among many identically constructed mechanisms, and is commonly referred to as parallelism. The other alternative is to develop a collection of specialized mechanisms capable of working simultaneously to form a general purpose organization; this is commonly called overlap, and an extreme form of this is called pipelining.

It is well known that present day machines are very wasteful in terms of resource utilization. One of the original arguments for time-shared multiprogrammed systems was the fact that they could permit better utilization of system resources than a batch-processing system by overlapping the operation of the processor, primary storage and secondary storage. The processor in contemporary computer systems is treated as a resource unit which is allocated to a user job or a task within a user job. Each processor typically consists of smaller processing units like adders, multipliers, an instruction fetch unit and an instruction decode unit, all of which can potentially be operated concurrently with each other. If it were possible to achieve this degree of concurrency, a much greater processing rate could be realized as a consequence. Of course, the over-

all throughput of the machine depends not only on the instruction processing rate of the processor, but also on factors like the speed of the main memory, its degree of interleaving, the type of secondary storage and finally on issues like the job mix and the scheduling strategy used. But by suitably reconfiguring main and secondary memory, the benefits of added concurrency in the processor could be realized as added throughput in the overall system. In most contemporary processors, overlapped operation of functional units within the processor is restricted to concurrent fetching and decoding of instructions. Many systems can be run in a multiprocessor configuration so that real parallelism is possible. However, this parallelism is between user jobs or between different tasks of the same user job, a situation which we term macro-parallelism. Dennis [D3] has advocated a computer organization in which parallel operation is possible right down to the level of instructions in a computer program or a user task. We term this micro-parallelism and it is clearly impractical on contemporary machines because of the overhead involved in switching the processor between instruction streams.

Jack Dennis has looked at the issues involved in designing a computer system to support micro-parallelism, and readers interested in his proposals for a memory organization and a representation scheme for programs and information structures are referred to his paper [D3]. We quote his remarks about the processing hardware:

> The organization of the processing hardware is intended to permit extensive sharing of multiple specialized cells by many computations to ensure statistically high utilization. It is envisioned that there be tens to hundreds of units of each cell class, operating independently and asynchronously using a service on demand principle of control.

The control needed in such a large complex system would be too formidable to tackle with a centrally clocked or a synchronous organization. Dennis has rightly pointed out the need for asynchronous operation of such a system. Each "cell" in Dennis' system corresponds to a functional unit of a certain type, like an adder or an instruction decoder, and the operation of a cell type will be overlapped or pipelined with the operation of other cell types. Thus, by a combination of parallelism and overlap, a statistically high instruction throughput should result. We will refer to such systems as "asynchronous concurrent systems" and will have occasion to use this term often in the course of the thesis. Dennis and Patil [D1, D2] have addressed themselves to the problem of evolving tools for the description and implementation of such systems. They have made much progress, and their efforts have resulted in a systematic technique for both the description of asynchronous concurrent systems together with a methodology for their realization using a set of elementary modules [D1, D2]. In this thesis, we would like to address ourselves to the question of modelling such systems with a view to predicting their performance. The performance of an asynchronous processor such as visualized by Dennis will depend upon several factors, some of which are listed below:

a) The number of functional units of each type.

b) The speed of operation of each functional unit.

c) Statistical properties of user jobs, e.g., their degree of parallelism, relative frequencies of the different instructions.

d) Gross statistical properties of user jobs, i.e., job arrival rates, mean length of user jobs.

Before we can incorporate these factors into a model for performance evaluation of asynchronous computer organizations, we must come up with a

suitable descriptive tool or model which can represent the structure of such systems in a manner that is both simple and accurate. In particular, factors like (a) and (b) should be readily incorporated into the model.

Models for asynchronous systems as they currently stand can successfully describe the structure of asynchronous systems, i.e., the structure and interconnection of the parts of the system. However, they do not have a way of incorporating information about factors like (b) and (c) as part of their definition. In this thesis, we have developed a model in which factors (a), (b), and (c) can be described in a natural and simple way. Thus, given a description of the structure of the system and the speed of operation of its parts together with statistics on the utilization each part, we can obtain a measure of the throughput of the system. If the system we are considering is an asynchronous pipelined processor, we can obtain a measure of its processing rate, given that we have information of types (a), (b) and (c) available to us. The actual throughput of such a processor when connected to memory units and in the presence of user jobs is not easy to find. We will not study issues of type (d) in connection with the performance of asynchronous computer systems, but will concern ourselves with finding an index of performance which we will call its information processing capacity or computation rate.

Our approach will be to study an existing model for asynchronous systems and explore in depth its applications to the modelling of various types of concurrent systems, including pipelined organizations in which parallelism may be present in each stage. This model, while adequate for describing the structure of asynchronous systems, does not contain information regarding the speed of operation of system components or any

information about the statistical utilization of the parts of a system. We show how such information can be incorporated into the model. The properties of this augmented model are studied, and a technique is given for analyzing the throughput rate of a large variety of asynchronous processing or computing systems. The model we have developed and the analysis techniques for it were motivated by the desire to study performance issues in asynchronous computer systems. However, the work can be applied with equal facility to analyzing numerous systems outside the realm of computer systems; the best example is that of an assembly line, and we shall interchangeably talk about pipelined processing systems or assembly lines, because both share fundamental characteristics such as overlapped and parallel operation. In the next section, we pursue the modelling of asynchronous systems in some depth, and illustrate the spirit and flavor of the thesis by concrete examples.

1.2 Modelling Asynchronous Systems

Our study of various types of asynchronous concurrent systems has led us to the conclusion that, by and large, we would like to distinguish between two broad classes of systems:

a) deterministic

b) non-deterministic (i.e., having decisions).

Let us explain briefly what we mean by this distinction.

Deterministic Systems: We will call an asynchronous system deterministic if during the course of operation of the system, there is never a situation when a decision has to be made between alternative courses of action in the system. An example will explain what we mean. Consider an automobile assembly line in which only one kind of car is being manufactured, and each car is made from similar components using the same sequence of assembly operations. Thus, in the course of operation of this system every assembly operation is needed for the assembly of each car, there being no difference in the sequence of steps needed for the manufacture of the fourth or eighth car output by the assembly line. To give another example, consider a pipelined floating-point adder, arranged in, say, three stages (see Fig. 1.2.1). Each stage performs a certain operation on the pair of operands input to the pipeline for addition. Thus, each operand pair that is absorbed at the input goes through the same sequence of operations before being output as a result. Such a pipelined system is also a deterministic system because no choice has to be made between alternative courses of action. So far we have looked at systems in which the objects being assembled (or added, etc.) go through an identical sequence of operations. A more general class of systems consists

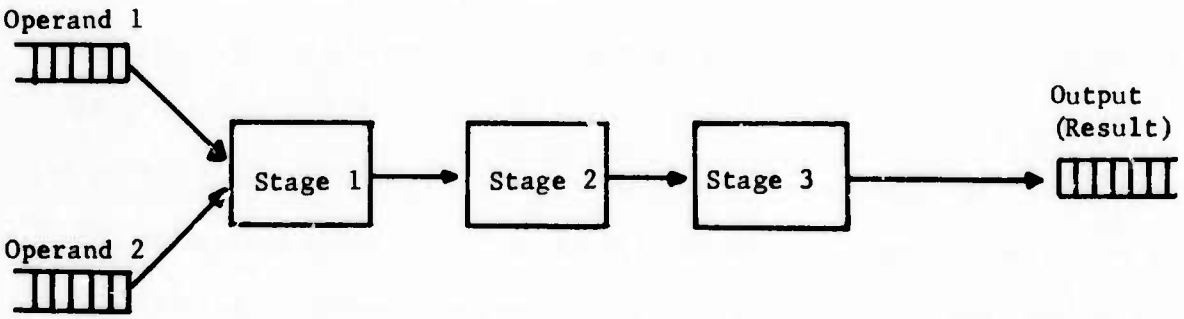


Figure 1.2.1

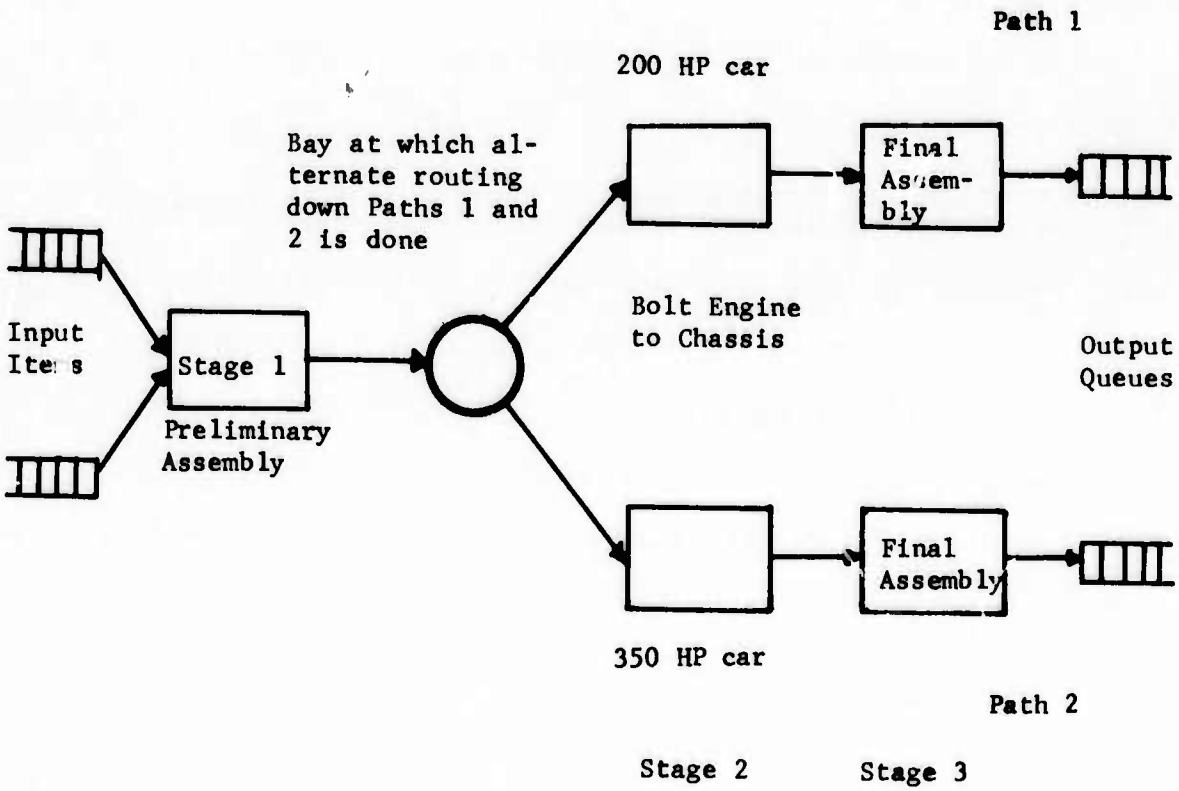


Figure 1.2.2

of those systems in which the objects being processed do not necessarily go through the same sequence of operations, but the operation of the system is still deterministic. An example of such a system is an assembly line for the manufacture of automobiles which alternately turns out two kinds of autos, one fitted with a 200 HP engine and the other with a 350 HP engine. We shall assume that the car with the more powerful engine has to be suitably braced and fitted with extra gadgetry, so that after the engine has been bolted to the chassis, subsequent assembly operations performed on the two cars are different. Thus, even though there is a point in the system at which a decision has to be made, this decision cannot, for example, be made arbitrarily by an assembly line operator. The operation of the system is, so to speak, "preordained," i.e., there is no point during the operation of the system at which a choice can be exercised between alternative courses of action.

Let us now move on to the class of systems we have termed non-deterministic systems.

Non-Deterministic Systems: We will loosely define non-deterministic systems as those in which there does exist a choice between alternative courses of action. An example of such a system is the assembly line of Figure 1.2.2, minus the restriction that the two types of automobiles be manufactured alternately. Thus, at the bay at which alternate routing of the two types of partially assembled autos was done, this routing can now be made random or controlled by some decision process other than a purely deterministic one. The decision could, for example, be made on the basis of up-to-the-minute customer orders received from car dealers. The operation of an assembly line such as this is said to be non-deterministic. Let us now give a more computer-related example. Consider the

simplified model of a pipeline processor shown in figure 1.2.3.

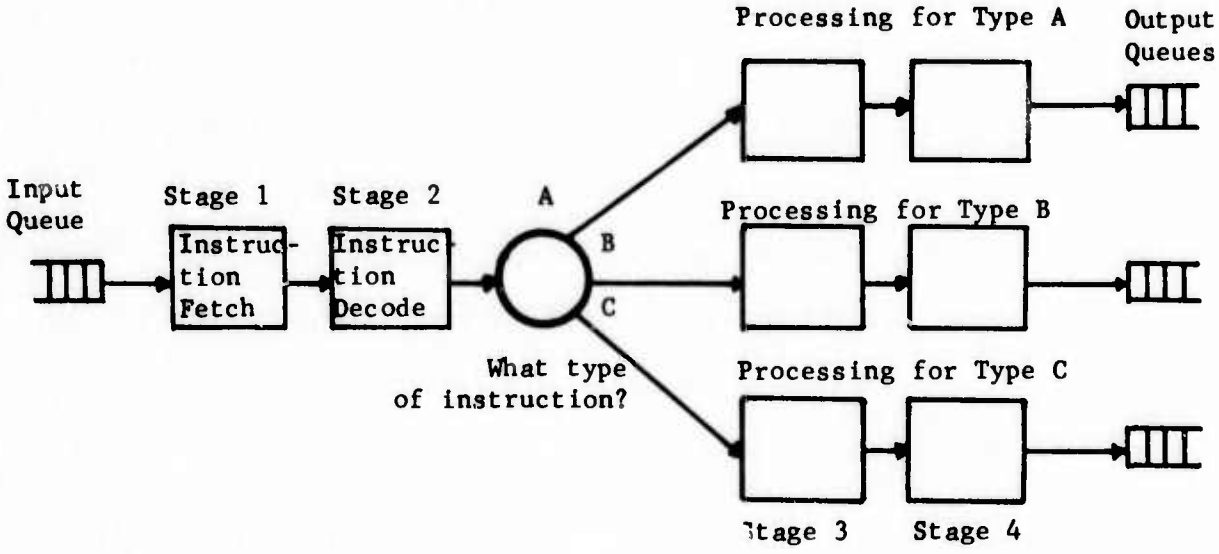


Figure 1.2.3. A Simplified Model of a Pipeline Processor for a Computer with Three Instruction Types.

We shall assume that the instruction set of the computer has three types of instructions, termed types A, B, and C. Instructions are fetched at Stage 1, decoded at Stage 2 and processed according to their type at Stages 3 and 4. The reader will notice that we have chosen the same notation for representing both the assembly line of figure 1.2.2. and the pipelined processor of figure 1.2.3. Both these types of systems,

together with a large variety of others, all have the basic characteristic that they consist of a set of parts together with some mechanism for coordinating their operation. In addition, the system parts could be arranged to exhibit overlapped operation and parallelism. Let us explain these two terms in the context of Figure 1.2.3. Each stage of the system is a part, whose operation can be overlapped with that of the other parts. We will say that the parts of the pipeline processor operate concurrently or in an overlapped fashion. We now come to parallelism. Suppose Stage 1 consists of two identical hardware units, each of which can independently fetch an instruction, and the two can operate concurrently with respect to each other. Stage 1 will be said to have parallelism of degree 2. In general, any stage is said to have parallelism of degree n if it has n physical hardware units available for processing.

Brief Statement of Thesis Problem: The problem we have addressed in this thesis is to study in some depth how to model asynchronous, concurrent systems such as the ones shown above. We have chosen a formalism known as Petri nets [H1, H2] to express the sequencing relationships between events in asynchronous systems. The problem with Petri nets, as with other models for asynchronous systems or parallel computation that we could have used is that they represent only the sequencing or "cause-and-effect" relationships between events in a system. Such a systems description is not adequate if we wish to study performance issues. For example, the assembly lines and the pipeline processors modelled in Figures 1.2.1. through 1.2.3. are all real world systems built from components or devices which take a certain amount of time to operate. Thus, the production of cars or the processing of instructions in an actual system is not instantaneous. The rate at which processed objects appear at

the output depends upon several factors, some of which we outlined in section 1.1 in connection with an asynchronous pipelined processor. Recapitulating, the throughput rate of an asynchronous processing system will depend upon two kinds of factors:

(a) factors intrinsic to the system, like its structure, the organization of its parts and the speed of operation of each part.

(b) extrinsic factors, like availability of items in the input queues, statistics on the type of items input to the system for processing.

The only extrinsic factor we will model will be statistics on the outcome of decisions during the course of operation of the system. In connection with Figure 1.2.3, this would mean the relative probabilities of occurrence of each of the instruction types A, B and C during the operation of the pipelined processor. As the reader will note, this is equivalent to the relative frequency of use of the system parts which perform the processing of these instruction types. We will not model other extrinsic factors, like arrival statistics for items in the input queues. The throughput rate that we thus calculate for a processing system will represent the maximum rate possible, assuming that it is connected to a balanced configuration of primary and secondary memory.

In the next section, we look at previous work in the areas of asynchronous systems, parallel computation and project scheduling which is relevant to the research presented in this thesis.

1.3 Related Work

Considerable work has been done in modelling the sequencing or control aspects of asynchronous systems, but few workers have actually considered issues of timing or speed of operation.

The work of R. Karp and R. Miller [K1, K2] is concerned with the development of a mathematical model called parallel program schemata to represent parallel algorithms. A set of uninterpreted operations is defined over a finite set of memory cells. With each operation is associated two subsets of the memory cells called the domain and range cells. Upon activation, called an initiation, an operation reads the current values in its domain cells and evaluates a function with these values as arguments. An unspecified time later, the results are deposited in the range locations, the latter action being called a termination. Control is conceptualized as a possibly infinite directed graph consisting of a set of control states (nodes) together with a transition function that specifies for each state initiation and termination pair the succeeding control state. A computation is a sequence of initiations and terminations that corresponds to a defined path in the control graph (emanating from a designated starting state) and that satisfy other rules. With this formulation, parallel activation of operations is possible, but their work on the whole is more concerned with control structures for parallel programs that properly terminate than with the issues involved in being able to represent continuously operating asynchronous concurrent systems.

Dijkstra [D4] considers a method by which asynchronous sequential processes may operate concurrently and communicate harmoniously. The processes are provided access to common integer values called semaphores.

The semaphores can be manipulated by means of two synchronizing primitives, the "P" and "V" operations which decrement and increment, respectively, the value of a semaphore by one. The P operation can be executed only when the current value of a semaphore is greater than zero. Conditions are investigated for an ensemble of interconnected sequential processes to operate without being deadlocked.

The work of Holt and Commoner[H2,C1] is very significant and has formed the starting point of our thesis. Their model, called Petri nets, is very simple to understand, and consists of two types of nodes termed places and transitions. A set of directed arcs connects places to transitions or transitions to places. Markers, called "tokens" are put on places, and each token that is put on a place corresponds to the holding of one instance of the condition corresponding to that place. When every input place of a transition has a token on it, it can "fire." A firing causes one token to be removed from each input place, and a token to be added to each output place. Commoner has investigated conditions for a subclass of Petri nets to operate in a deadlock-free manner. Hack has done an extensive investigation of a subclass of Petri nets termed Free Choice Petri Nets [H1].

So far, we have discussed models for asynchronous systems in which no mention was made of real time. Two bodies of work exist in which real time issues are entered into for asynchronous concurrent systems.

The first is the PERT network used in project scheduling [F1]. A PERT network is an acyclic directed graph with an input vertex and an output vertex. All arcs in the system lie on paths from the input vertex to the output vertex. Each arc denotes an activity in a project, and a method is given for calculating the shortest amount of time that it

takes to complete the project (the critical path), and also the earliest and latest time that any given activity can initiate in order that the project be completed in the shortest time possible. While PERT networks can explicitly represent concurrent activity in a system, they do not have the power to represent systems which operate in a recurrent or repetitive fashion. Nor, for that matter, can they explicitly represent system components with parallelism within each component. Most important, there is no provision in the structure of PERT networks for representing a choice between alternative actions in a system. This makes the model inadequate for modelling the complex asynchronous systems we would like to handle.

Martin and Estrin at UCLA have studied a model of parallel computation called the program flowchart [B1, M1, M2, M3]. Program flowcharts are directed graphs consisting of nodes and arcs. Nodes represent operations in a computer program and arcs represent data paths between them. An operation can take place when some suitable logical combination (and, exclusive-or) of its incoming arcs have data values on them. When an operation takes place, it absorbs data values from its incoming arcs and puts data values on some logical combination of outgoing arcs. The model is extremely powerful and can represent decisions explicitly. Martin and Estrin are concerned with modelling parallel computer programs, with a view to finding their mean computation time, i.e., the amount of time it takes to execute a computer program from start to finish. For this purpose, they assume each node to have a fixed (deterministic) time duration. A technique is given for finding the mean execution time of well-formed program flowcharts. Although the model is capable of representing recurrent processes, Martin and Estrin have confined themselves, in their

analysis, to models of processes which operate in a "one shot" fashion. Also, they have not developed their model to a point where it can be used for modelling complex pipeline processors or assembly lines. We will show in the course of our work the added generality that is possible with our approach to modelling asynchronous systems.

1.4 Overview of Following Chapters

1) It is shown how asynchronous concurrent systems can be looked upon as an ensemble of suitably interconnected finite state machines. Petri nets are introduced to formalize this notion, and also as a graphical tool for visualizing the structure of asynchronous systems. It is shown what it means for an asynchronous system to be well-formed in terms of the corresponding notions of liveness and boundedness for Petri nets. The two classes of asynchronous systems, deterministic and non-deterministic, are formalized. Boundedness is shown to be a decidable property for Petri nets, and liveness in turn is shown decidable for bounded nets. (Chapter 2).

2) An in-depth discussion is given of the structure of asynchronous concurrent systems in terms of SMD (State Machine Decomposable) Petri nets. The notion of a "current assignment" is introduced for transitions in a Petri net, and a consistent Petri net is defined as one which can support a consistent current assignment. Subclasses of Petri nets are introduced, chiefly Event Graphs, LSP (Live, Safe, Persistent) Petri nets and SMA (State Machine Allocatable) Petri nets. Deterministic and non-deterministic systems are studied in terms of these subclasses. (Chapter 3).

3) Timed Petri nets are introduced. These are Petri nets in which a transition executes for a fixed non-zero time called its firing time. The maximum rate at which an event occurs in a system is its computation rate. The computation rate of a large class of deterministic systems is found. (Chapter 4).

4) The general problem of finding the computation rate of non-deterministic systems is addressed. A bound is obtained for the computation rate of non-deterministic systems. Systems are identified for which this bound

is achievable. (Chapter 5).

5) Applications of our work to the modelling and analysis of real-world systems are given. (Chapter 6).

6) Unsolved problems are given, together with recommendations for future research. (Chapter 7).

CHAPTER 2

PETRI NETS AND VECTOR ADDITION SYSTEMS

2.1 Informal Introduction to Petri Nets

We will present the idea of the Petri Net model as an extension of the finite state machine model of conventional automata theory. The latter model is based on the assumption that we can abstract from a system a total system state, and the action of the system is described by the set of all allowable states and a set of rules that governs the transition of the system from one state to another. Let us give a simple example here to explain what we mean.

Consider a stage of the pipelined floating point adder shown in Figure 2.1.1(a). Assume that it can handle one pair of operands at a time and that when it is performing an operation on an operand pair, other operand pairs in its input queues must wait until it is through with the current operand pair. We can thus represent the operation of the stage of the pipelined adder as shown in Figure 2.1.1(a).

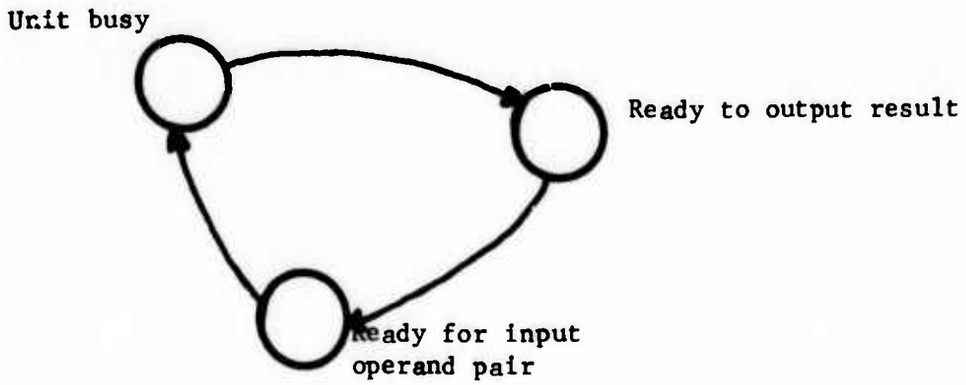


Figure 2.1.1(a)

The stage can be in one of three states, denoted by circles in Figure 2.1.1(a) :

(a) Ready for input operand pair.

(b) Unit busy (processing an operand pair).

(c) Ready to output result.

Figure 2.1.1(a) is drawn using the notation of finite-state automata theory, and a transition from one state to another is denoted by an arc. For reasons that will become clear soon, we will insert into each arc in Figure 2.1.1(a) a bar to denote the transition from one state to another. Also we would like to indicate what state a system is in at any given instant of time. We will designate this by the presence of a marker on the circle corresponding to that state (see Figure 2.1.1(b)).

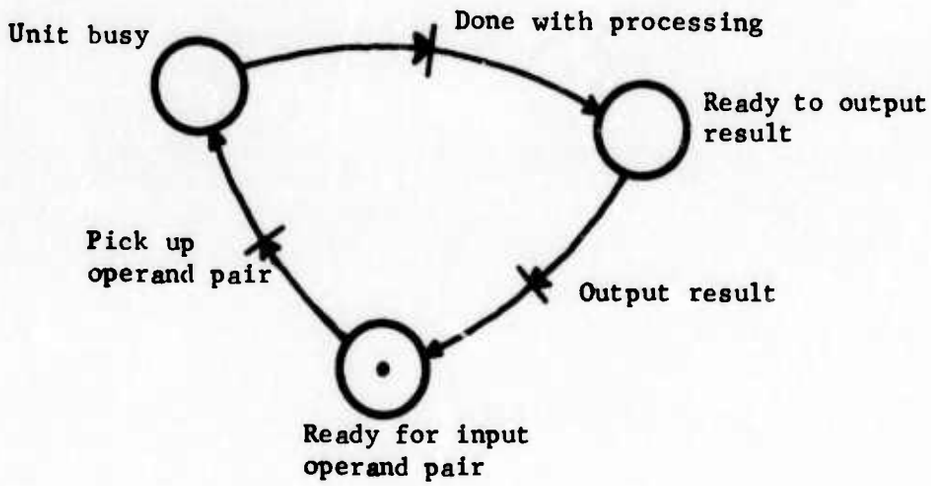


Figure 2.1.1(b)

The system shown in Figure 2.1.1(b) is in the state : "ready for input operand pair". When the system absorbs an input operand pair, it makes a transition to the state : "unit busy". Finally, when it is done with processing the operand pair, it becomes ready to output the result and makes a transition to the state entitled "ready to output result".

The system then outputs the result and becomes ready for the next input operand pair. This sequence of operations repeats.

Each stage of the system can be modelled in this fashion. We now consider how to model a two-stage pipelined adder, given that each stage is modelled in this fashion. Once again, by using techniques of conventional automata theory, we see that the two-stage system can be represented by a finite state machine which is the cross-product of the two machines[H4]. Let us attempt to carry out this construction. We will assume that the states of the two machines are labelled a,b,c and a',b',c', respectively (see Figure 2.1.2).

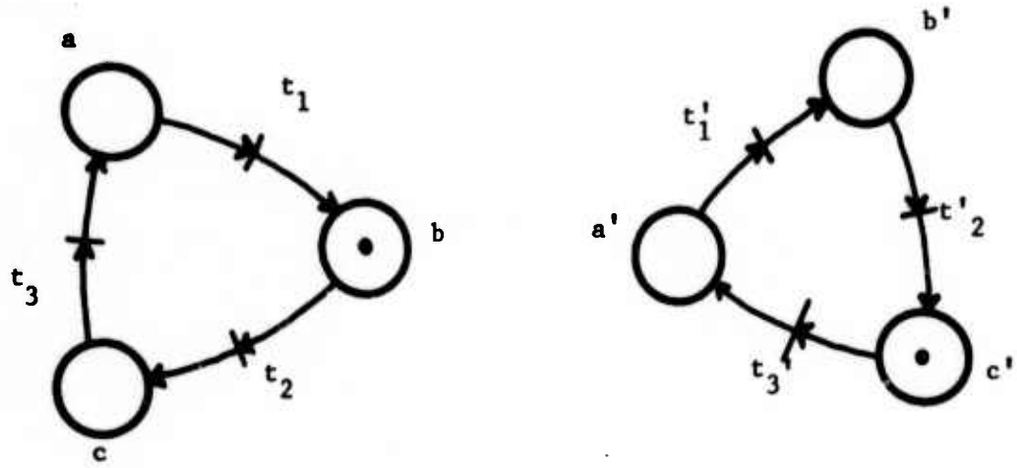


Figure 2.1.2

Figure 2.1.3 shows the resulting finite state machine. We see that it has nine states and a total of eighteen transitions between states. The problems with this representation are the following:

(a) The number of states grows as the <u>product</u> of the number of states in the individual stages.

(b) The identity of the individual stages has been lost. For all we know,

this could be the state diagram of a nine-state system consisting of only one system component (or stage).

The nine-state finite state machine obscures the structure of our two-stage pipelined adder. We note that if the pipeline has n stages, the number of states in the finite-state machine used to represent it would be 3^n ! To represent a five-stage pipeline, we would need 243 states. We clearly need a formalism which provides more economical descriptions of concurrent systems.

Let us briefly discuss parallelism in a system like a pipelined adder. Suppose we consider a two-stage pipelined adder in which the first stage has five functional units in parallel, and can thereby support up to five concurrent computations. The second stage will be assumed to have eight parallel stages. We can represent parallelism of degree five in the first stage by placing a total of five tokens on the finite-state machine used to represent it. Similarly, we place eight tokens on the finite state machine used to represent the second stage. (See Figure 2.1.4). Each token is assumed to move from one state to another independently of all other tokens. The state of the first stage is now a vector (n(a), n(b), n(c)), where each element of the vector represents the number of tokens on the corresponding place. Note that :

$$n(a) + n(b) + n(c) = 5, \quad \text{and}$$

$$n(a') + n(b') + n(c') = 8.$$

In order to model the first stage as a finite state machine, 3^5 states are needed. Similarly, 3^8 states are needed for the second stage, giving 3^{13} states in all for the cross product machine ! Suppose instead of attempting to model the above system as a finite state machine, we use

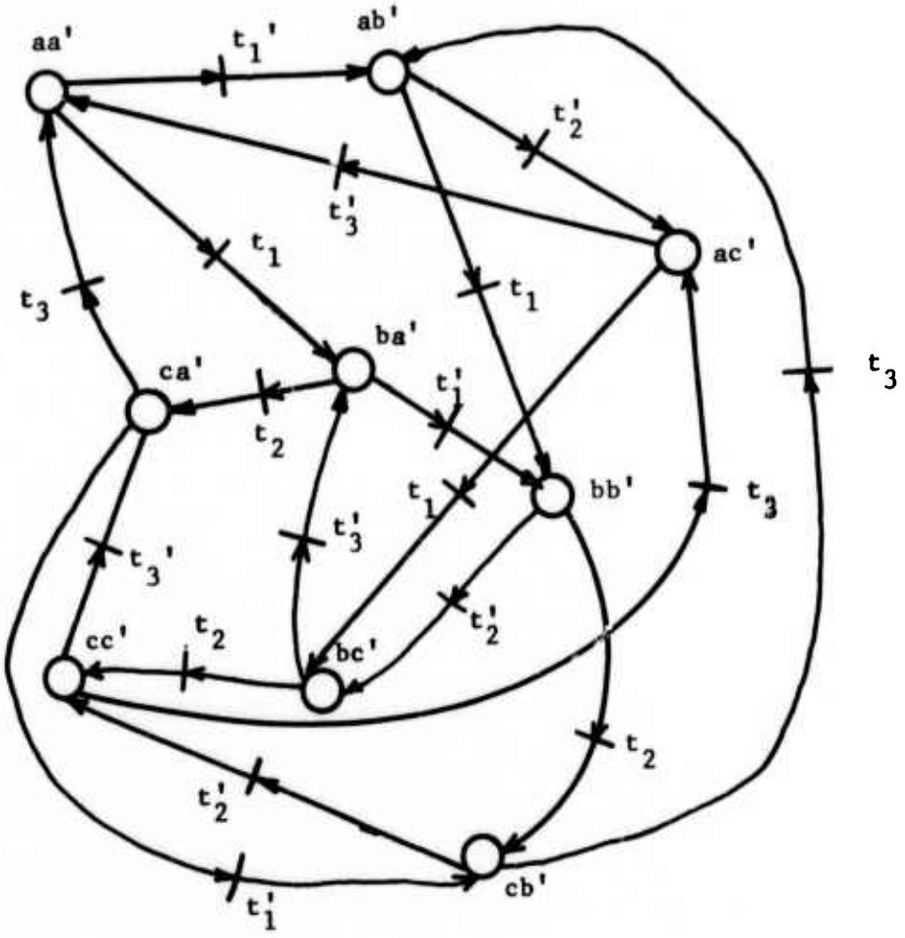


Figure 2.1.3 Finite State Machine Model of Pipelined Adder

the following artifice : coalesce t_3 and t_1' together, giving the diagram in Figure 2.1.5. The coalesced bar is relabelled 't'.

The output terminal of stage 1 is thus made the input terminal of stage 2. A transfer of an operand can take place when stage 1 is ready to output a result and stage 2 is ready for an input operand. In terms of the net in Figure 2.1.5, this is true when state c in stage 1 and state a' in stage 2 each have at least one marker on them. Figure 2.1.6 shows a configuration of the pipeline in which an operand pair can be transferred between the stages. Figure 2.1.7 shows the configuration that results after an

operand pair has been transferred.

We are now ready to introduce some nomenclature. Each circle in the diagram of Figure 2.1.5 is called a place , and the bars are called transitions. The act of transferring an operand pair between stages was achieved by an action called the firing of transition t.

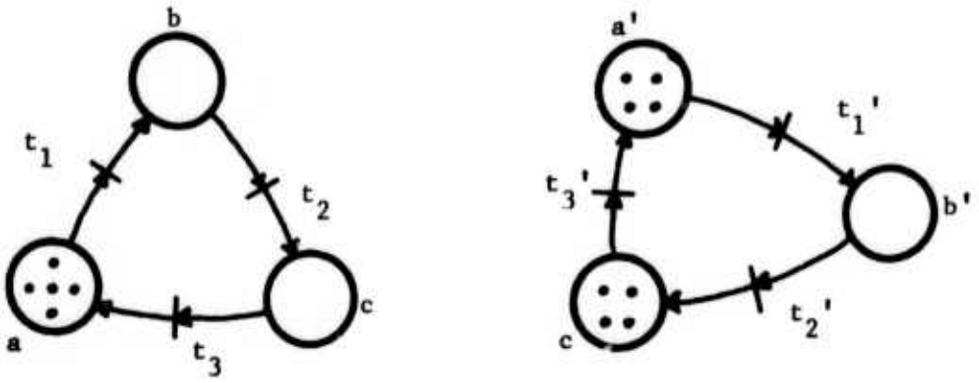


Figure 2.1.4.

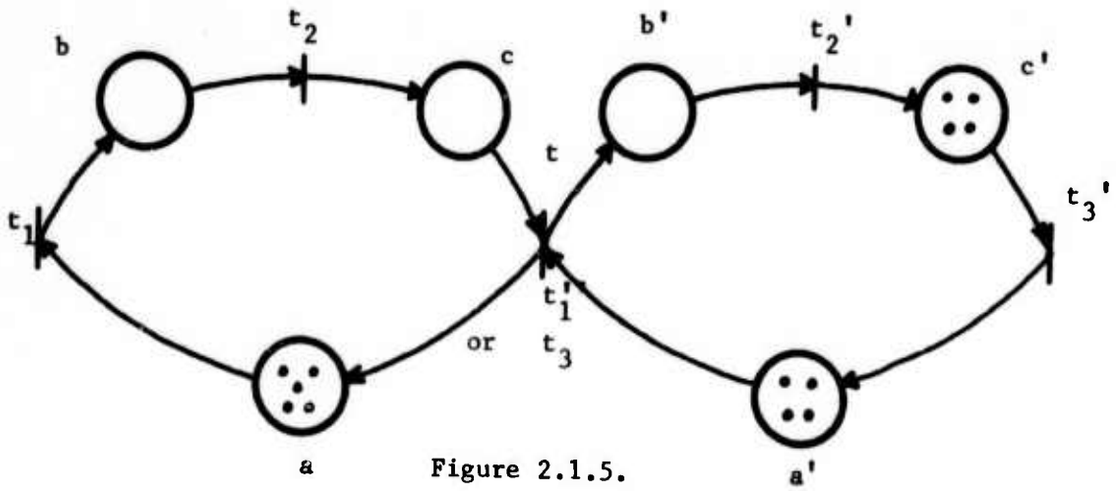
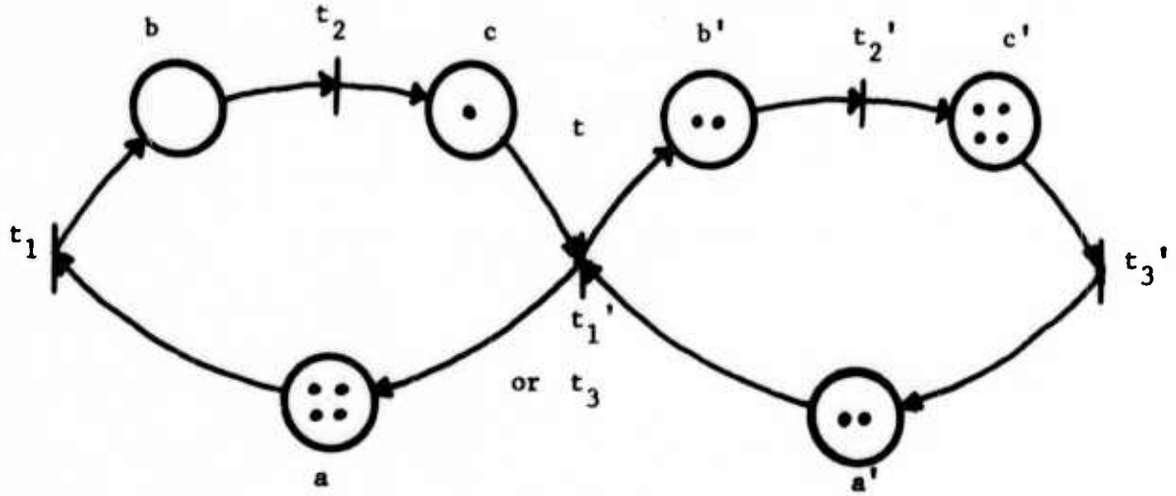


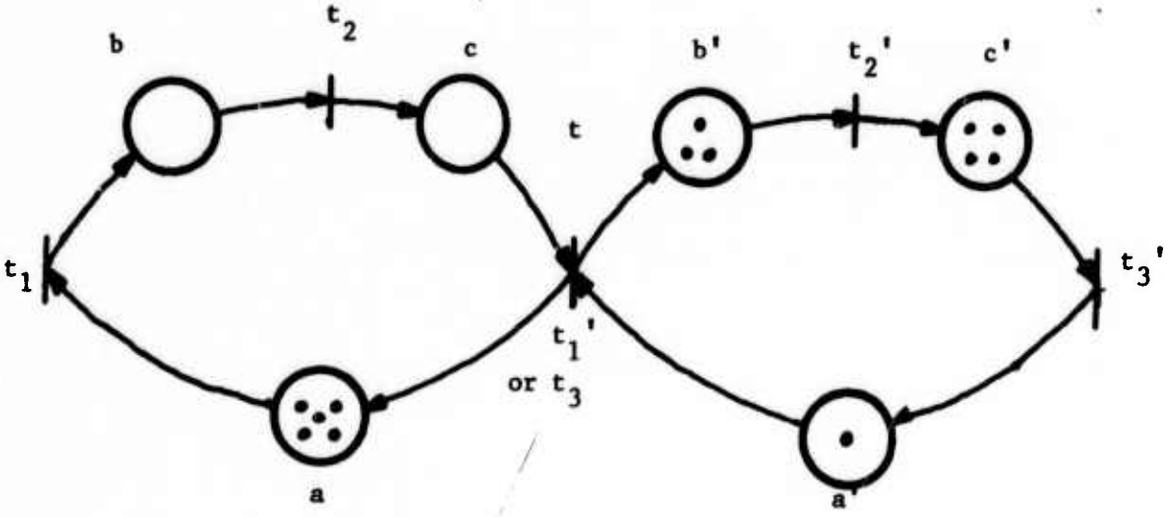
Figure 2.1.5.

Figure 2.1.6 shows a configuration of the system in which transition t can be fired. When t fires, a token is removed from each input place and added to each output place (see Figure 2.1.7). In system terms, each input



Before firing transition t

Figure 2.1.6



After firing t

Figure 2.1.7

place to a transition represents a pre-condition that has to be satisfied (or hold) in order that the action corresponding to the transition may take place. If there are several tokens on a place, it represents several instances of the holding of the corresponding condition. Thus, when the action corresponding to t occurs, it causes one instance of each pre-condition to cease and one instance of each output condition or post-condition to begin holding. When the finite-state machines representing Stages 1 and 2 are not connected, the stages they model are said to be mutually independent. In Figure 2.1.5, the finite state machines are said to be interconnected. Many other terms can be used instead of interconnected stages, for example, cooperating or mutually synchronized stages, but we will continue to use the term interconnected.

What we have done in the last few paragraphs is to introduce a way of modelling, in an economical way, systems of interacting parts in which overlapped operation and parallelism are both present. The terminology we have used is part of the definition of Petri nets, which are discussed more formally in the next Section. We have introduced the idea of Petri nets as a natural extension to the finite state machine model, and their advantages are obvious and considerable.

We would like to continue our informal approach in order to introduce some important notions that we will need in understanding the structure of asynchronous systems. Consider the finite-state machine in Figure 2.1.1(b). The state machine is strongly-connected, i.e., from every state there exists a directed path to every other state. The state machine in Figure 2.1.8 is not strongly-connected and we see that there is the possibility that the token can appear in place p_3, after which there is no way for the

token to appear in either place p_1 or place p_2. What this implies is that once the token appears in place p_3, transitions t_1,t_2 and t_3 cannot fire.

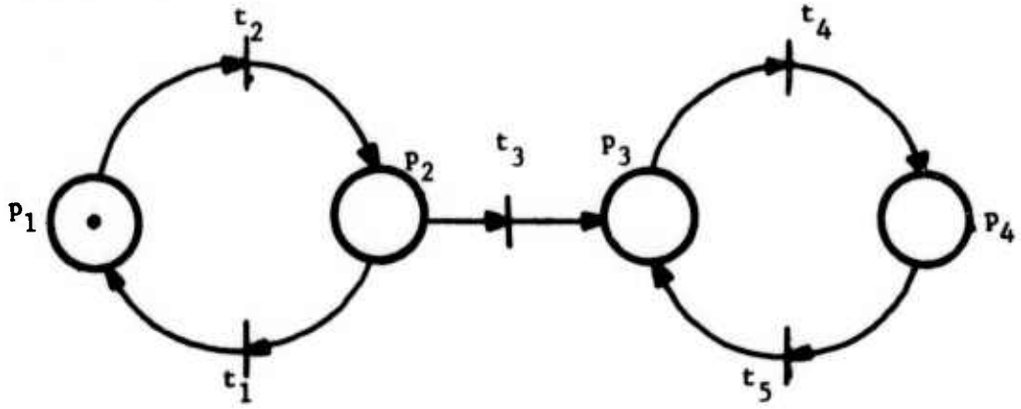


Figure 2.1.8 A finite state machine that is not strongly-connected.

We have seen that transitions can be used to represent operators in an actual system. In the systems we are interested in, we insist that every operator always be usable. A system component whose operation can be represented by the finite state machine of Figure 2.1.8 has certain operators (represented by t_1,t_2 and t_3) which are not used after some transient behavior of the system. In the steady state, transitions t_4 and t_5 fire alternately, over and over again. We will insist that each finite state machine be strongly-connected, and the reader can see that this is necessary for the composite system to satisfy the requirement that every operator always be usable in the course of operation of the system. If it so happens that some of the operators can never be used (i.e. their corresponding transitions can never be fired) , then those operators can be removed from the system without affecting its operation. Such operators are termed redundant.

The reader can verify that in the resulting net, transitions t_5, t_6 and t_6' are redundant.

A Petri net such as the one shown in Figure 2.1.9(b) has no redundant transitions and will be termed <u>live</u>. On the other hand, in the net shown in Figure 2.1.9(c), transitions t_5, t_6 and t_6' are redundant, whereas all others can always be fired. Such a net is termed <u>pseudo-live</u>. In some Petri nets, the operation of the net may lead to a configuration in which no transition can be fired. Such a net is termed non-live. In Figure 2.1.10(b) we show the construction of such a net from the two state machines of Figure 2.1.10(a).

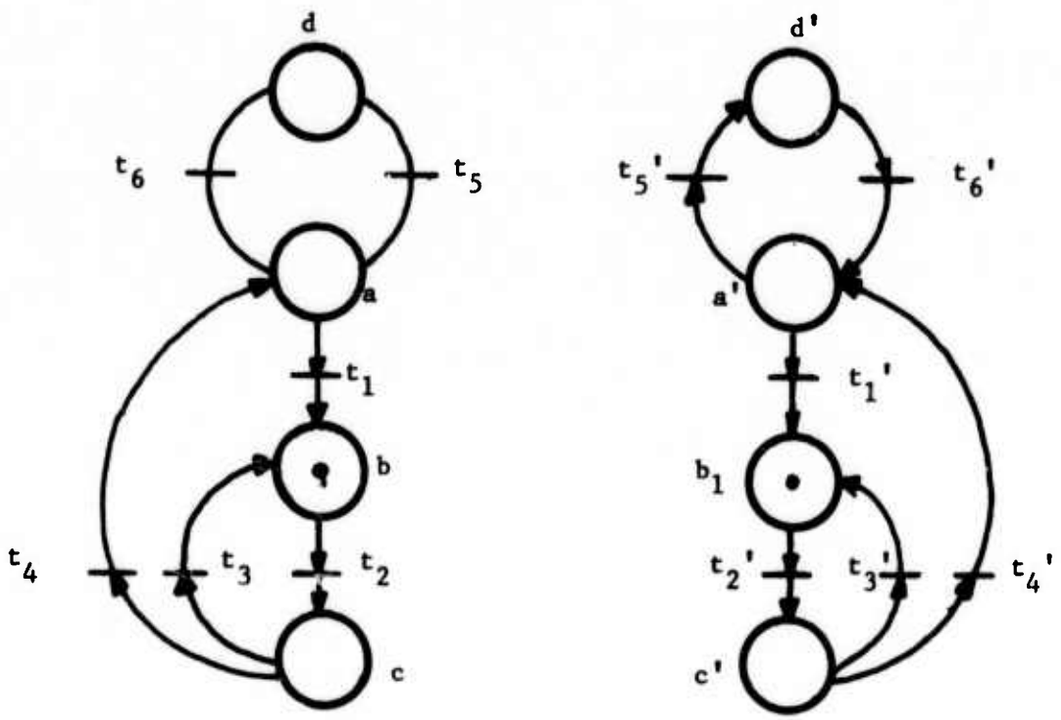


Figure 2.1.9(a)

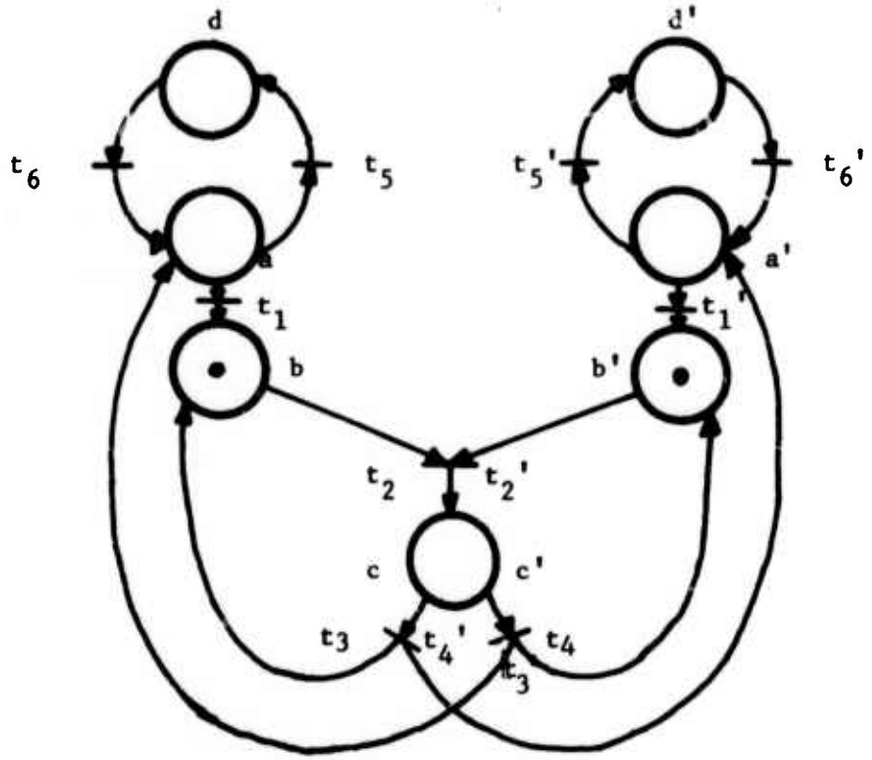


Figure 2.1.9(b)

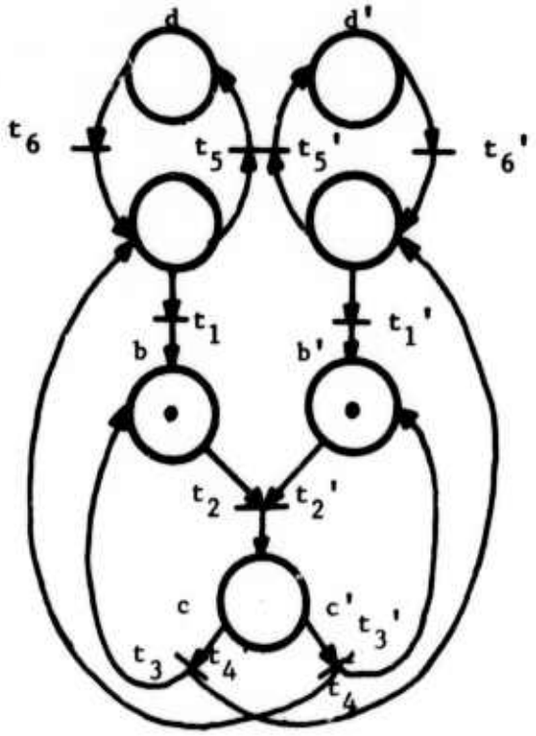


Figure 2.1.9(c)

In the configuration shown in Figure 2.1.10(b), either transition t_1 or t_2 can be fired. If t_2 is fired, the net attains a configuration in which no further transitions can be fired (see Figure 2.1.10(c)). The net is an example of a non-live net, and the configuration shown in Figure 2.1.10(c) is a deadlocked configuration.

What we have done so far is to motivate a formal study of Petri Nets, which we now proceed to do in the next section.

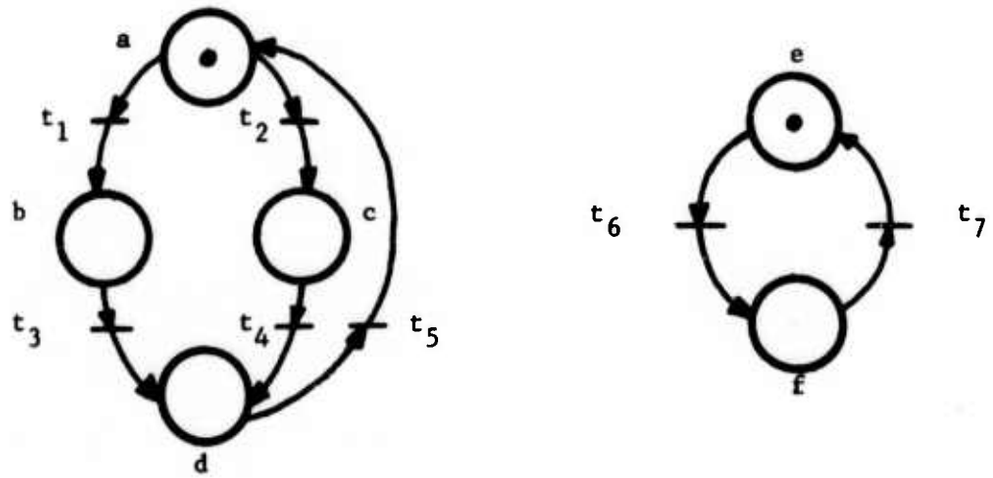


Figure 2.1.10(a)

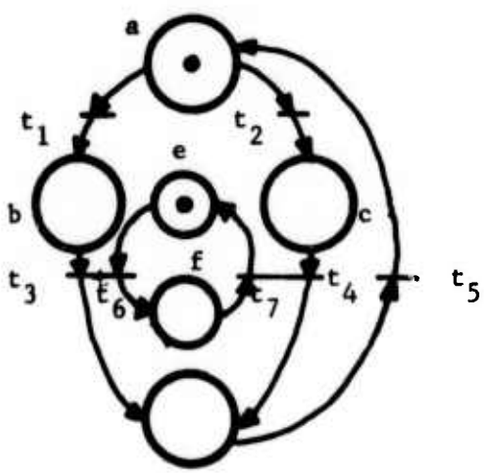


Figure 2.1.10(b)

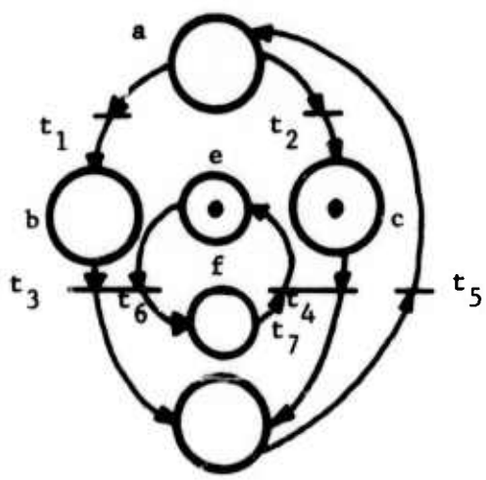


Figure 2.1.10(c)

2.2 Formal Definition of Petri Nets and Relevant Concepts

2.2.1 Petri Nets

We are now in a position to introduce the reader to the formal definition of Petri nets together with the related terminology that we will use in the rest of this thesis.

Definition 2.2.1 A Petri Net $\mathcal{P}$ is a three-tuple $\langle P,T,A \rangle$

where P is a non-empty set of distinctly-labelled places $\{p_1, p_2, \ldots p_n\}$.

T is a non-empty set of distinctly-labelled transitions $\{t_1, t_2, \ldots t_m\}$.

A is a relation; it corresponds to a set of arcs, where each arc is either from a place to a transition or from a transition to a place:

$A \subseteq P \times T \cup T \times P$.

Definition 2.2.2 A marking M is a function such that $M: P \longrightarrow \eta$, where η is the set of non-negative integers. The non-negative integer associated with a place represents the token load of that place, or the number of tokens on it.

A Petri net with a marking will be referred to as a marked Petri net (see figure 2.2.1 for an example).

Notation and Terminology

A node of a Petri net is either a place or a transition. We now introduce a convenient notation for the predecessor or successor nodes of any node in a Petri net. We will refer to it as the dot notation.

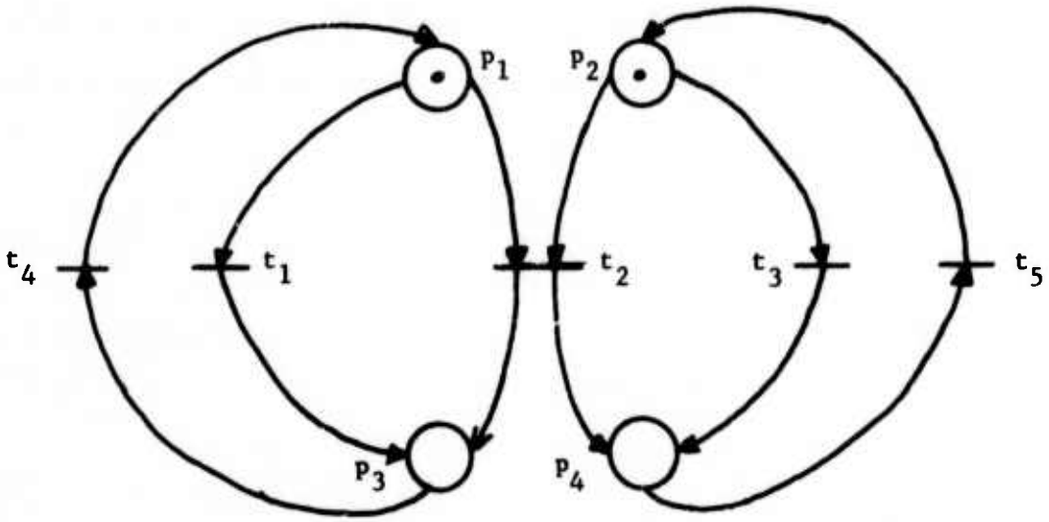


Figure 2.2.1. A marked Petri Net

$\langle p,t \rangle \in A$ is written $p \cdot t$.

$\{y | x \cdot y\}$ is written $x^{\bullet}$.

$\{y | y \cdot x\}$ is written $^{\bullet}x$.

Example: In figure 2.2.1, $^{\bullet}t_2 = \{p_1, p_2\}$.

$p_1^{\bullet} = \{t_1, t_2\}$.

The dot notation is also applied to designate the successor or predecessor set of a set of places or transitions. Thus,

$^{\bullet}\{t_2, t_4\} = \{p_1, p_2, p_3\}$.

$\{p_3, p_2\}^{\bullet} = \{t_2, t_3, t_4\}$.

A transition t in a Petri Net $\mathcal{P}$ is said to be enabled iff every input

place $p_i \in \cdot t$ has at least one token on it. An enabled transition can be fired. When a transition fires, a token is removed from each input place and added to each output place $p_j \in t\cdot$. Suppose M^a is the marking that results when an enabled transition t_a fires at marking M. We write this

$$M \xrightarrow{t_a} M^a.$$

Now suppose transition t_b can be fired at marking M^a. Let M^b be the marking that results when transition t_b fires. We write

$$M \xrightarrow{t_a} M^a \xrightarrow{t_b} M^b.$$

Continuing in this fashion, let us suppose that at every new marking that results when a transition fires, at least one transition can be fired. We can write this

$$M \xrightarrow{t_a} M^a \xrightarrow{t_b} M^b \text{ --- } \cdots\cdots \xrightarrow{t_n} M^n.$$

The sequence of transition firings $t_a t_b t_c \cdots\cdots t_n$ is termed a firing sequence. If $\sigma = t_a t_b \cdots\cdots t_n$, we write

$$M \xrightarrow{t_a t_b \ldots t_n} M^n$$

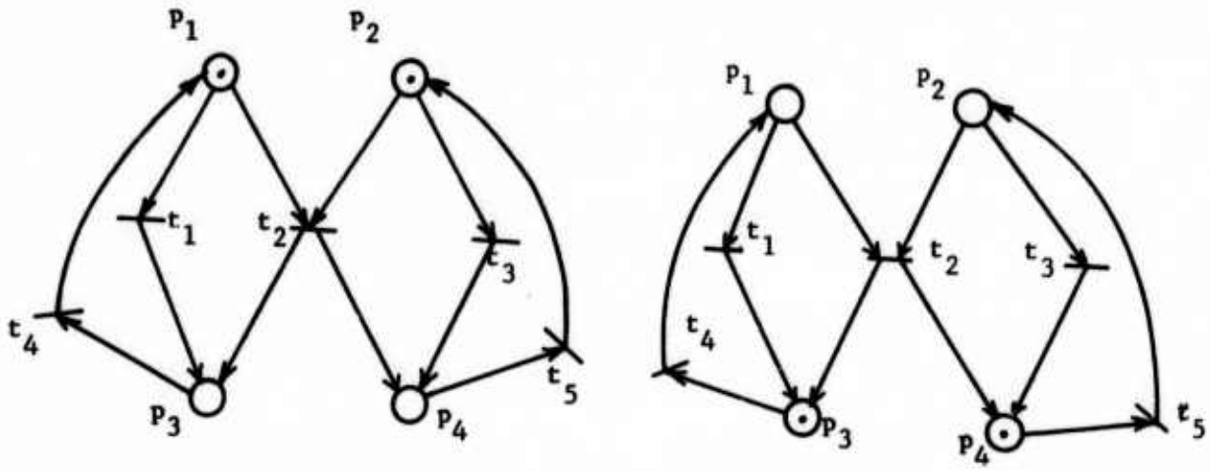
or $$M \xrightarrow{\sigma} M^n.$$

Example.

In Figure 2.2.2(a) transition t_2 is enabled. Figure 2.2.2(b) shows the marking that results when transition t_2 is fired at marking M. The reader can also verify that $t_2 t_4 t_5 t_1 t_3$ is a firing sequence for the net.

Definition 2.2.3: A marking M^j is said to be reachable from marking M^i if there exists a firing sequence σ such that

$$M^i \xrightarrow{\sigma} M^j.$$



Marking M
(a)

Marking M'
(b)

Figure 2.2.2

Definition 2.2.4: The forward marking class $\vec{M}$ of a marking M is the set of markings that is reachable from M. i.e.,

$$\vec{M} = \{M^1 \mid \exists\, \sigma \in T^* \text{ and } M \xrightarrow{\sigma} M^1\}.$$

Note: T^* is the set of strings on the alphabet

$$T = \{t_1, t_2, \ldots\ldots t_n\}.$$

Notation: We have defined M as a function that assigns a token load to every place $p_i \in P$. M can also be looked upon as a vector, the ith element of which corresponds to the token load of the ith place p_i in the Petri net.

Example: The marking M of the net in Figure 2.2.2(a) can be written as

$$M = (1, 1, 0, 0)$$

The forward marking class of M is seen to be

$$\vec{M} = \{(1,1,0,0),\ (0,0,1,1),\ (0,1,1,0),\ (1,0,0,1)\}.$$

Definition 2.2.5: A marking M is live for a transition t iff for every marking M^1 in the forward marking class $\vec{M}$ there exists a firing sequence which fires t.

Definition 2.2.6: A marking M is live for a Petri Net $\mathcal{P}$ iff it is live for every transition in the net.

Definition 2.2.7: A marking M is bounded for a place p, iff there exists an integer N such that for every marking $M^1 \in \vec{M}$, $M^1(p) \leq N$. If N = 1, the marking is safe for place p.

Definition 2.2.8: A marking M is bounded (or safe) for the Petri net $\mathcal{P}$ iff M is bounded (or safe) for every place in the net.

In Section 2.1, we pointed out that there exist marked Petri nets in which a marking is reached in which no transitions can be fired, i.e., the net can reach a deadlocked configuration. Suppose this deadlocked configuration is called M^1, and suppose that the net is in some configuration (i.e., has a marking) M. If $M \xrightarrow{\sigma} M^1$, then σ is said to be a killing sequence for the net at the given marking M. A net with a live marking has no killing sequence.

There is one more important issue we would like to consider in this subsection. In Section 2.1, we pointed out that the only Petri nets we will use for modelling asynchronous concurrent systems will be nets which have a live marking. We would also like the marking for a Petri net to be bounded. This means that the only Petri nets we would like to consider

are strongly-connected nets. Let us see why. Consider a non-strongly connected Petri-net $\mathcal{P}$. Then, there must exist in the net two portions of the net A and A' such that all arcs between them are directed from A to A', as shown in Figure 2.2.3. Each of the nodes a and c in A can either be a place or a transition. Suppose a is a place. Then b must be a transition. Since the marking for the net is live, it means that b can be fired repeatedly. But each firing of transition b removes a token from place a. Now suppose we do not fire transition b at all. This would mean that all the tokens which were previously being removed by firings of b can now stay on place a, which means that the number of tokens on a can become unbounded.

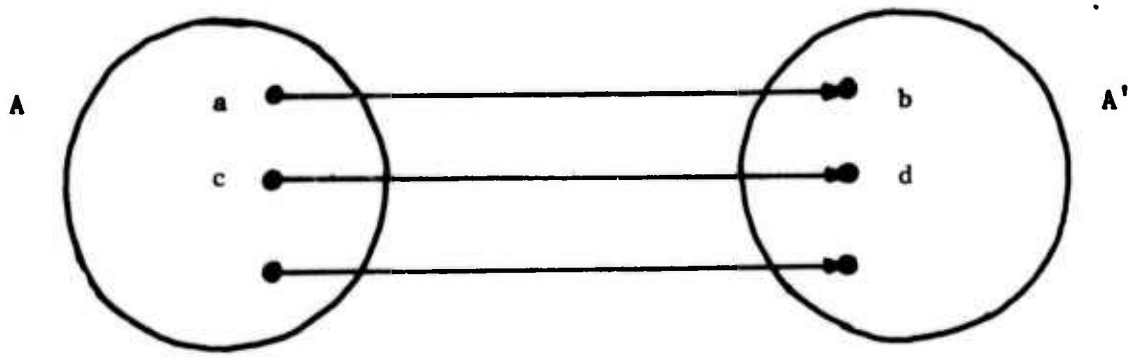


Figure 2.2.3 A Petri net that is not strongly-connected.

Now suppose that a is a transition and b is a place. Since the marking for $\mathcal{P}$ is live, we can fire transition a repeatedly, causing the token content of b to become unbounded. Thus, if $\mathcal{P}$ is not strongly-connected,

the token content of some place in $\mathcal{P}$ becomes unbounded. Now, since we are concerned only with nets which have a live, bounded marking, all Petri nets we will consider will be strongly-connected, unless explicitly stated otherwise.

In Section 1.2, we said that there are two broad classes of systems that we would like to distinguish between. We now pursue that line of thinking formally.

2.2.2 Deterministic and Non-Deterministic Petri Nets

We would like to make an important distinction between two classes of Petri nets, which we term deterministic and non-deterministic. First, we look at a structural condition of Petri nets known as conflict.

Definition: Consider a Petri net $\mathcal{P} = \langle P,T,A \rangle$. Two or more transitions $t_1, t_2, \ldots\ldots t_k$ are said to be in conflict if there exists a place p such that $p \cdot t_1 \wedge p \cdot t_2 \ldots\ldots \wedge p \cdot t_k$.

Note: '$\wedge$' denotes the logical "and" operator.

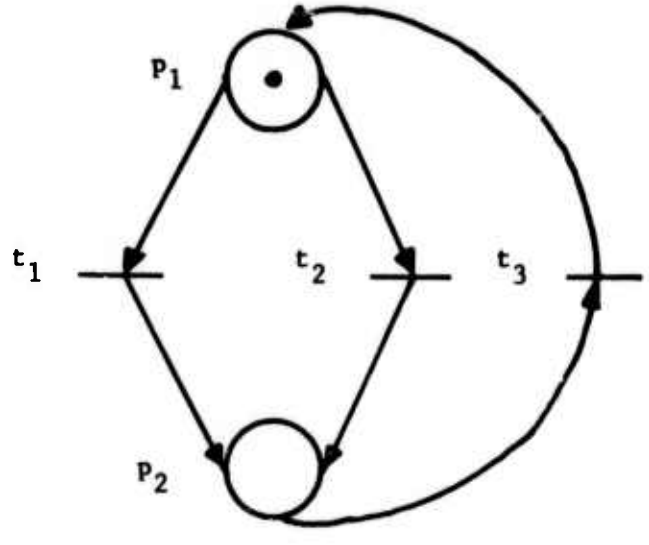


Figure 2.2.4

In Figure 2.2.4 transitions t_1 and t_2 are in conflict at place p. With the marking shown, a token in place p_1 can be removed by the firing of either transition t_1 or transition t_2. Thus, for the given marking, we have a choice between firing either transition t_1 or transition t_2, and when either transition fires, the other ceases to be enabled. Before we can go any further, we must make a few definitions.

Definition 2.2.9: A marking M is persistent for a transition t in a Petri net $\mathcal{P}$ if t has the property that once it is enabled at any marking $M^1 \in \vec{M}$, it cannot cease to be enabled by the firing of any transition other than itself.

Definition 2.2.10: A marking M is persistent for a Petri net $\mathcal{P}$ iff it is persistent for every transition $t \in \mathcal{P}$.

The net in figure 2.2.4 has the property that it has no marking which is persistent for transitions t_1 and t_2. Such a net is termed a non-deterministic net, and represents a system in which there is a distinct choice between alternative actions (see section 1.2). In contrast, the net in figure 2.2.6 is a deterministic net or a net with a persistent marking.

We now introduce some convenient ways of referring to the nets we have been discussing.

A net with a Live, Bounded or a Live, Safe marking will be termed an LB or LS net, respectively. If a Live, Bounded marking is also Persistent we will call the Petri net an LBP net. A net with a Live, Safe, Persistent marking will be called an LSP net. The net in Figure 2.2.5 is an LSP net.

Petri nets are a graphical representation of a mathematical system

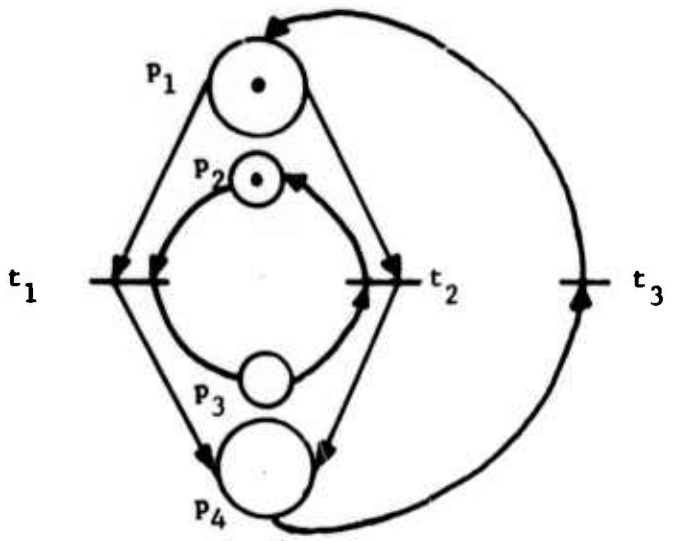


Figure 2.2.5 A Petri net with a live, safe, persistent marking

known as vector addition systems. In the next section, we use some results in vector addition systems to prove the decidability of certain issues in Petri nets.

2.3 Vector Addition Systems and their Relationship to Petri Nets

In this section, we give a brief description of vector addition systems, but a more thorough coverage can be found in Karp and Miller [K2].

Definition 2.3.1: An r-dimensional vector addition system V is a pair

$V = (s,W)$ where

(1) $s \in N^r$, $N = \{0,1, \ldots\}$.

(2) W is a finite set of r-dimensional integer vectors

$W = \{w_1,\ldots,w_k\}$, $w_i \in \{0,\pm1,\pm2, \ldots\}^r$

The reachability set R(V) is the set of vectors given by

$R(V) = \{x_i | x_i = s + w_{i1} + w_{i2} + \ldots + w_{ik}\}$ where $w_{ij} \in W$.

a. $w_{ij} \in W$, $j = 1,2,\ldots k$.

b. $s + w_{i1} + w_{i2} + \ldots + w_{ik} \geq 0$, $k = 1,2,\ldots t$.

The reachability set is the set of all points that can be reached by some path from s using vectors from W and never leaving the first orthant.

Example 2.3.1: As an example of a four element vector addition system, consider

$V = (s,w)$

where $s = (1,1,0,0)$

$W = \{w_1 = (-1,0,+1,0), w_2 = (-1,-1,+1,+1), w_3 = (0,-1,0,+1),$
$w_4 = (+1,0,-1,0), w_5 = (0,+1,0,-1)\}$.

The reachability set R(V) of this vector addition system consists of four vectors $\{(1,1,0,0), (0,0,1,1), (0,1,1,0), (1,0,0,1)\}$.

Notation

Let us denote the vectors in the reachability set by y_i,

i.e., $R(V) = \{y_1, y_2, \ldots y_m\}$.

Also, if y_i is a vector, then $(y_i)_k$ will denote the k^{th} element of y_i. We will draw a directed graph with the elements of R(V) as vertices, and an arc from vertex y_i to y_j if there exists a vector $w_l \in W$ such that

$$y_j = y_i + w_l.$$

Each arc will be labelled with its corresponding vector w_l above, and the resulting diagram shows at a glance the vectors in the reachability set and how they can be reached from one another. We will call this diagram the reachability diagram of the vector addition system V.

As an example, we give the reachability diagram corresponding to the vector addition system of example 2.3.1.

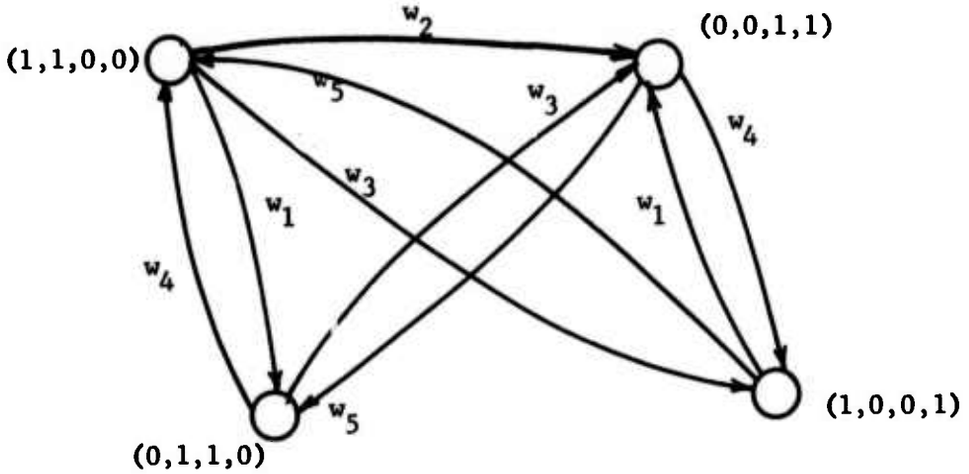


Figure 2.3.1 The Reachability Diagram for the Vector Addition System of Example 2.3.1

If the reachability set of a vector addition system is finite, the reachability diagram is a finite-state machine, and it is practical to draw it as we did for our example. If, on the other hand, the reachability set is infinite, we must find an alternative representational tool. For this purpose, we introduce the following terminology:

(1) We define a quantity ω such that if n is an integer then $n < \omega$, and $n + \omega = \omega$. The quantity ω essentially represents infinity and we discuss its use shortly.

(2) A rooted tree is a directed graph such that one node (the root node) has no arcs directed into it, each other node has exactly one arc directed into it and each node lies on a directed path from the root node. If β and γ are distinct nodes of a rooted tree and there is a directed path from β to γ, we write $\beta < \gamma$. If there is an arc directed from β to γ, then γ is a successor of β. A node with no successors is called a leaf. With an r-dimensional vector addition system $V = (s,W)$, we shall associate a rooted tree $T(V)$. Each node $\beta \in T(V)$ is labelled with an r-dimensional vector $\ell(\beta)$, where each element of $\ell(\beta)$ belongs to the set $N \cup \{\omega\}$. A recursive procedure is given below for the construction of $T(V)$. Note that the label for a node is assigned when the node is added to the tree.

(1) Create a root node and label it s.

(2) Let β be a node in the partially created tree with no successors. If, for some node γ, $\gamma < \beta$ and $\ell(\gamma) = \ell(\beta)$, then β is a leaf in $T(V)$. Otherwise, for each $w \in W$ such that $w + \ell(\beta) \geq 0$, add a node β_w and make it the successor of β. For each β_w the ith coordinate of $\ell(\beta_w)$ is assigned as follows:

(i) If there exists $\gamma < \beta_w$ such that $\ell(\gamma) \leq \ell(\beta) + w$ and $\ell(\gamma)_i < (\ell(\beta) + w)_i$ then $\ell(\beta_w)_i = \omega$.

(ii) If no such γ exists, then $\ell(\beta_w)_i = (\ell(\beta) + w)_i$.

(3) Repeat step (2) until no new nodes can be added to the tree.

We show in Appendix I that for any vector addition system V, the tree T(V) is finite.

Example 2.3.2. As an example (taken from Karp and Miller), consider the following vector addition system V = (s,W).

s = (1,0,0,0,0)

W = {(-1,1,0,0,0), (-1,0,0,1,0), (0,-1,2,0,0), (0,1,-1,0,0), (0,0,0,-1,2) (0,0,0,1,-1)}.

The rooted tree T(V) is :

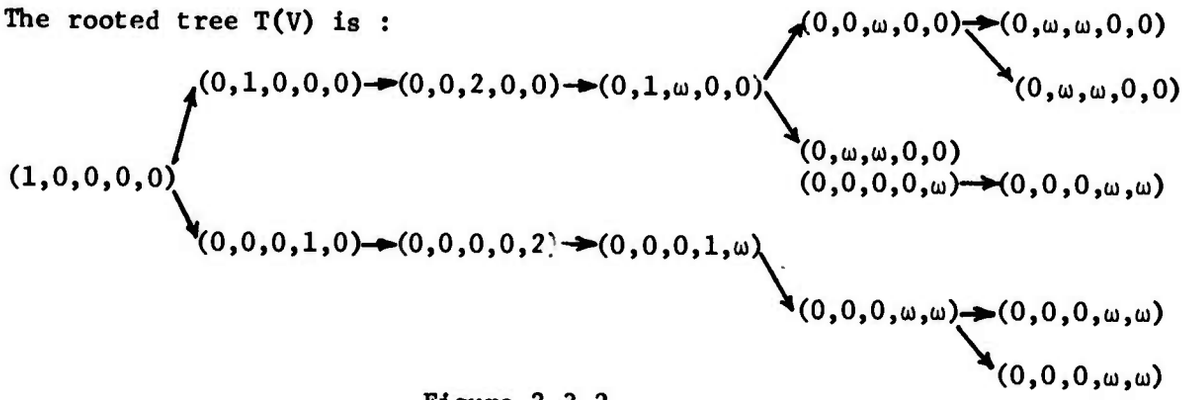


Figure 2.3.2

All our decidable results about vector addition systems reduce to inspecting the tree T(V) and using the following theorem.

Theorem 2.3.1: For any vector addition system V and any integer vector x of the same dimension

$(\exists y \in R(V)$ such that $x \leq y) \Leftrightarrow (\exists\, \beta \in T(V)$ such that $x \leq \ell(\beta))$.

Proof: Given in Appendix I.

Corollaries

The following questions are decidable:

Corollary 2.3.1.1

$\forall$ y $\in$ R(V) is $(y)_i \leq n$ for some finite n?

Corollary 2.3.1.2

$\forall$ y $\in$ R(V), is a given set of elements of the vector y simultaneously bounded?

Corollary 2.3.1.3

Is R(V) a finite set?

In addition, if the questions in Corollaries 2.3.1.1 or 2.3.1.2 are true, the bounds can be effectively computed. For the vector addition system in Example 2.3.2, we see that $\forall$ y $\in$ R(V), there are elements which become unbounded, and hence R(V) is infinite. Furthermore, the elements that can become simultaneously unbounded are (3), (5), (2,3) or (4,5).

Petri Nets and Vector Addition Systems

It is easy to see that for every marked Petri net there is a corresponding vector addition system. Let us explore this issue in detail.

Let $\mathcal{P}$ be a Petri net with initial marking M. We can construct an equivalent vector addition system V = (s,W) as follows:

The dimension of the vector addition system is made equal to the number of places in the Petri net, and the starting vector s is taken to

be the initial marking of the net. It now remains for us to show how to construct the set of vectors W. Consider a marking M' in the marking class $\vec{M}$, at which a transition t_h is enabled. Since the firing of t_h decrements the token content of each of its input places by one, and increments the token load of each of its output places by one, we can represent a firing of t_h by adding a vector w_h to the marking M' to yield the marking that results when transition t_h is fired. The elements of the vector w_h are given the following values:

+1 if the corresponding place is an output place of t_h.

-1 if the corresponding place is an input place of t_h.

0 otherwise (we are assuming that there are no self-loops in $\mathcal{P}$; see figure 2.3.3).

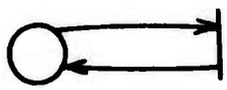


Figure 2.3.3. A Self-loop

We now assert that the vector addition system V is equivalent to the Petri net $\mathcal{P}$ with marking M in the following sense:

(a) For every marking $M_q \in \vec{M}$, there exists a vector $y \in R(V)$ such that $y = M_q$.

(b) A vector addition sequence ν is a sequence of vector additions

$w_j + \ldots + w_s$ such that it transforms a vector y_i to vector y_s. i.e.,

$y_i \xrightarrow{\nu} y_s$ where

$\nu = w_j \ldots w_s$.

We see that corresponding to every vector addition sequence ν there exists a firing sequence $\sigma = t_j \ldots t_s$ in the Petri net $\mathcal{P}$, and vice versa.

We now establish some decidable results for Petri nets based on the decidability of the corresponding issues for Vector Addition Systems.

Theorem 2.3.2 It is decidable if a given marking M for a Petri net $\mathcal{P}$ is bounded.

Proof: Let $\mathcal{P}$ be a Petri net and M its initial marking. By the construction above, we can find an equivalent vector addition system V = (s,W). From Corollary 2.3.1.2, we know that the following is decidable:

Given any vector $y \in R(V)$, is a given set of elements of y simultaneously bounded?

Thus, it is decidable if M is bounded for the Petri net $\mathcal{P}$. ■

We can now establish the following result:

Theorem 2.3.3 It is decidable if a given marking for a Petri net is live and bounded (i.e., if a given Petri net is LB).

Proof: Since we can decide if the given mark' ı bounded, we proceed to show that there exists a decision pro decide if a given bounded marking is live. For a bounded marking, the reachability set of the corresponding vector addition system V is finite. Hence, the

reachability diagram for V is finite. The marking M is live iff for any marking $M' \in \vec{M}$ there exists a firing sequence that fires every transition in the net. In terms of the reachability diagram, this means that starting at any vertex y_i and given a vector $w_k \in W$, there must exist a vector addition sequence that contains w_k.

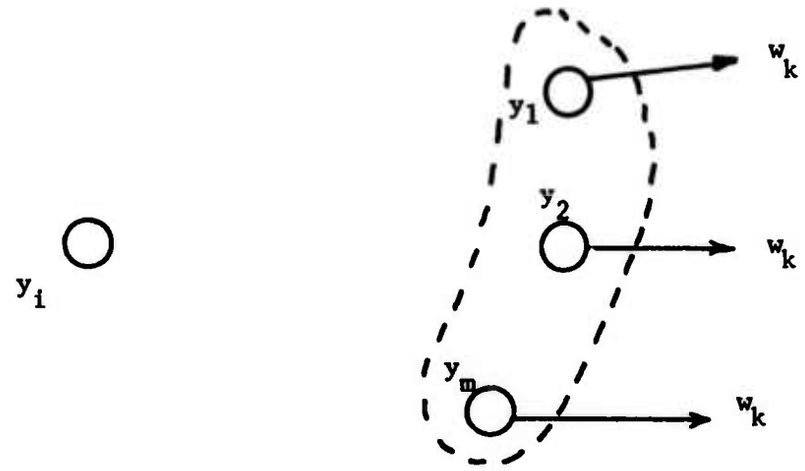


Figure 2.3.4

Consider figure 2.3.4. Find the set of vertices $\{y_1, y_2, \ldots y_m\}$ in the reachability diagram which have w_k as an outgoing arc. We have to find if there exists a directed path from y_i to at least one vertex in this set. Since the graph is finite, this can be done by exhaustion. The procedure is repeated for every vector $w_k \in W$, starting at the vertex y_i. We now choose all the vertices in turn and carry out this test. The total number of tests involved is $|W| \times |R(V)|$ where

$|W|$ = the number of vectors in the set W.

$|R(V)|$ = number of vectors in the reachability set of V.

Since both quantities are finite, the algorithm must terminate. This proves the required result. ∎

We now prove another theorem for a Petri net with a bounded marking.

Theorem 2.3.4 It is decidable if a bounded marking M for a Petri net $\mathcal{P}$ is persistent.

Proof: To check if a bounded marking is persistent, perform the following test for every marking in the marking class:

Let M' be a marking in the marking class $\vec{M}$, and let $\{t_1', t_2', \ldots t_n'\}$ be the set of enabled transitions. Let the firing of a transition $t_k' \in \{t_1', \ldots t_n'\}$ lead to the marking M_k', i.e., $M' \xrightarrow{t_k'} M_k'$. For the firing of each transition t_k', we check to see if all other enabled transitions $t_i' \neq t_k'$ are still enabled. If not, the marking M' is not persistent. If the marking M' is persistent for all of its enabled transitions then we check the next marking in the marking class, and so on until all markings $M' \in \vec{M}$ have been exhausted. The marking M is persistent iff all markings $M' \in \vec{M}$ are persistent for their enabled transitions. Since the total number of tests to be performed is finite and bounded, the algorithm must terminate. ∎

This concludes our discussion of decidable issues in Petri nets. We have so far looked at Petri nets in terms of their markings and their marking class. In the next chapter, we look at Petri nets in terms of their structure.

CHAPTER 3

RELEVANT RESULTS FROM PETRI NET THEORY

3.1 State Machine Decomposable Petri nets

In Section 2.1, we saw how an asynchronous concurrent system can be viewed as an ensemble of interacting components, where each component is structurally similar to the finite-state machine of automata theory. Each component has a finite number of allowable states; since we are concerned with systems which have no redundant operators (see Section 2.1), the Petri nets of interest are LB. We will use a restricted class of Petri nets known as LB SMD (State Machine Decomposable) Petri nets to model asynchronous concurrent systems. In this Section, we formally define the notions of state machine and SMD Petri nets.

Definition 3.1.1 A closed subnet Π' of a Petri net $\mathcal{P}$ is a strongly-connected Petri net $\langle P', T', A' \rangle$ where

$P' \subseteq P$ is a set of places,

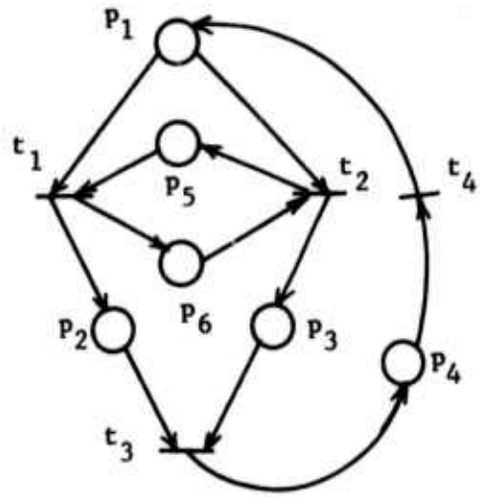
$T' \subseteq T$ is a set of transitions,

$A' \subseteq A$ is a set of arcs, such that

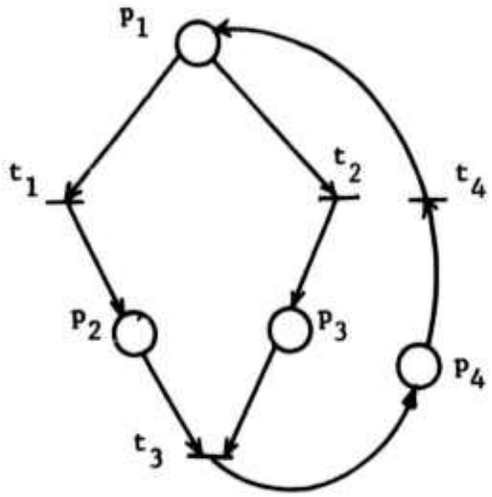
$\cdot P' = P' \cdot = T'$, and

$$A' = [(P' \times T') \cup (T' \times P')] \cap A.$$

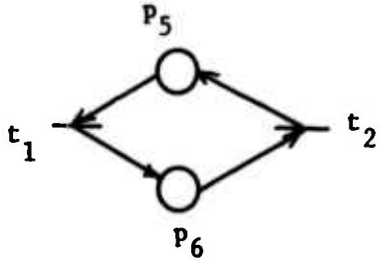
The Petri net N in Figure 3.1.1 has five closed subnets N_1, N_2, N_3, N_4 and N_5. Clearly, every strongly-connected Petri net is a closed subnet of itself, because the relation $\cdot P = P \cdot = T$ is trivially satisfied. We are interested in identifying components or parts of a system that can be represented by a Petri net, and for this reason we would like to define a minimal structure which is part of a Petri net and is still a



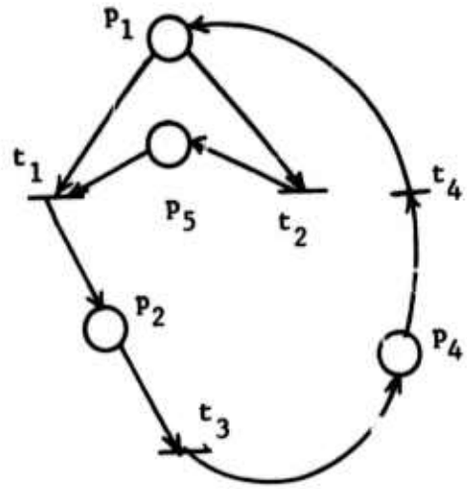
N



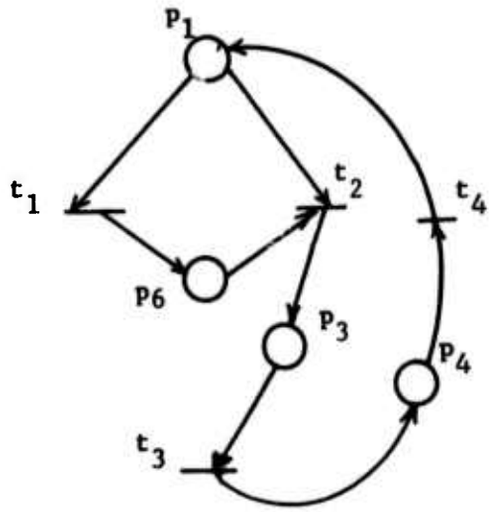
N_1



N_2

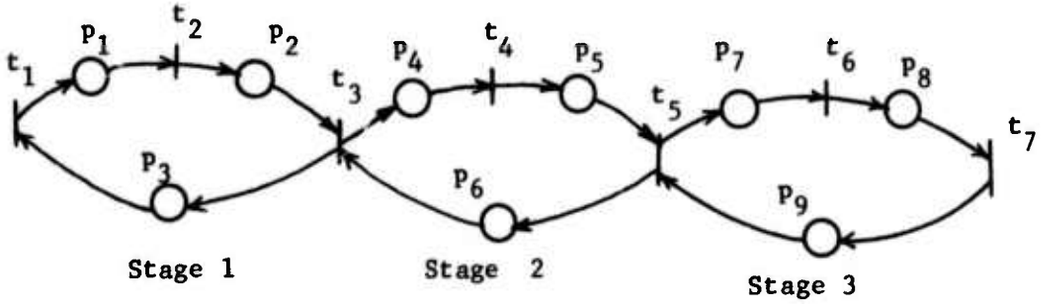


N_3

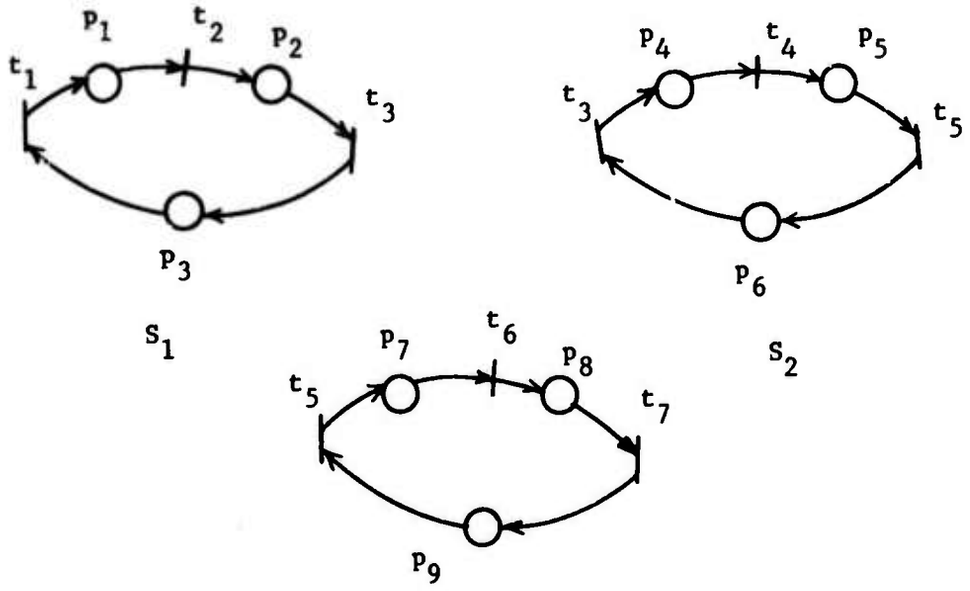


N_4

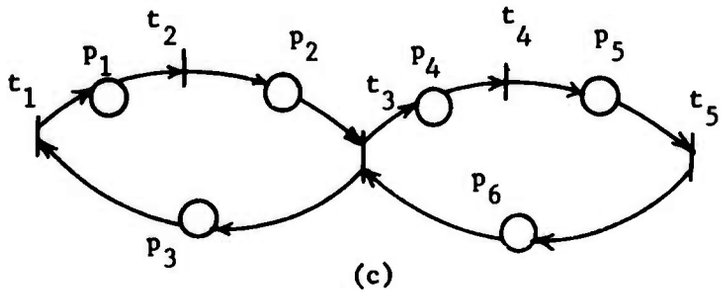
Figure 3.1.1



(a)



(b)



(c)

Figure 3.1.2

closed subnet.

Definition 3.1.2 A closed subnet is a minimal closed subnet if and only if no closed subnet can be obtained by deleting any portion of it.

Example 3.1.1 N is not a minimal closed subnet because by deleting appropriate places and/or transitions, the four closed subnets N_1, N_2, N_3 and N_4 are obtained. However, the closed subnets N_1, N_2, N_3 and N_4 are minimal closed subnets, as the reader can easily verify.

Example 3.1.2 Consider the Petri net model of a pipeline processor shown in Figure 3.1.2(a). The closed subnets representing each individual stage are minimal closed subnets (see Figure 3.1.2(b)).On the other hand, the closed subnet in Figure 3.1.2(c) is not minimal. All the Petri nets considered so far have the property that their minimal closed subnets have disjoint places. Lest the reader be under the impression that this is a requirement on minimal closed subnets, we would like to emphasize that this is not so. Consider the net in Figure 3.1.3(a). It has the minimal closed subnets S_1, S_2 and S_3 shown in Figure 3.1.3(b). We will now formally define the notion of state-machine based on a suitable structural restriction on Petri nets.

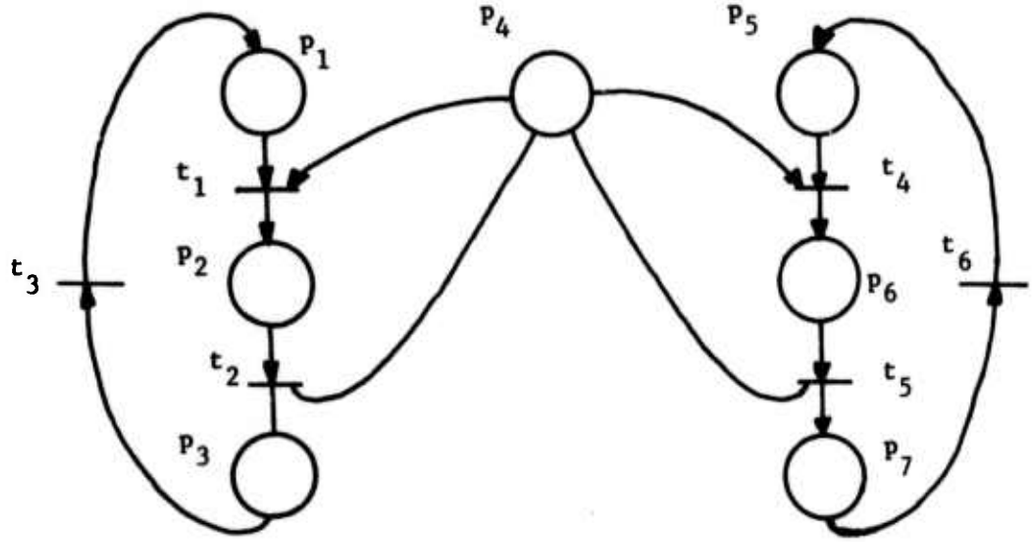
Definition 3.1.3 A Petri net $\mathcal{P}$ is a state-machine if and only if every transition has exactly one input place and exactly one output place.

Example 3.1.3 For an example of a state-machine, see Figure 3.1.4.

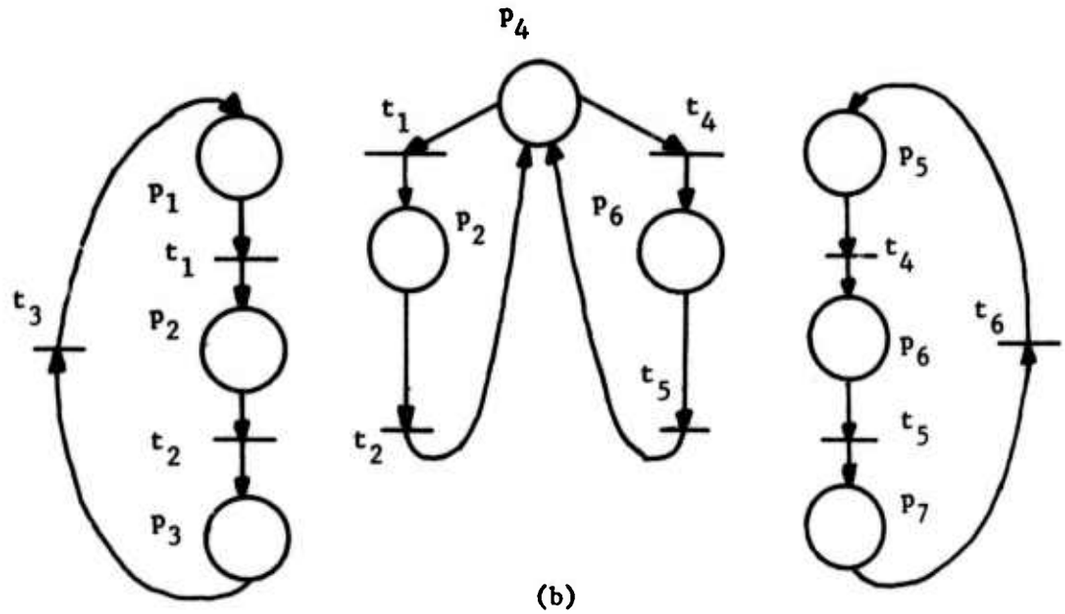
Definition 3.1.4 A Petri net $\mathcal{P}$ is said to be covered by a set of closed subnets $\{ \Pi_1, \Pi_2, \ldots \Pi_k \}$ if and only if

$$\mathcal{P} = \langle \bigcup_i P_i , \bigcup_i T_i , \bigcup_i A_i \rangle .$$

Example 3.1.4 The Petri net in Figure 3.1.2(a) is covered by the set



(a)



(b)

Figure 3.1.3

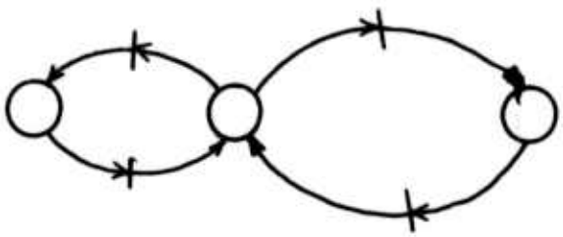


Figure 3.1.4 A State Machine

of closed subnets $\{S_1, S_2, S_3\}$. Similarly, the net in Figure 3.1.3(a) is covered by the subnets of Figure 3.1.3(b).

Definition 3.1.5 A Petri net $\mathcal{P}$ is SMD (State Machine Decomposable) iff every minimal closed subnet is a state machine and there exists a set of state machines $\{S_1, \ldots S_k\}$ which covers the net.

Example 3.1.5 The Petri nets in Figures 3.1.2(a) and 3.1.3(b) are SMD.

On the other hand, the net in Figure 3.1.1(a) is not SMD, because the minimal closed subnets N_1, N_3 and N_4 are not state machines.

Definition 3.1.6 The token content of a Petri net $\langle P, T, A \rangle$ with a marking M is the number of tokens on all places in the net, and is given by $N(P) = \sum_{p_i \in P} M(p_i)$

Lemma 3.1.1 The token content of a marked state machine is invariant under transition firings.

Proof Suppose we consider a state machine $\mathcal{P} = \langle P, T, A \rangle$ with mar-

king M. The only way in which the token content N(p) of the state machine can change is by the firing of a transition. However, the firing of a transition removes exactly one token from the net and adds exactly one token to the net. Thus, the firing of a transition does not change the number of tokens on the net.

■

Theorem 3.1.1 Every marking for an SMD Petri net is bounded.

Proof An SMD Petri net has a finite number of state machine components. Let these components be $S_1,\ldots S_k$. Also, by Lemma 3.1.1, the token content of each state machine is invariant. Let the token content of the nth state machine be $N(P_n)$. Now, recall the definition of a bounded marking for a Petri net. We have to show that for each place p in the net, there exists an integer Z(p) such that $M(p) \leq Z(p)$ for each M in $\vec{M}$.

Let $S_{p1},\ldots S_{pm}$ be the set of state machines which contain place p. Then, if $N(S_{p1}),\ldots N(S_{pm})$ are their respective token contents, we see that $Z(p) \leq \min[\, N(S_{p1}),\ldots N(S_{pm})]$. If Z(p) were greater than $\min [\, N(S_{p1}),\ldots N(S_{pm})]$, it would imply that there exists a state machine S_k such that $Z(p) > N(S_k)$, which is impossible. Hence, for each place p in the net, we can find an integer Z(p) which bounds its token content. This proves the Theorem.

■

Corollary 3.1.1.1 It is decidable if any given marking M for an SMD Petri net $\mathcal{P}$ is live.

Proof By Theorem 2.3.3, it is decidable if a bounded marking M for Petri net $\mathcal{P}$ is live. Also, since any given marking for $\mathcal{P}$ is bounded, the desired result follows immediately.

■

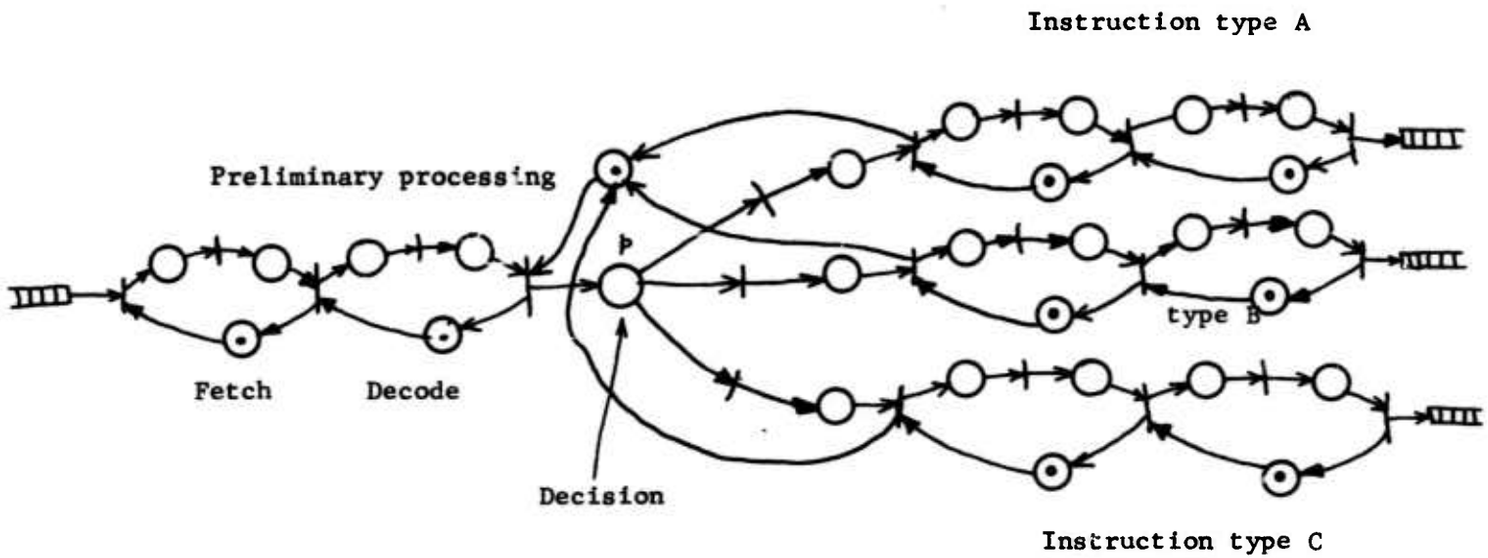


Figure 3.1.5 SMD Petri net model of a pipeline processor with decisions

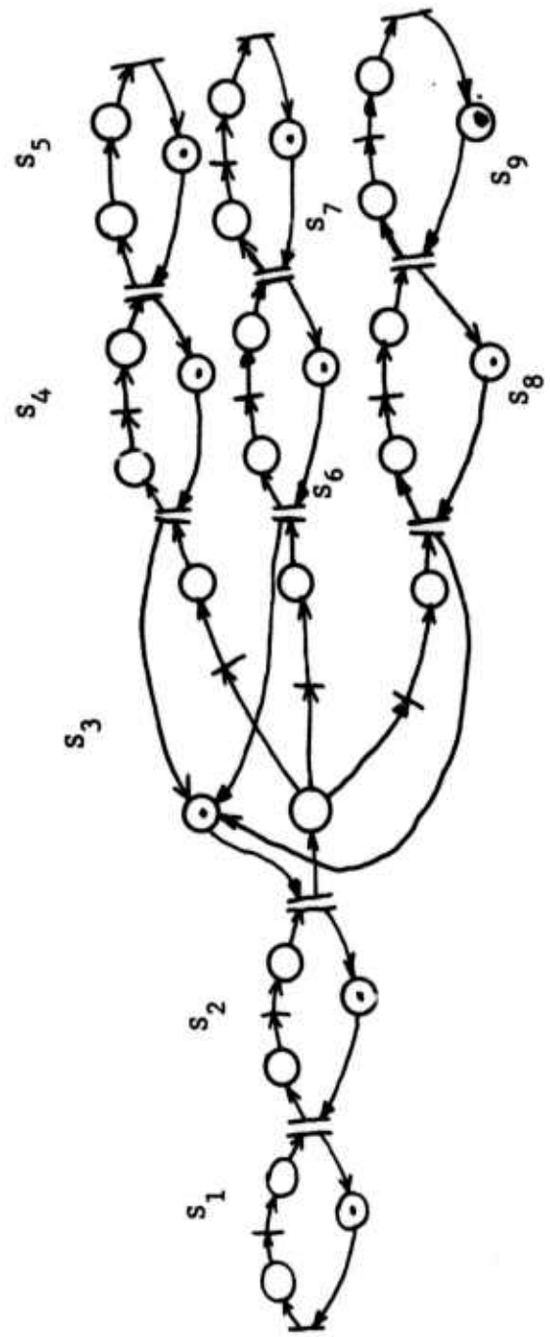


Figure 3.1.6 State Machine components of net in Figure 3.1.5

This result is useful because it tells us that we can decide if a given marking M for a Petri net $\mathcal{P}$ is live, but it does not give us an effective procedure for constructing a live marking for $\mathcal{P}$. We will look at this issue in Sections 3.3 and 3.5. Before we proceed to the next Section, we give the reader an example which illustrates the descriptive power of SMD Petri nets. The example of the pipeline processor we gave in Figure 3.1.2(a) was that of a deterministic system. In Figure 3.1.5, we give an SMD Petri net model of the pipeline processor for the computer system with three instruction types which we discussed in Section 1.2 (see Figure 1.2.3). Figure 3.1.6 shows the net in Figure 3.1.5 split up into its state machine components, which are labelled S_1 through S_9. The reader will note that the decision about processing an instruction after it has been decoded can be made in the state machine S_3. Place p in state machine S_3 has three output arcs, one for each instruction type. We said in Section 1.2 that in order to estimate the processing rate of such a pipelined processor, we must know the relative frequency of occurrence of each of the instruction types A, B and C. In other words, we must know the relative frequency of occurrence (or the probability) of each of the output transitions of place p. Thus, on the average, the rates at which each instruction type occurs will be reflected in the number of times each of the transitions t_a, t_b and t_c fires in the long run. We continue this train of thought in the next Section, where we present the important notion of consistency. In the meantime, the reader should explore the descriptive power of SMD Petri nets by constructing examples of his own.

3.2 Consistent Petri Nets

In this Section, we will look at the number of times any transition in an LB Petri net can fire relative to the other transitions in the net. We will first need some definitions.

Definition 3.2.1 A current assignment for a Petri net $\mathcal{P} = \langle P, T, A \rangle$ is a function Φ that assigns to each transition $t_i \in T$ a positive rational number φ_i called its current. A current assignment for a Petri net must satisfy the following two constraints:

(1) Every arc carries a current equal to that associated with the transition it is connected to.

(2) At every place, the sum of the currents on the input arcs must equal the sum of the currents on the output arcs (i.e., Kirchoff's current law).

Definition 3.2.2 A Petri net is consistent iff it has a current assignment Φ with $\varphi_i > 0$ for each $t_i \in T$.

Checking the Consistency of a Petri net

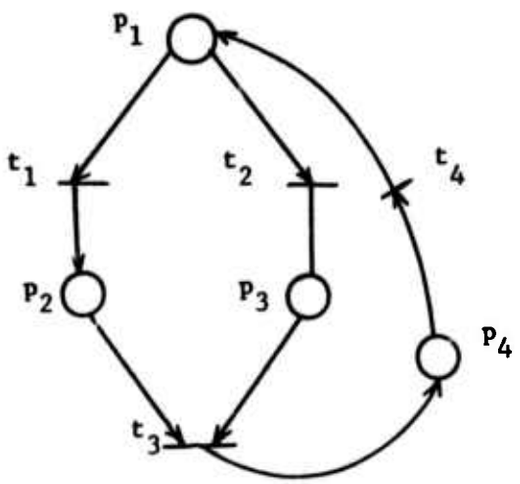


Figure 3.2.1

Consider the Petri net shown in Figure 3.2.1. Let each transition t_i in the net be assigned a current φ_i. For each place, we can write an equation that specifies a constraint on the input and output currents.

$$P_1: \quad -\varphi_1 \quad -\varphi_2 \quad +\varphi_4 = 0 \quad \ldots \quad (1)$$

$$P_2: \quad +\varphi_1 \quad -\varphi_3 = 0 \quad \ldots \quad (2)$$

$$P_3: \quad +\varphi_2 \quad -\varphi_3 = 0 \quad \ldots \quad (3)$$

$$P_4: \quad +\varphi_3 \quad -\varphi_4 = 0 \quad \ldots \quad (4)$$

From Equations (2), (3) and (4) we get,

$$\varphi_1 = \varphi_2 = \varphi_3 = \varphi_4 = \varphi \quad \text{(say)}.$$

This violates Equation (1), which requires that

$$\varphi_4 = \varphi_1 + \varphi_2 = 2\varphi.$$

We conclude that the net in Figure 3.2.1 is not consistent, or is <u>inconsistent</u>. A slight modification of this net leads to the consistent net of Figure 3.2.2.

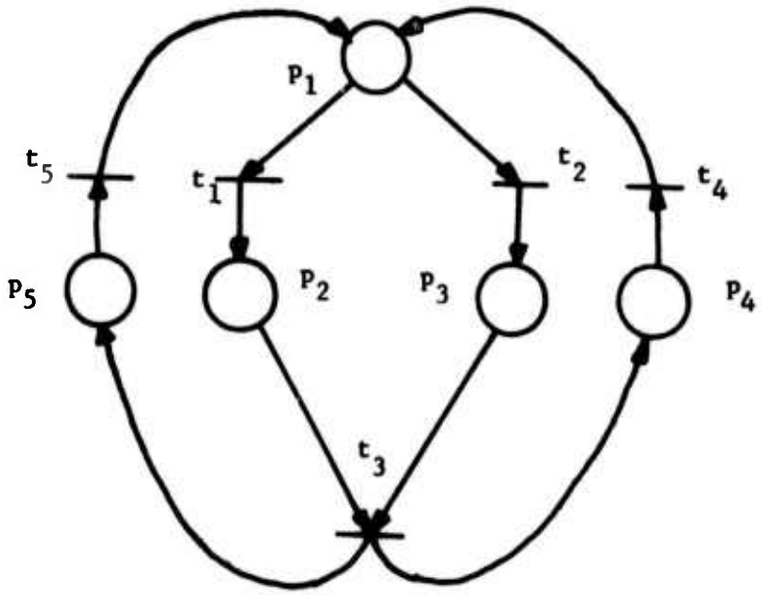


Figure 3.2.2

The reader will note that all we have done is to feed another current of magnitude ϕ into the place p_1. The equations become

$$p_1 : \quad -\phi_1 - \phi_2 + \phi_4 + \phi_5 = 0$$

$$p_2 : \quad +\phi_1 - \phi_3 = 0$$

$$p_3 : \quad +\phi_2 - \phi_3 = 0$$

$$p_4 : \quad +\phi_3 + \phi_4 = 0$$

$$p_5 : \quad +\phi_3 - \phi_5 = 0$$

These equations are, indeed, consistent and we get the following consistent current assignment:

$\phi_1 = \phi_2 = \phi_3 = \phi_4 = \phi_5 = \phi$, where ϕ is any non-zero real number.

Consider a consistent current assignment for a Petri net $\mathcal{P}$. Multiply all currents by the least common multiple of their denominators, and divide each resulting current by their greatest common divisor. The resulting currents are said to form a <u>minimal integer consistent current assignment</u> for $\mathcal{P}$. The reader will note that consistency is a purely structural property of a Petri net. The following Theorem and discussion explain the relationship between the notion of consistency and the structure of LB Petri nets. The ideas in the following material have been partly inspired by Baker [B4].

<u>Definition 3.2.3</u> Let $\mathcal{P}$ be a Petri net with marking M. A cyclic firing sequence is a firing sequence σ which fires every transition of $\mathcal{P}$ at least once and brings the marking of the net back to M.

Definition 3.2.4 The firing count of a transition t_i in a firing sequence σ is the number of occurrences of t_i in σ . The firing vector Ψ of a firing sequence σ is a vector whose ith element $\psi(i)$ is the firing count of transition t_i.

Theorem 3.2.1 A Petri net $\mathcal{P}$ is consistent if and only if there exists an initial marking M for which there exists a cyclic firing sequence.

Proof: Necessity

Consider the minimal integer consistent current assignment derived from the given current assignment. Let this current assignment be Φ . We will show how to construct a finite firing sequence σ whose firing vector Ψ is such that $\psi(i)$ is the current through transition t_i. We construct the marking M as follows: $M(p_k)$ must be big enough so that firing its output transitions $t_i \in p_k^{\cdot}$ $\psi(i)$ times does not drive the token load of p_k negative, i.e., $M(p_k) = \sum_{t_i \in p_k^{\cdot}} \psi(i)$.

Under this marking, a cyclic firing sequence σ is given by

$$= t_1^{\psi(1)} \ldots t_n^{\psi(n)}$$

where $t_1 \ldots t_n$ are the transitions in the Petri net $\mathcal{P}$. Since Φ is a consistent current assignment, the firing sequence σ is such that for any place p_k, the number of tokens removed by σ is equal to the number of tokens added by σ to p_k. The marking M' after σ has occurred is the same as the marking M before the occurrence of σ , so that σ is a cyclic firing sequence.

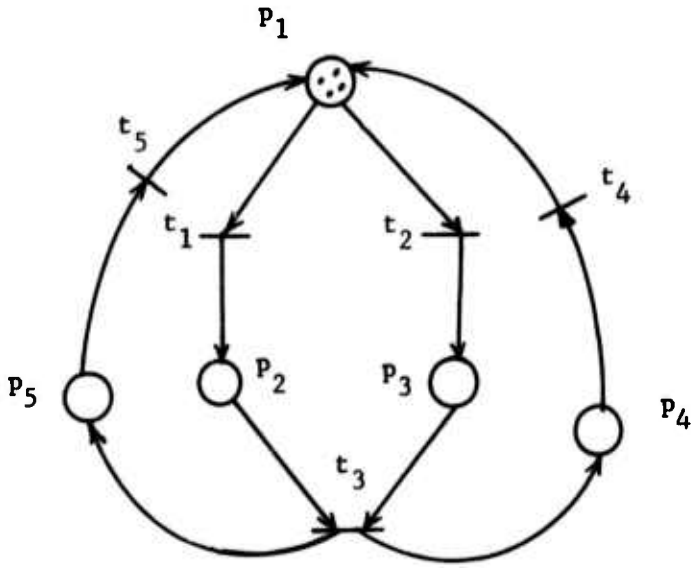
Sufficiency Let σ be a cyclic firing sequence for the net, and let Ψ be the firing vector of σ. With each transition t_i, associate an integer valued current equal to $\psi(i)$. By definition of current, each input and output arc of t_i has a current $\psi(i)$ associated with it. Under this assignment, every transition has been assigned a non-zero integer current, and each transition has been assigned a unique current. Now consider any place p_j. By the definition of a cyclic firing sequence, p_j has the same number of tokens before and after the cyclic firing sequence has occurred. This implies that the sum of the tokens entering place p_j is the same as the sum of the tokens leaving p_j, i.e., the sum of the input currents equals the sum of the output currents for every place p_j. This is the definition of a consistent current assignment.

■

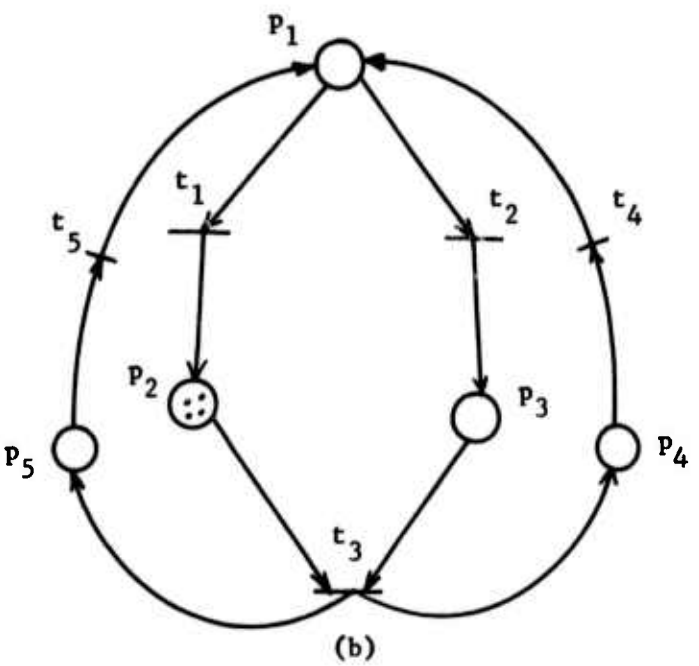
We have pointed out in Chapter 2 that LB Petri nets are the only ones of interest to us. We now establish the connection between consistency and LB Petri nets.

Theorem 3.2.2 A Petri net $\mathcal{P}$ which has a live, bounded marking M is consistent.

Proof We have to show the existence of a cyclic firing sequence which fires every transition at least once. Since the marking M is bounded, its reachability diagram is a finite-state machine. Also, since M is live, there must exist a strongly-connected portion of the reachability diagram that contains every transition name at least once. Let M' be some node (marking) in the strongly-connected portion of the reachability diagram.



(a)



(b)

Figure 3.2.3

We know that in a strongly-connected graph, there exists a chain (i.e., a sequence of directed arcs, all arcs directed in the same sense) that starts and ends at M' and which goes through every arc once. This chain can be used to describe a cyclic firing sequence, and since $\mathcal{P}$ has a live marking, every transition name must be included at least once in this firing sequence.

As a Corollary to Theorems 3.2.1 and 3.2.2, we have the following:

Corollary 3.2.1 If a Petri net $\mathcal{P}$ is LB, then there exists an initial marking M for which there exists a cyclic firing sequence.

Proof Follows by combining Theorems 3.2.1 and 3.2.2.

For completeness, we mention that the converse of Theorem 3.2.2 is not true. The net in Figure 3.2.3 provides the necessary counter-example. The reader can verify that any initial marking of the net can lead to a marking like the one shown in Figure 3.2.3(b), at which no transition is enabled.

We would like to consider the connection between consistency and the structure of LB SMD Petri nets, because that is the class of Petri nets we will use for modelling asynchronous systems. We will look at the concepts of cyclic firing sequence and consistency for SMD Petri nets in some depth in Section 3.5. Before we can do this, we look at a deterministic subclass of Petri nets known as event graphs. We do this in Section 3.3. In Section 3.4, we complete our study of those aspects of Petri net theory which are relevant to this thesis.

3.3 Event Graphs

Event graphs are what Commoner and Holt have called Marked Graphs[C1]. We have chosen to call them event graphs because we would like to preserve the distinction between the structure and marking of a Petri net. In this Section, we would like to define event graphs and to state some results that we will need in Chapter 4, where we look at the idea of timing relationships between activities in deterministic systems.

Definition 3.3.1 An event graph is an SMD Petri net in which every place has exactly one input transition and exactly one output transition.

Example of an event graph

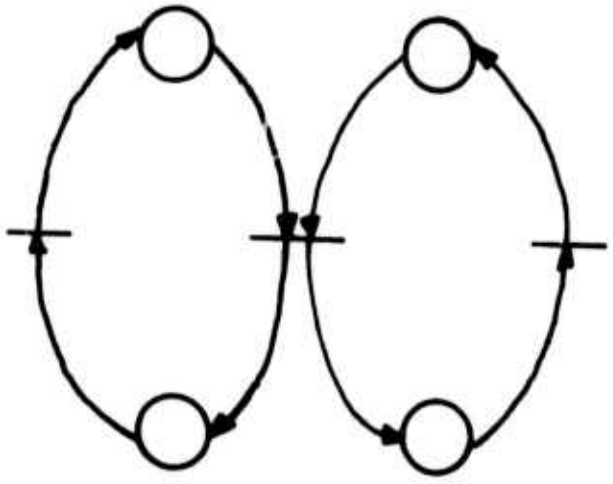


Figure 3.3.1 An event graph

Recall the definition of conflict given in Section 2.2. It should be noted that no transitions in the net are in conflict. Every marking for an event graph is persistent, and every event graph is a deterministic Petri net. The structure of event graphs enables us to define them in

the following convenient fashion:

Definition 3.3.2 An event graph is a strongly-connected directed graph

$G = \langle V, A \rangle$

where $V = \{v_1, \ldots v_m\}$ is the set of transitions

$A = \{a_{ij}\} \subseteq V \times V$ is the set of arcs, where arc a_{ij} connects transition v_i to v_j.

A marking of an event graph is a function that assigns to every arc a_{ij} in the net a non-negative integer called the token load of the arc. Note that in this definition of event graphs, an arc corresponds to a place together with its incoming and outgoing arc in Definition 3.3.1 (see Figure 3.3.2).

An arc in Definition 3.3.2 | Corresponding structure in Definition 3.3.1.

Figure 3.3.2

Figure 3.3.3 gives an event graph corresponding to the one in Figure 3.3.1.

Event graphs are seen to be conflict-free SMD Petri nets in which each simple circuit is a state-machine component.

Theorem 3.3.1 (due to Commoner and Genrich)

A marking for an event graph G is live if and only if the token content

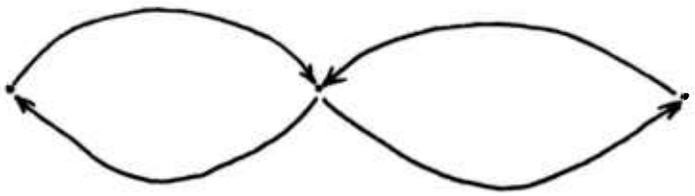


Figure 3.3.3

of every simple circuit is non-zero.

Proof

Necessity If the token content of some simple circuit is zero, no transition in this circuit can be fired; since the token content does not change if other vertices are fired (lemma 3.1.1), no vertex on this circuit can be enabled through transition firings.

Sufficiency Assume that the token content of every simple circuit is non-zero. Let v_i be any transition in the net. Consider the unmarked arcs entering v_i.

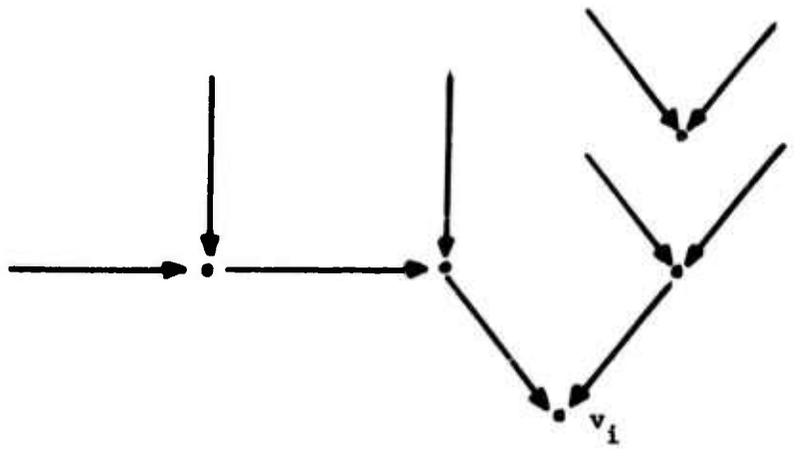


Figure 3.3.4

If there are none, the vertex is enabled. If not, consider the vertices from which these arcs emanate. If each of these is enabled, then clearly v_i will become enabled after all of them are fired. If some are not, consider the unmarked arcs entering them, etc. As we continue this back-tracking, we are selecting a subgraph of G which consists of v_i, the vertices from which these arcs emanate, the unmarked arcs entering them, etc. This process must terminate, since G is finite. Now this subgraph

must be circuit-free (i.e., a tree, as shown in Figure 3.2.4), since there are no token-free simple circuits. Thus, the subgraph must have at least one transition that has no incoming arcs which belong to the subgraph. This vertex can be fired at the present marking of G. After firing it, the subgraph of token-free back-tracking from v_i is reduced by one transition. By repeating this process, we can enable v_i. ■

There is one more result we will need in connection with event graphs, the proof of which is given in [C1].

Theorem 3.3.2 A live marking for an event graph is safe if and only if every arc is contained in some simple circuit containing exactly one token.

Example Figure 3.2.5 gives an example of an event graph with a live, safe marking. Note that every circuit has at least one token on it, and that every arc is contained in some one-token circuit.

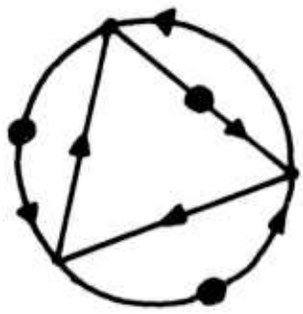


Figure 3.3.5

It is easy to see that event graphs have a live, safe marking that is persistent, and are thus deterministic nets. However, event graphs are not the only Petri nets with this property. In the next Section, we study a larger class of deterministic Petri nets.

3.4 LSP Petri nets (Petri nets with a live, safe and persistent marking)

The class of Petri nets is a large one, but we will confine our attention to LSP Petri nets because their behavior can be represented by Petri nets whose structure is very similar to that of event graphs. The marked Petri net in Figure 3.4.1 is a good example of a Petri net which has a live, safe and persistent marking but which is not an event graph. The reader can verify that this net is LSP by drawing its reachability diagram, which we give in Figure 3.4.2. It is seen that in no marking in the forward marking class can there be more than one token on any place. Furthermore, at each marking exactly one transition is seen to be enabled. This verifies our claim that the net is LSP. In general, one can determine if a marked Petri net is LSP by using the results of Section 2.3. As we have pointed out before, the reachability diagram of a marked Petri net obscures the concurrency that is inherent in the system it represents, and this is precisely what happened when we drew the reachability diagram for the net in Figure 3.4.1. We now present an alternative means of representing the operation of LSP Petri nets. We will do this by constructing for LSP Petri nets a graph known as a behavior graph, and shall explain its construction by means of an example.

Representing the Behavior of LSP Petri nets: Consider the LSP Petri net $\mathcal{P}$ shown in Figure 3.4.3(a). We begin by drawing and labelling the set $P_1 \subseteq P$ of marked places in $\mathcal{P}$, (in this case $\{p_1, p_2, p_6\}$). Let $T_1 = \{t_1, t_2\}$ be the set of enabled transitions corresponding to P_1, and note that P_1 corresponds to the initial marking of $\mathcal{P}$. Let P_2 be the set of marked places that results when all transitions in T_1 are fired. Draw

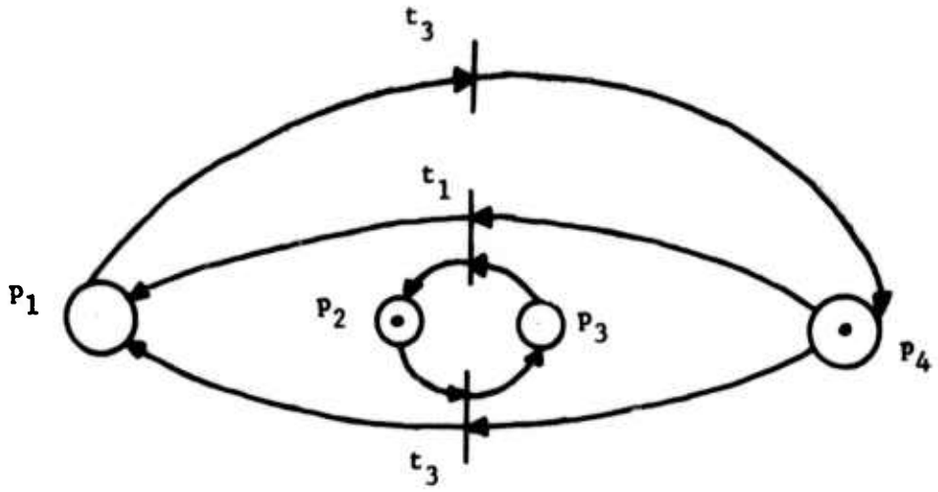


Figure 3.4.1

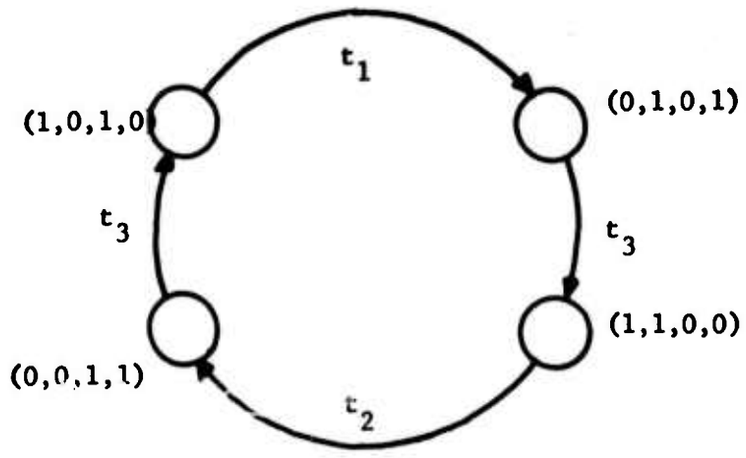


Figure 3.4.2

all arcs $P_1 \times T_1$ that are contained in $P \times T$. Draw the places $P_1' = T_1'^{\cdot}$. Draw all the arcs $T_1 \times P_1'$ which are contained in $T \times P$.

Define $P_2 = (P_1 - \dot{T}_1) \times T_1^{\cdot}$.

This process of constructing the marked places P_{k+1} that results when all enabled transitions for P_k are fired is called <u>extending</u> the behavior graph from P_k to P_{k+1}. Since $\mathcal{P}$ has a live marking, its behavior graph can be extended indefinitely. Figure 3.4.3(b) shows the behavior graph of $\mathcal{P}$. We now make some definitions.

<u>Definition 3.4.1</u> A <u>chain</u> in a behavior graph is any directed path in it.

<u>Example</u> $p_1t_1p_3t_3p_5t_4\ldots$ is a chain in $\mathcal{B}$.

<u>Definition 3.4.2</u> A <u>slice</u> of a behavior graph $\mathcal{B}$ is a set of places that forms a cut-set of $\mathcal{B}$.

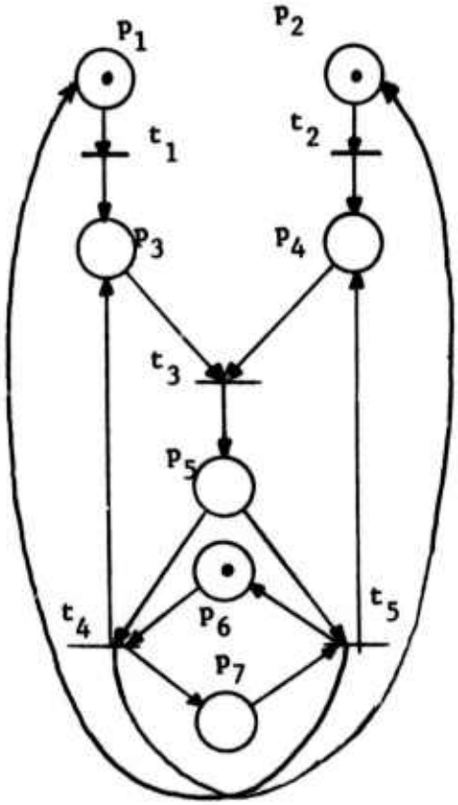
<u>Note</u> In a cut-set, no two elements of the set can belong to the same chain.

<u>Example</u> $P = \{p_1, p_2, p_6\}$ is a slice of $\mathcal{B}$.

Each slice of a behavior graph corresponds to a marking of the LSP Petri net.

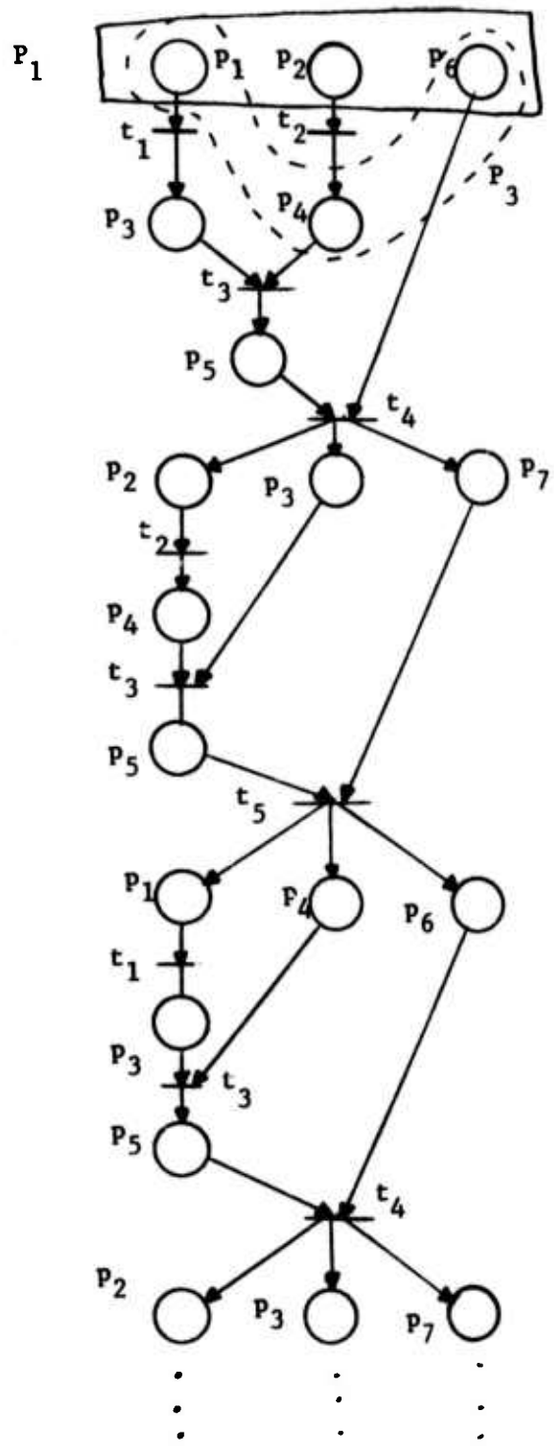
Lemma 3.4.1 Each place in a behavior graph for an LSP Petri net must have exactly one input transition and exactly one output transition.

<u>Proof</u> Suppose some place p in the behavior graph $\mathcal{B}$ has more than one input transition. Then, there must exist a marking M' in the marking class of



LSP Petri net $\mathcal{P}$.

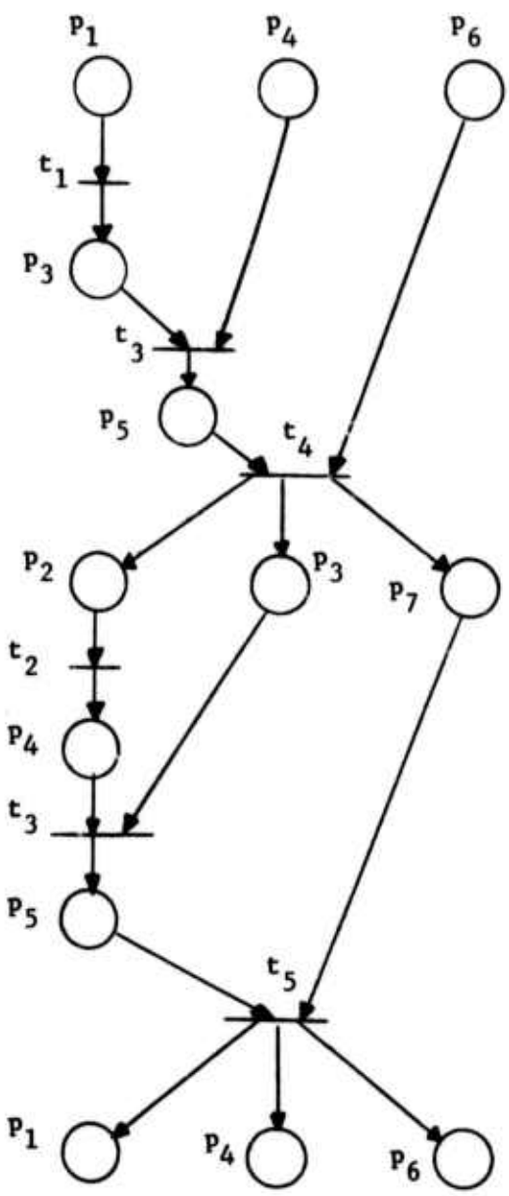
(a)



Behavior Graph $\mathcal{B}$.

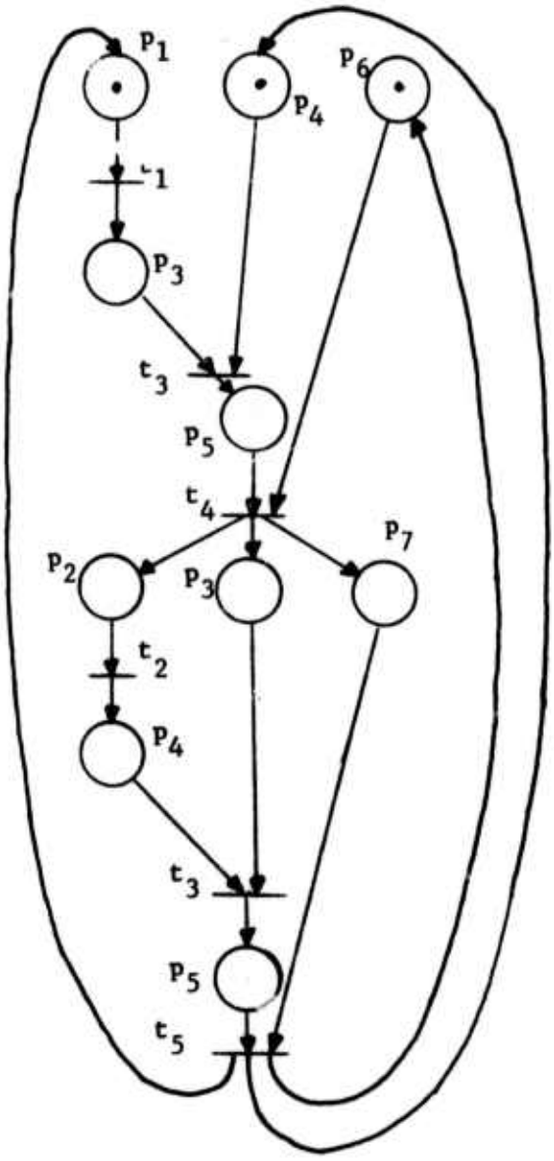
(b)

Figure 3.4.3



Cyclic Frustrum of $\mathcal{B}$.

(c)



Steady-state equivalent net $\mathcal{S}$.

(d)

Figure 3.4.3

the net $\mathcal{P}$ in which more than one token is placed on p_1, implying that $\mathcal{P}$ has a marking that is not safe. Hence, every place in $\mathcal{B}$ has exactly one input transition.

Also, suppose some place p in $\mathcal{P}$ has more than one output transition. This implies that there exists a marking M' in the forward marking class for $\mathcal{P}$ at which more than one output transition is enabled. This violates the assumption that $\mathcal{P}$ has a persistent marking. Hence, each place in $\mathcal{P}$ has exactly one input and exactly one output transition.

Lemma 3.4.2 There exists a slice in the behavior graph $\mathcal{B}$ of an LSP Petri net $\mathcal{P}$ that occurs repeatedly.

Proof Each slice of $\mathcal{B}$ corresponds to a marking of $\mathcal{P}$. Since $\mathcal{P}$ has a safe marking, the number of distinct markings in the marking class for the net is finite. Therefore, since the behavior graph is infinite, there must exist a slice in $\mathcal{B}$ that occurs repeatedly.

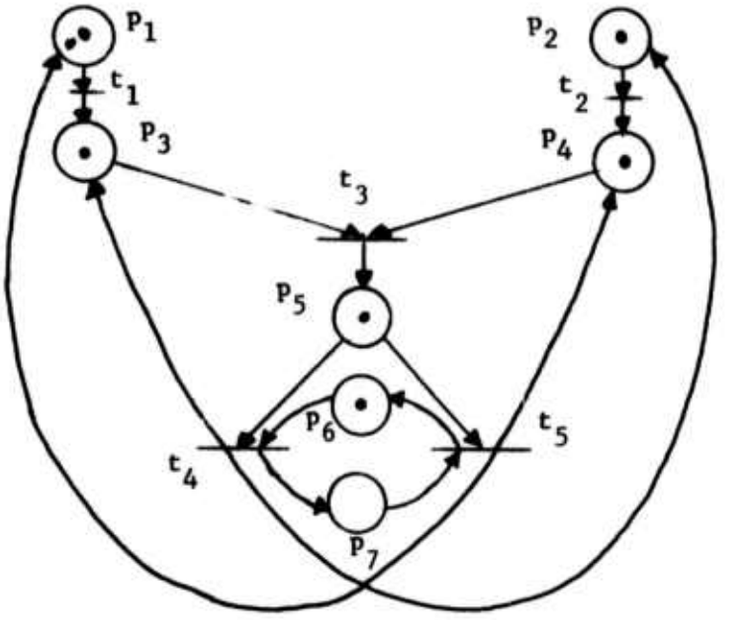
The reader is now asked to refer to Figure 3.4.3. The slice P_1 represents the initial marking of $\mathcal{P}$. P_1 does not occur again in $\mathcal{B}$. A behavior graph that has slices which do not repeat is said to have a transient, and such slices are termed transient slices. In terms of the Petri net $\mathcal{P}$, a transient slice represents a marking that cannot be reached after an initial occurrence. The slice P_3 (shown dotted) is a slice which has repeated occurrences in $\mathcal{B}$. A cyclic frustrum of $\mathcal{B}$ is the portion of $\mathcal{B}$ between two consecutive occurrences of some repeated slice. In Figure 3.4.3(c), we show a cyclic frustrum of $\mathcal{B}$ bounded by consecutive occurrences of the slice $\{p_1, p_4, p_6\}$ in $\mathcal{B}$. Since $\mathcal{B}$ is derived from a persistent marking of $\mathcal{P}$, only one way is possible of extending this

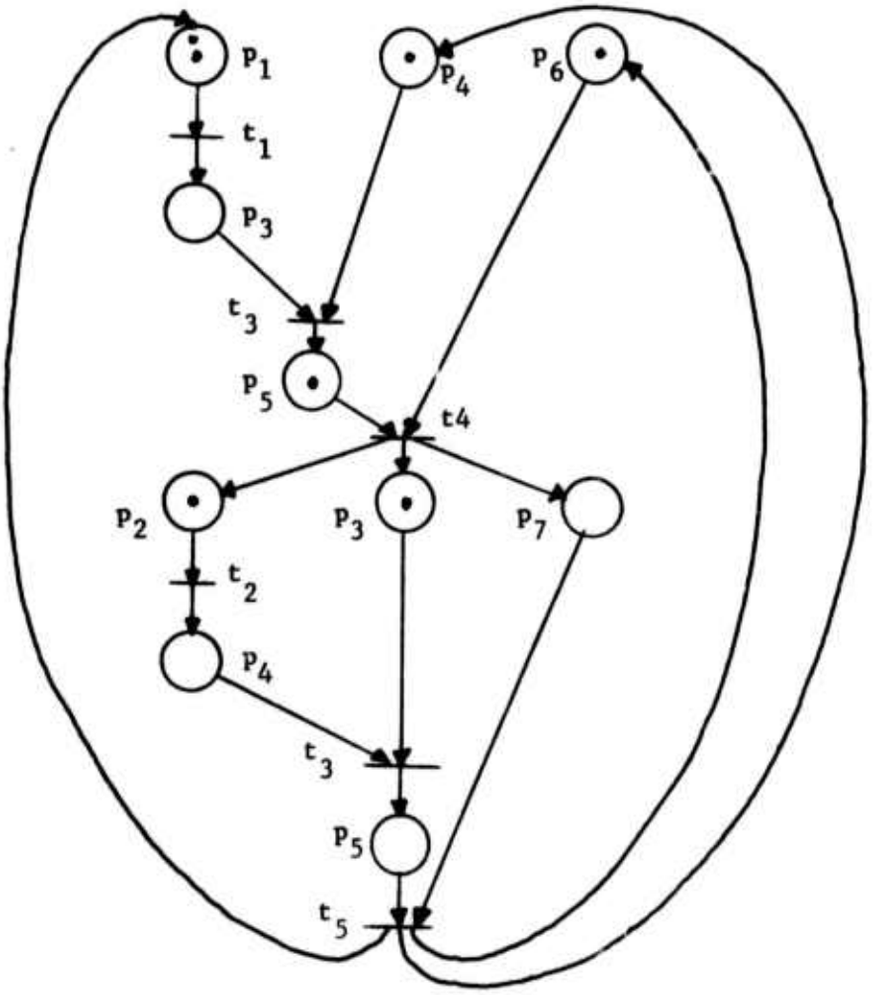
to $\{p_3, p_4, p_6\}$. In general, for any slice P_k of the behavior graph, there exists a unique extension P_{k+1}. Thus, every cyclic frustrum bounded at each end by the slice $\{p_1, p_4, p_6\}$ is identical to every other cyclic frustrum bounded at each end by this slice. Hence, instead of drawing an infinite behavior graph $\mathcal{B}$, we will choose some cyclic frustrum of $\mathcal{B}$ and then coalesce corresponding slices together in the initial and terminal slice of the cyclic frustrum. The net so obtained is termed the steady-state equivalent net of $\mathcal{P}$, and is shown in Figure 3.4.3(d). The marking M for the steady-state equivalent net $\mathcal{S}$ is obtained by putting one token on each place in the initial slice. The construction of this net is such that the set of firing sequences of $\mathcal{S}$ is identical to that of the net $\mathcal{P}$. Also, the reachability diagram of $\mathcal{S}$ is contained in that for $\mathcal{P}$, there possibly being some extra states in $\mathcal{P}$ corresponding to the transient. The reader can see this from the graph $\mathcal{B}$, and it is not necessary to construct the reachability diagram of $\mathcal{P}$.

We have shown that in $\mathcal{B}$, and hence in $\mathcal{S}$, each place has exactly one input transition and exactly one output transition. $\mathcal{S}$ thus has the structure of an event graph, with the difference that certain place and transition names occur more than once in it. Transition t_3 and places p_3 and p_4 occur twice, for example. Such an event graph is termed a multiply-labelled event graph. We will not define multiply-labelled event graphs formally, but will merely say that they are event graphs in which certain places and transitions have repeated occurrences (or instances).

The LSP Petri net $\mathcal{P}$ has the property that it is possible to add tokens to certain places and still have a marking that is live and bounded, but no longer safe. Tokens can be added to all places except p_6 and p_7, such

(a)





(b)

Figure 3.4.4

that the resulting marking is live, bounded and persistent. The steady state equivalent net for $\mathcal{P}$ with this new marking is simply the net $\mathcal{S}$; the marking for $\mathcal{S}$ is constructed by adding the same number of tokens to a place in $\mathcal{S}$ as were added to the corresponding place in $\mathcal{P}$. If there are multiple instances of a place in $\mathcal{S}$, then tokens can be added to any of those instances, provided the sum of the tokens added to all instances of a place equals the number of tokens added to the corresponding place in $\mathcal{P}$. Figure 3.4.4(a) shows a live, bounded, persistent marking for and Figure 3.4.4(b) its corresponding steady-state equivalent net $\mathcal{S}$.

We should mention that there exist Petri nets which have a live, bounded marking but no live, safe marking. Figure 3.4.5 shows such a Petri net. Thus, the preceding technique cannot be used to construct a steady-state equivalent net.

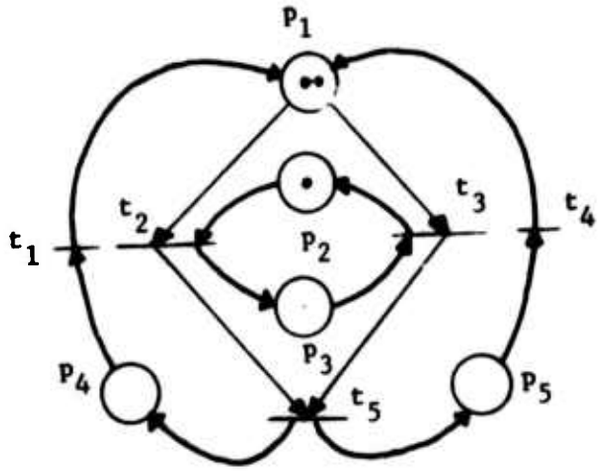


Figure 3.4.5

3.5 The Structure and Consistency of SMD Petri nets

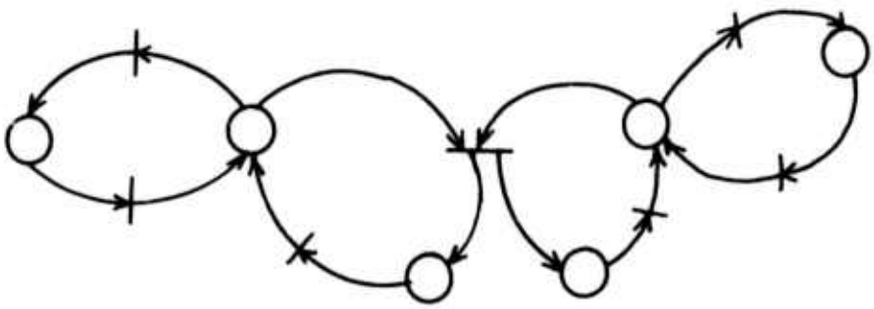
3.5.1 Existence of a live marking for an SMD Petri net

In Section 3.3, we saw that for event graphs a necessary and sufficient condition for a marking to be live is that there be at least one token on each simple circuit. Since a simple circuit in an event graph corresponds to a state machine in an SMD Petri net, the reader may be tempted to ask if we can get a live marking for an SMD net by adding at least one token on each state machine component. The nets given in Figure 2.1.9(c) and 2.1.10(c) are counter-examples to this conjecture. In Figure 2.1.9(c), a subset of the transitions in the net can never be fired, whereas in Figure 2.1.10(c), no transition can be fired. Both are examples of Petri nets which do not have a live marking. We now give a necessary condition for a marking M for an SMD Petri net $\mathcal{P}$ to be live.

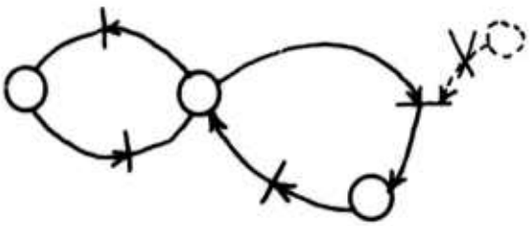
Theorem 3.5.1 A marking M for an SMD Petri net $\mathcal{P}$ is live only if the token content of every state machine is non-zero.

Proof We will show that if the token content of some state machine is zero, then no transition in it can be fired. Without loss of generality, let S_1 be a state machine component of $\mathcal{P}$ and let t_i be a transition in it, i.e., if $S_1 = \langle P_1, T_1, A_1 \rangle$ then $t_i \in T_1$. By hypothesis, the input place p_j of transition t_i in state machine S_1 must be unmarked. Also, by Lemma 3.1.1, it must stay unmarked. This implies that there is no marking $M' \in \vec{M}$ at which transition t_i can be fired. Hence, the marking M is not live. Note that no transition in S_1 can be fired at any marking $M' \in \vec{M}$. ■

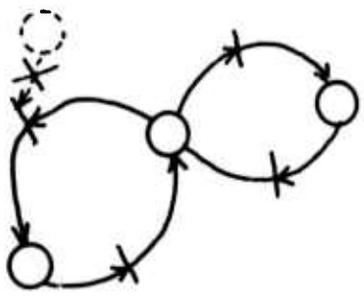
Hack[H5] has shown that a subclass of SMD Petri nets called SMA (state machine allocatable) Petri nets are similar to marked graphs in that any



(a)



Reduced net No. 1



Reduced net No. 2

(b)

Figure 3.5.1

marking that puts at least one token on every state machine is live. We proceed to describe the structure of these nets in what follows. The material in Subsection 3.5.2 is taken from Hack[h1, H5].

3.5.2 State Machine Allocatable Petri nets

Definition 3.5.1 A state machine allocation over a Petri net $\langle P,T,A \rangle$ is a function $B: T \longrightarrow P$ such that

$$\forall\, t \in T \quad B(t) \in {}^{\bullet}t.$$

Informally, this means that for each transition in the net, we pick one of its input places and ignore the others. Given such a state machine allocation, we perform a reduction by deleting certain places and transitions in the following manner:

Step 1 Delete all places for which at least one output arc has been deleted.

Step 2 Delete all transitions that have all output places already deleted. Repeat Steps 1 and 2 until neither is applicable anymore. What is left over is the reduced net. Each step eliminates some nodes and arcs that are not part of the reduced net, until no more nodes and arcs can be deleted.

A Petri net is said to be state-machine allocatable iff every state machine allocation gives a reduced net that is either a strongly-connected state machine or a set of strongly-connected state machines. We will abbreviate the last remark by using the contraction "SSM" to denote "a strongly-connected state machine or a set of strongly-connected state machines". In Figure 3.5.1(a) we give an example of a Petri net which is state-machine allocatable. In Figure 3.1.5(b), we show two allocation

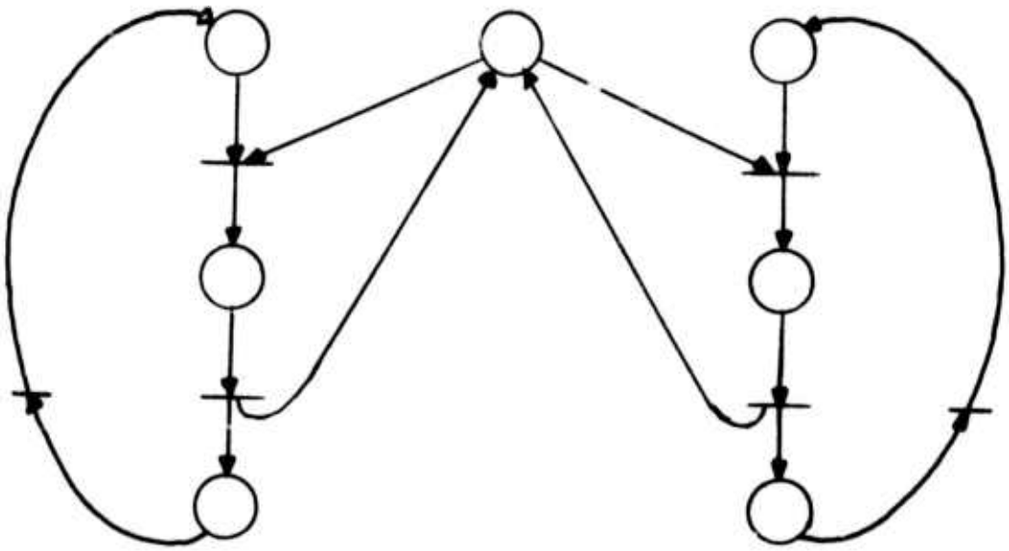
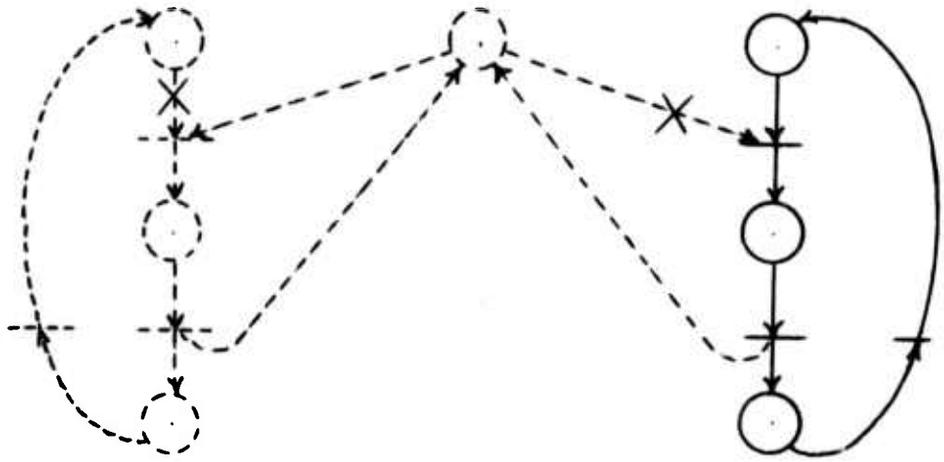
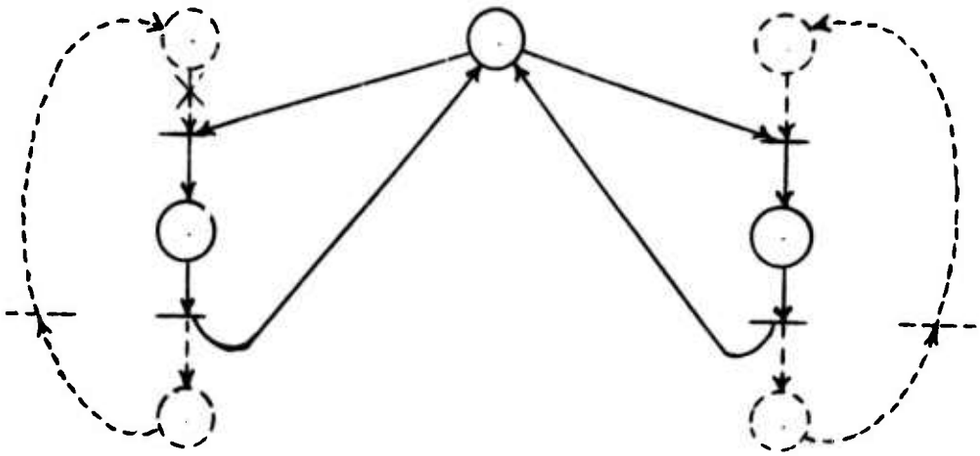


Figure 3.5.2(a)

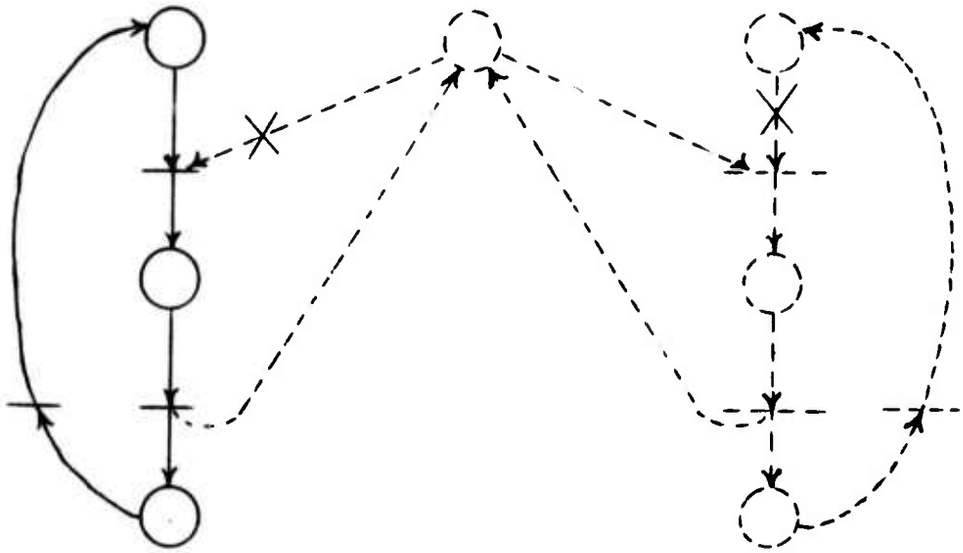


SMA Reduced net No. 1

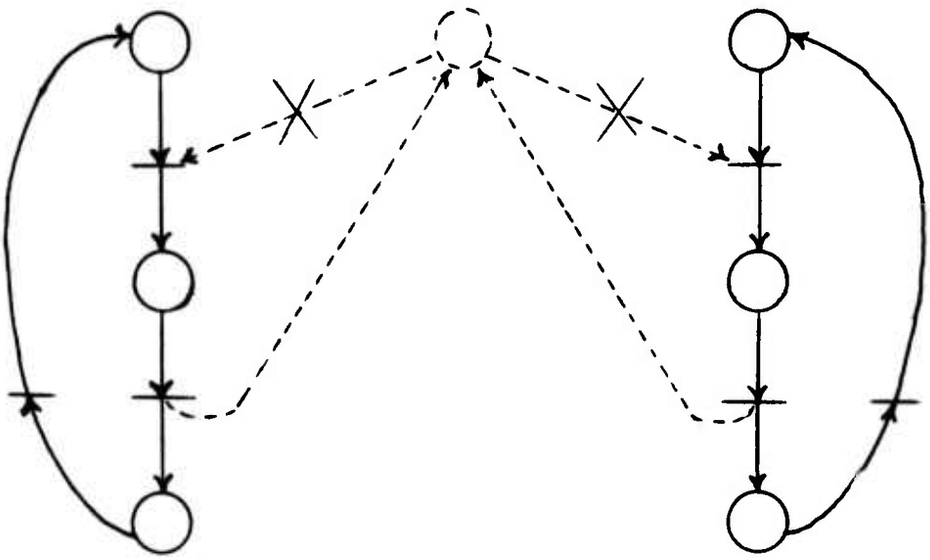


SMA Reduced net No. 2

Figure 3.5.2(b)

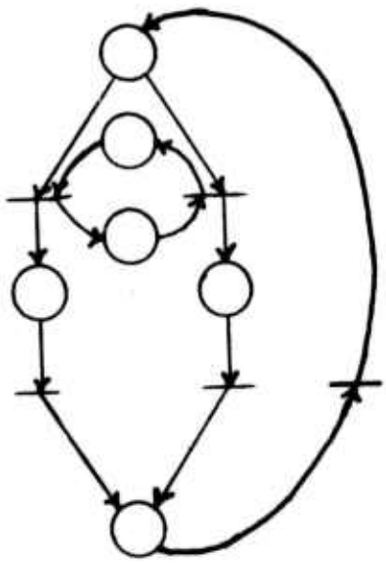


SMA Reduced net No. 3



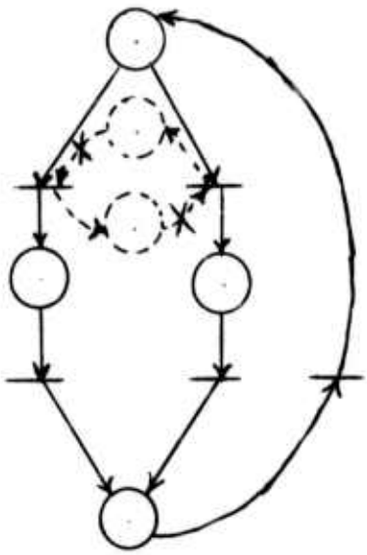
SMA Reduced net No. 4

Figure 3.5.2(b)

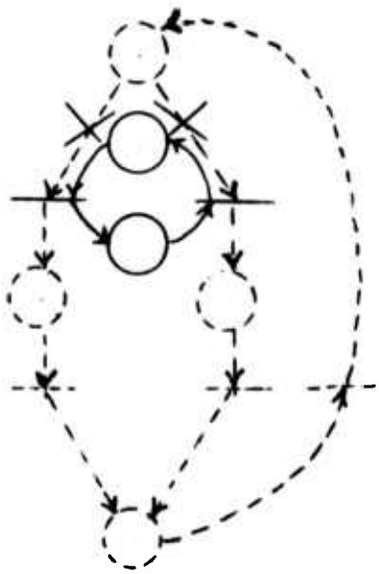


An SMD Petri net that is not SMA

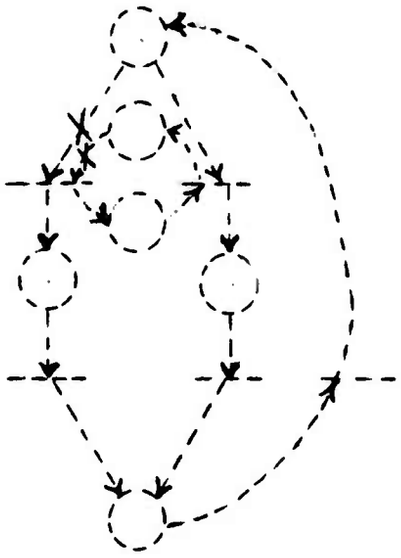
Figure 3.5.3(a)



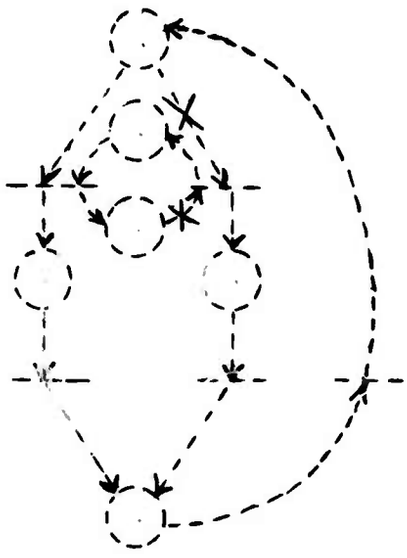
SMA Reduction No 1



SMA Reduction No. 2



SMA Reduction No. 3



SMA Reduction No. 4

Figure 3.5.3(b)

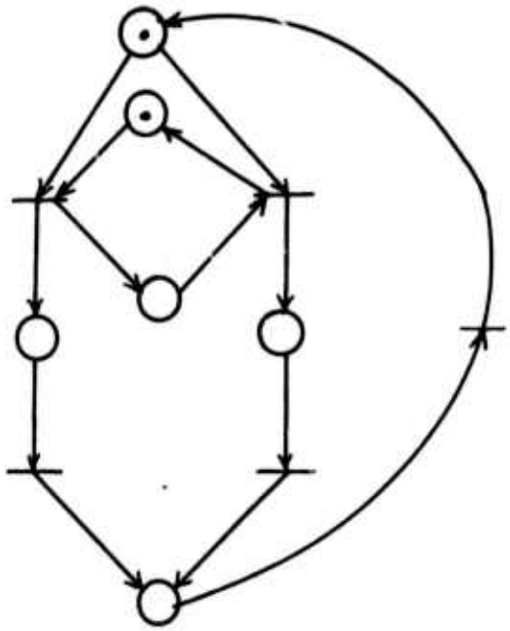


Figure 3.5.3(c) Non-SMA Petri net with a live marking

reduced SSM's. Reduced net No. 1 results from allocating t_4 to p_2 and reduced net No. 2 results from allocating t_4 to p_5. Since each of the two allocation reduced nets is an SSM, the net is SMA.

Let us give another example. Consider the Petri net in Figure 3.5.2. Since each of the state machine allocation reductions is an SSM, the net is SMA.

We now give an example of a Petri net which is SMD but not SMA. Such a net is given in Figure 3.5.3. We see that two of the SMA reductions are SSM's and the other two are null; the net is not SMA. In case the reader is a little confused, we would like to point out that the property of a Petri net being SMA is a structural property of the net, and he will note that no mention has been made of markings so far.

The reason we have introduced SMA Petri nets is that they form the largest subclass of SMD Petri nets we know of that has the property that any marking which puts at least one token on each state-machine component is live. This is the result presented in the next Theorem. We present this Theorem without proof and readers interested in the details are referred to Hack[H1,H5].

Theorem 3.5.2 (The SMA Petri net liveness Theorem) A marking M for an SMA Petri net $\mathcal{P}$ is live iff it puts at least one token on each state-machine component of the net.

This Theorem is of interest because it tells us how to construct a live marking for any SMA Petri net- simply put one or more tokens on each state machine component of the net. The non-SMA Petri net shown in Figure 3.5.3(a) also has the property that any marking that puts at least one token on every state machine is live, as the reader can easily verify.

reduced SSM's. Reduced net No. 1 results from allocating t_4 to p_2 and reduced net No. 2 results from allocating t_4 to p_5. Since each of the two allocation reduced nets is an SSM, the net is SMA.

Let us give another example. Consider the Petri net in Figure 3.5.2. Since each of the state machine allocation reductions is an SSM, the net is SMA.

We now give an example of a Petri net which is SMD but not SMA. Such a net is given in Figure 3.5.3. We see that two of the SMA reductions are SSM's and the other two are null; the net is not SMA. In case the reader is a little confused, we would like to point out that the property of a Petri net being SMA is a structural property of the net, and he will note that no mention has been made of markings so far.

The reason we have introduced SMA Petri nets is that they form the largest subclass of SMD Petri nets we know of that has the property that any marking which puts at least one token on each state-machine component is live. This is the result presented in the next Theorem. We present this Theorem without proof and readers interested in the details are referred to Hack[H1,H5].

Theorem 3.5.2 (The SMA Petri net liveness Theorem) A marking M for an SMA Petri net $\mathcal{P}$ is live iff it puts at least one token on each state-machine component of the net.

This Theorem is of interest because it tells us how to construct a live marking for any SMA Petri net- simply put one or more tokens on each state machine component of the net. The non-SMA Petri net shown in Figure 3.5.3(a) also has the property that any marking that puts at least one token on every state machine is live, as the reader can easily verify.

Figure 3.5.3(c) shows the Petri net of Figure 3.5.3(c) with a live marking. We present the following as an open problem:

What is the largest class of SMD Petri nets which has the property that any marking that puts at least one token on every state machine component is live?

We now turn to examining the issue of consistency for SMD Petri nets.

3.5.3 Consistency of SMD Petri nets which have a live marking

Recall Corollary 3.2.1, which we repeat here for convenience:

Corollary 3.2.1 If a Petri net $\mathcal{P}$ is LB, then there exists an initial marking M for which there exists a cyclic firing sequence.

In Section 3.4, we established the connection between a cyclic firing sequence and the steady-state equivalent net for event graphs and LSP Petri nets. We now introduce a concept similar to the steady-state equivalent net in connection with live SMD Petri nets. A firing sequence, as we have pointed out in Section 3.4, expresses an ordering relation on transition firings in a fashion which obscures the concurrency that is inherent in the Petri net. To preserve this inherent concurrency, we introduced the behavior graph for LSP Petri nets. We will now introduce a more general concept to study the behavior of SMD Petri nets- the occurrence graph. This notion is not a new one, having been studied extensively by Holt[H6]. As we did for behavior graphs, we shall illustrate the construction of an occurrence for an LS SMD Petri net by means of an example. We begin with a live, safe marking for an SMD Petri net(see Figure 3.5.4).

Construction of Occurrence Graph Begin by drawing every marked place

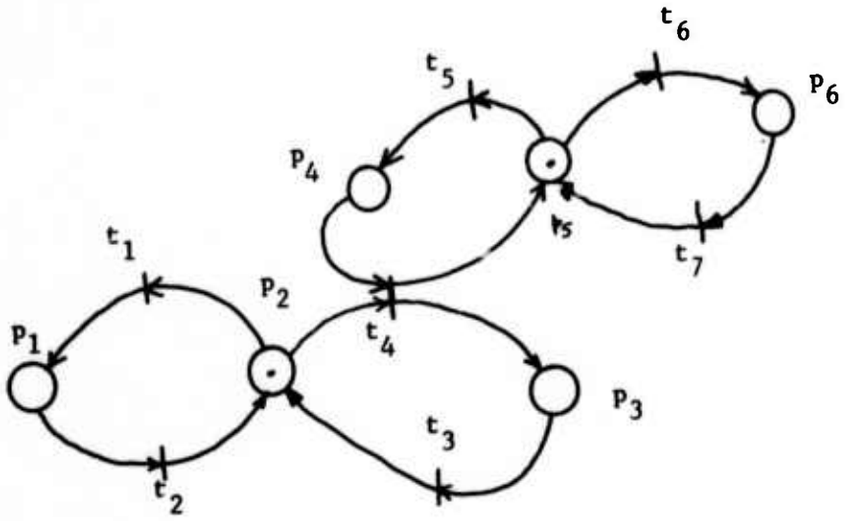
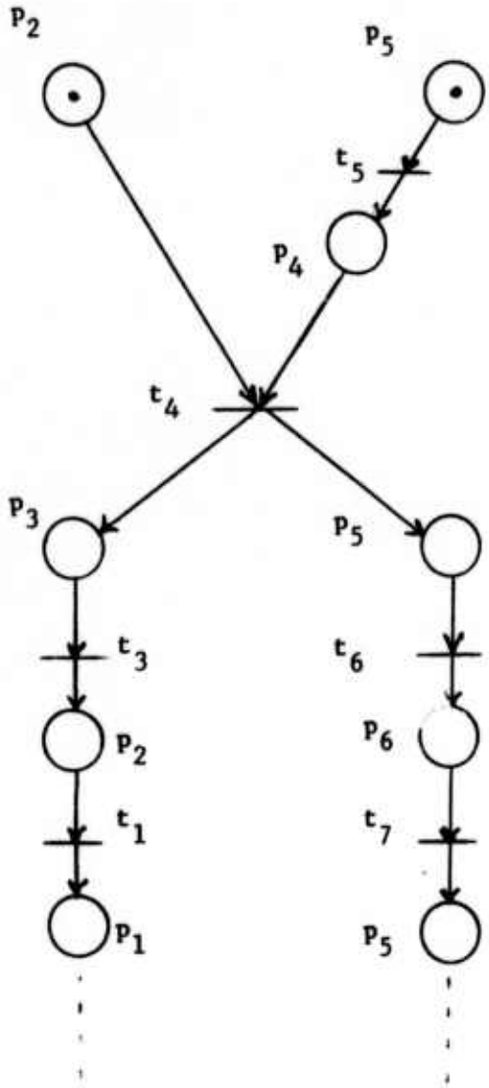
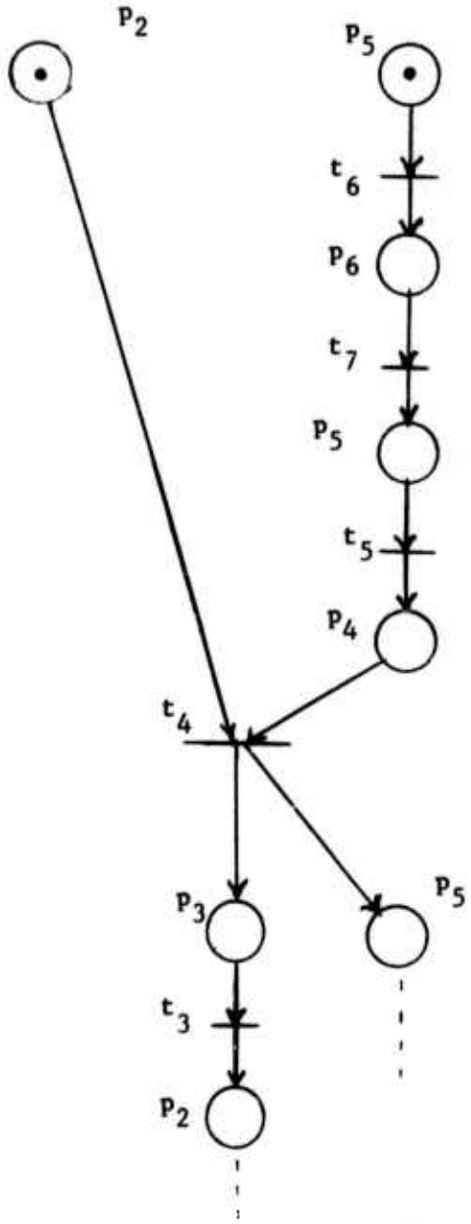


Figure 3.5.4



Occurrence Graph No. 1

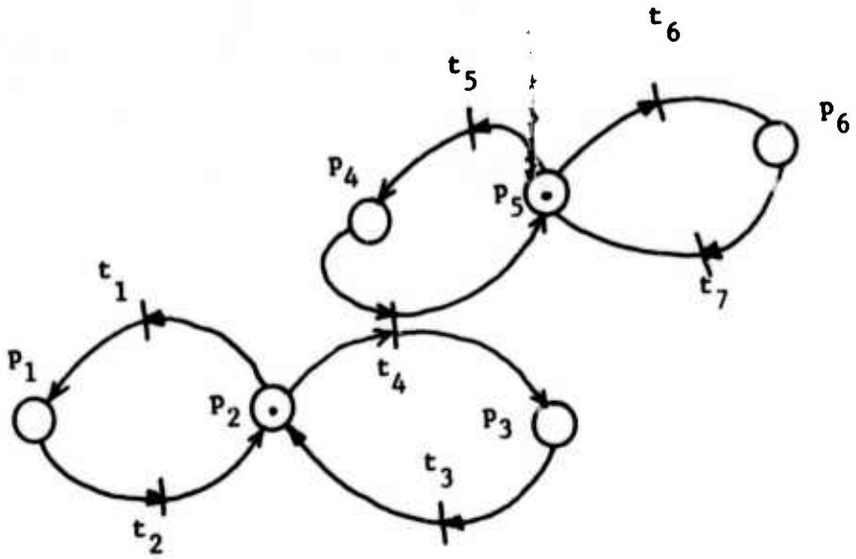


Occurrence Graph No. 2

Figure 3.5.5

in the net together with a token on each place (i.e., places p_2 and p_5). Now, since a place may have more than one output transition, we may have several enabled transitions connected to the same place. In event graphs and LSP Petri nets, we saw that every place has <u>at most one</u> transition enabled at any marking. This is not necessarily true for SMD Petri nets. As an example, consider the marked Petri net of Figure 3.5.4. Both the output transitions t_5 and t_6 of place p_5 are seen to be enabled; since we can fire only one of these transitions, we must make a choice between them and fire the one we choose. We will <u>extend a place</u> in an occurrence graph by drawing the output transition chosen for firing, firing that transition and then drawing the new marked place(s) that result. A <u>slice</u> of an occurrence graph is a set of places that forms a cut-set of the graph. We can talk about <u>extending a slice</u> in the same way as we did for behavior graphs, the difference being that a choice may have to be made between enabled transitions. In the construction of a behavior graph, there never occurs a slice for which a choice has to be made between output transitions. Thus, there is only one behavior graph for an event graph or an LSP Petri net, and this graph is unique. On the other hand, several occurrence graphs may be possible for LS SMD Petri nets. In Figure 3.5.5, we show two possible occurrence graphs for the net in Figure 3.5.4. The reader will realize that an infinite number of occurrence graphs is possible for this net, or, for that matter, in any LB net with a non-persistent marking.

Now let us apply Corollary 3.2.1 to the occurrence graph for an LS SMD Petri net. An occurrence graph is a concurrent representation of a firing sequence for a Petri net, and each slice represents a marking of the net. A repeated slice thus represents the repeated occurrence of



Consistent current assignment

$\varphi_1 = 1$ $\varphi_2 = 1$ $\varphi_3 = 2$ $\varphi_4 = 2$ $\varphi_5 = 2$ $\varphi_6 = 3$

$\varphi_7 = 3$

Figure 3.5.6

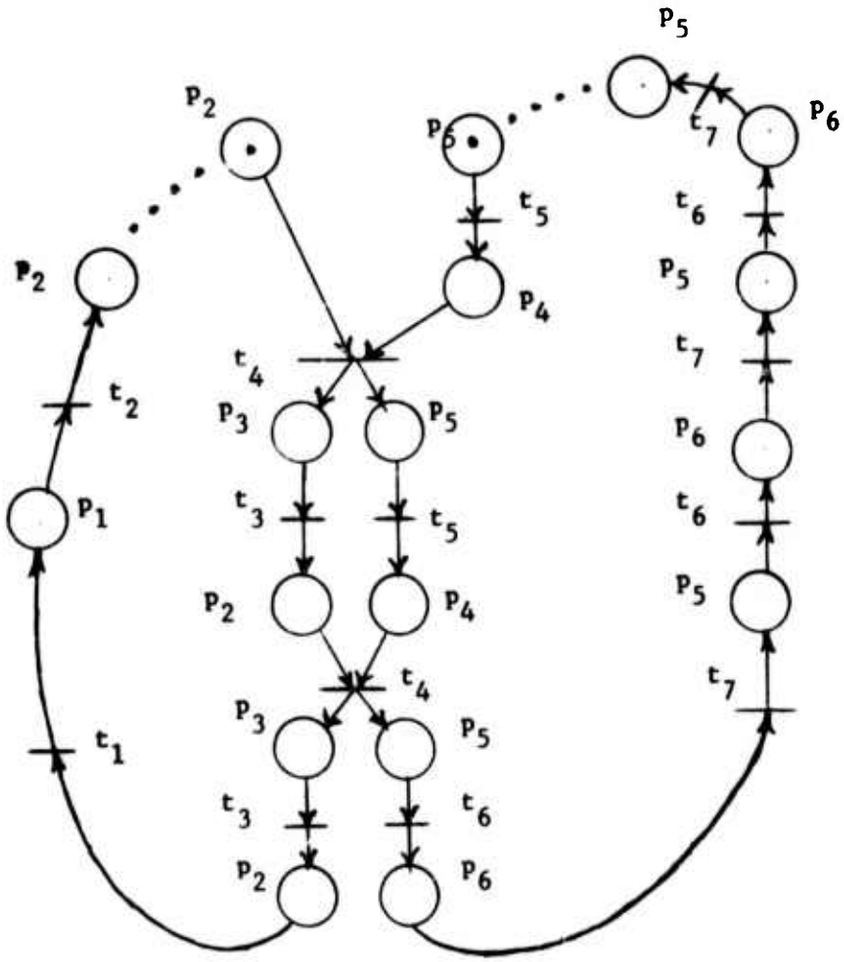


Figure 3.5.7 Cyclic Frustrum

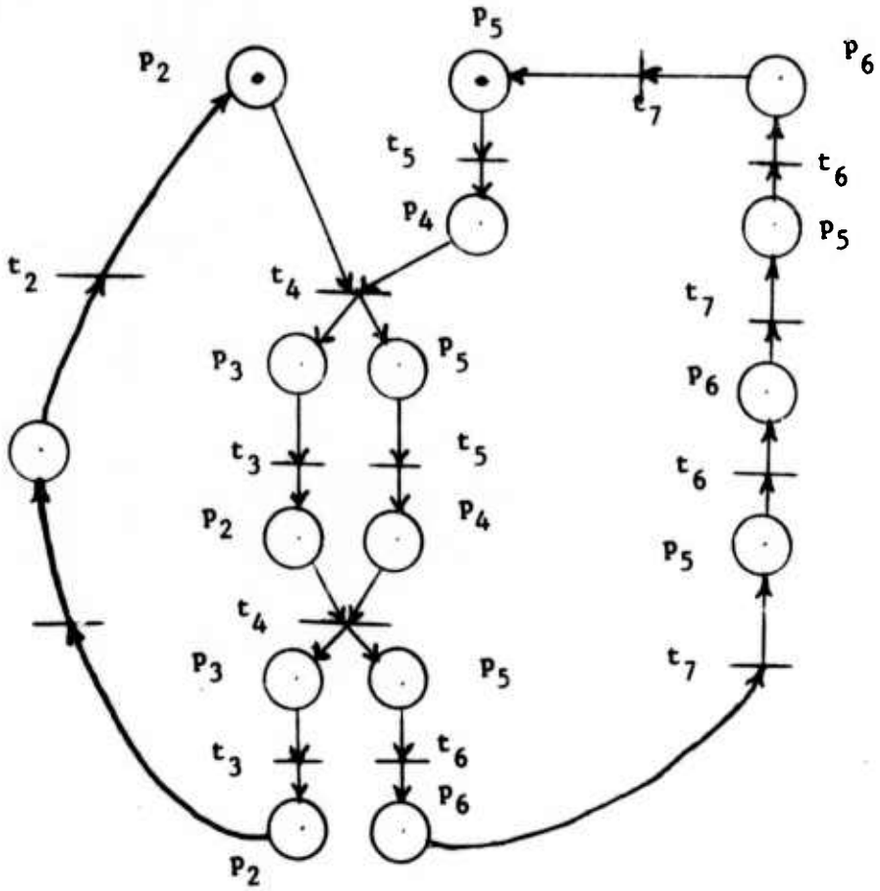
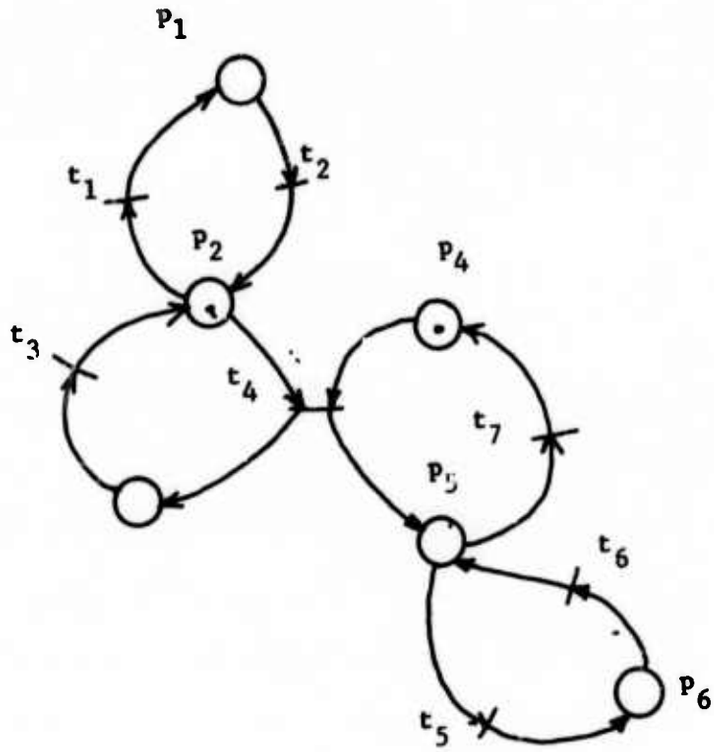


Figure 3.5.8 c-equivalent net

some marking. The portion of an occurrence graph between two consecutive occurrences af a slice is termed a cyclic frustrum, and corresponds to a cyclic firing sequence for the marked net. Theorem 3.2.1 tells us that for any consistent current assignment for an LB Petri net, we can find a cyclic frustrum in the occurrence graph of the net. The number of occurrences of any transition in the cyclic frustrum equals its current in a consistent current assignment.

Example 3.5.1 Consider the SMD Petri net of Figure 3.5.4. In Figure 3.5.6, we show this Petri net with a minimal integer consistent current assignment. We draw the occurrence graph as discussed earlier, and Figure 3.5.7 shows a cyclic frustrum of this occurrence graph, in which the multiplicity of each transition equals its associated current in the consistent current assignment exhibited in Figure 3.5.6. We now coalesce corresponding places in the repeated slice in a manner similar to what we did for behavior graphs. The resulting strongly-connected net is termed a consistency -equivalent net for the SMD Petri net, abbreviated to "c-equivalent net" . The c-equivalent net for the cyclic frustrum of Figure 3.5.7 is shown in Figure 3.5.8.

Let us now note some facts about the relationship between an SMD Petri net and its c-equivalent Petri net. Let us begin by saying that we will consider only minimal integer consistent current assignments. Note that the c-equivalent net of an SMD Petri net $\mathcal{P}$ is not unique. Figure 3.5.9(a) shows an SMD Petri net $\mathcal{P}$ with a minimal integer consistent current assignment. The net $\mathcal{P}$ has several c-equivalent nets, two of which are shown. We observe that in the construction of a c-equivalent net for an



Minimal integer consistent current assignment

$\varphi_1 = \varphi_2 = 1$

$\varphi_3 = \varphi_4 = \varphi_7 = 2$

$\varphi_5 = \varphi_6 = 2$

Figure 3.5.9(a) SMD Petri net $\mathcal{P}$.

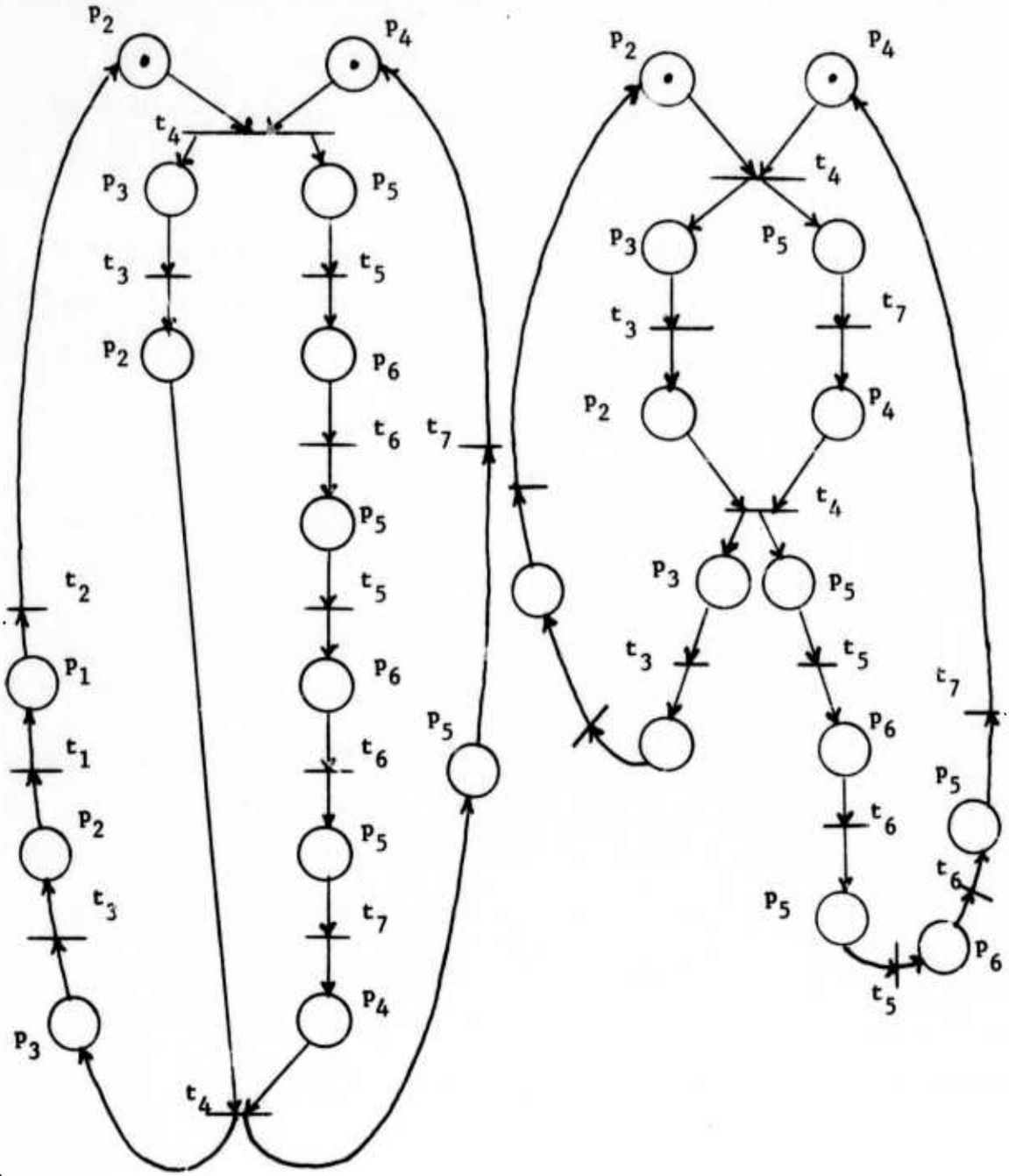

p2
p4
t4
p3
p5
t3
t5
p2
p6
t6
t7
p5
t2
t5
p1
p6
t1
t6
p5
p2
t3
t7
p4
p3
t4
p2
p4
t4
p3
p5
t3
t7
p2
p4
t4
p3
p5
t3
t5
t7
p6
p5
t6
t6
p5
p6
t5

Figure 3.5.9 (b)

SMD Petri net $\mathcal{P}$ with a minimal integer consistent current assignment, each state-machine in $\mathcal{P}$ corresponds to a circuit in the c-equivalent net. We formalize this in the following Lemma.

Lemma 3.5.1 Let $\mathcal{P}$ be an LS SMD Petri net with a minimal integer consistent current assignment Φ, and let $\mathcal{C}$ be a c-equivalent net for $\mathcal{P}$. Then, every state machine component in $\mathcal{P}$ corresponds to a simple circuit in $\mathcal{C}$.

Proof Let $\mathcal{F}$ be the cyclic frustrum corresponding to $\mathcal{S}$. Applying Theorem 3.5.1, we see that the initial slice of $\mathcal{F}$ must contain at least one place from each state-machine in $\mathcal{P}$. Now consider a chain in $\mathcal{F}$ bounded at its extremeties by two consecutive instances of some place p_i. In $\mathcal{S}$, this chain corresponds to a simple circuit. By the construction procedure for occurrence graphs, it is obvious that each state machine in corresponds to a chain in $\mathcal{F}$. This proves the Lemma.

Example Consider the LS SMD Petri net of Figure 3.5.6. Each of the two state machines corresponds to a circuit in the c-equivalent net shown in Figure 3.5.8. Note that there are two circuits in the c-equivalent net which do not correspond to any state-machine in the SMD Petri net. An example of such a circuit is $p_2t_4p_5t_5p_4t_4p_3t_3p_2t_2p_1t_1p_2$.

Since the c-equivalent net corresponds to a cyclic firing sequence for an LB SMD Petri net, we see by applying Theorem 3.2.1 that the multiplicity of a transition in the c-equivalent net must equal its current in a minimal integer consistent current assignment.

To recapitulate the main results of this Section, we have the following:
(1) We have introduced the subclass of SMD Petri nets known as SMA Petri

nets, which have the property that a marking is live if and only if each state machine contains at least one token. This is the largest class of SMD Petri nets known to date that has this property.

(2) We have introduced the c-equivalent net of a Petri net which has a live, bounded marking. For SMD Petri nets, we have seen that every state-machine corresponds to a circuit in a c-equivalent net. The converse is not true in general.

We now turn to the issue of applying Petri net theory to the analysis of asynchronous concurrent systems.

CHAPTER 4

TIMED PETRI NETS

4.1 Timing in Petri Nets

In our discussions so far, we have not entered into any timing considerations in connection with Petri nets. Thus, while SMD Petri nets can model the structure of asynchronous concurrent systems, they do not contain enough information to be used for a study of issues of performance of the type discussed in Chapter 1. In defining Petri nets, we made no assumptions about the length of time it takes for a transition to fire. In real-world systems, activities do not take place instantaneously. Every activity in a system has a time duration which is different from zero, and in all the systems we will model, we will make the added assumption that all activities complete in a finite amount of time. In the Petri nets that we use to model these systems, we will assume that every transition takes a bounded, non-zero amount of time to fire. The resulting model of asynchronous, concurrent systems is termed Timed Petri nets, and is formalized below.

Definition 4.1.1 A Timed Petri Net is a pair $\langle \mathcal{P}, \Omega \rangle$ where $\mathcal{P}$ is a Petri net $\langle P, T, A \rangle$ and Ω is a function that assigns a real, non-negative number τ_i to each transition t_i in the net.

$\Omega : T \longrightarrow \mathbb{R}$ { $\mathbb{R}$ is the set of non-negative real numbers }.

This non-negative real number $\tau_i = \Omega(t_i)$ is termed the firing time of transition t_i.

The operation of the net can now be assumed to take place in real time. At any instant τ of real time, the net has a marking $M(\tau)$, with the understanding that we may view M both as a vector and a function. We denote the initial marking as $M(0)$. We write $M(\tau,p)$ for $M(\tau)(p)$, i.e., the number of tokens on place p at time τ. A transition t_i is said to be enabled at time τ if and only if every input place of transition t_i has at least one token on it, i.e.,

$$M(\tau,p_k) \geq 1 \quad \text{for all } p_k \in {}^{\cdot}t_i.$$

When transition t_i is enabled, a firing can be initiated. When a firing is initiated, a token is removed from each input place of t_i and transition t_i is said to be executing. This execution phase lasts for τ_i seconds, where τ_i is the firing time of transition t_i. At the end of this time duration, the firing of transition t_i terminates, and a token is placed on each output place $p_j \in t_i^{\cdot}$. This completes the firing of transition t_i.

The three phases of a transition firing can be visualized by imagining every transition as consisting of two transitions and an intermediate place as shown in Figures 4.1.1. The firing time of the transition t_i can now be associated with the place π_i in the following fashion: When transition t_i initiates, t_i' fires instantaneously, a token is removed from each input place of t_i', and a token is deposited on place π_i. This token is held by place π_i for a duration equal to τ_i, the firing time of t_i. At the end of this interval, transition t_i'' fires, corresponding to the termination of t_i.

The initiations and terminations of transitions in a Petri net must

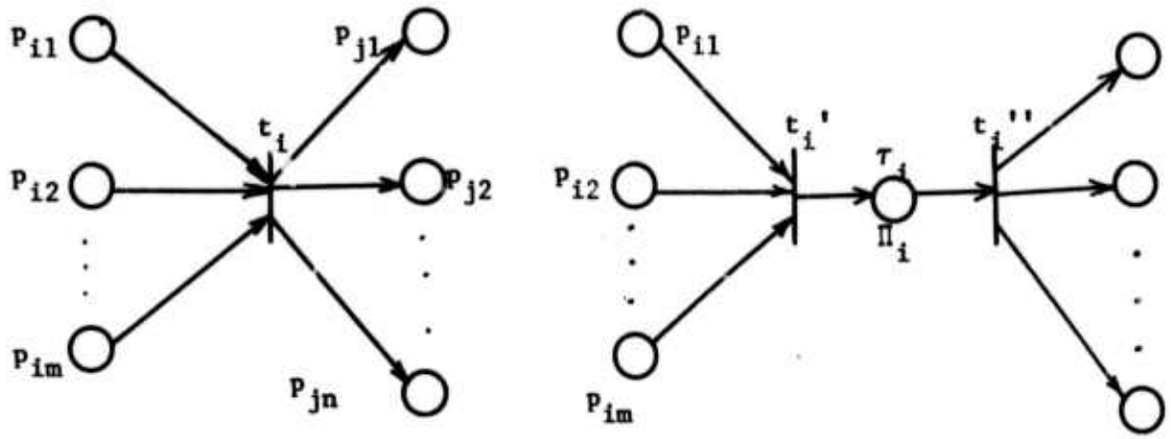


Figure 4.1.1

satisfy the following balance equation:

Notation

Let p be a place in a Petri net $\mathcal{P}$. Let $t_{i1},\ldots t_{in}$ be the input transitions and $t_{j1},\ldots t_{jm}$ the output transitions of p.

Let $I(\tau,t)$ denote the number of initiations of transition t up to and including time τ,

and $T(\tau,t)$ denote the number of terminations of transition t up to and including time τ.

Then, if $M(0,p)$ is the number of tokens on place p at time zero, and $M(\tau,p)$ that at time τ, the following must be satisfied:

$$M(0,p) + T(\tau,t_{i1})+\ldots+ T(\tau,t_{in}) = M(\tau,p) + I(\tau,t_{j1}) +\ldots+ I(\tau,t_{jm})$$

We will refer to this as the token balance equation, and will make use of it in later sections. A word now about the choice of a firing time for a transition in a Petri net.

We have defined τ_i to be a non-negative real number, thereby assuming that the duration of each activity is fixed. This may not be a very accurate picture of real-world systems, for , in practice, the execution time of an operator depends upon the data it is called upon to handle. In a floating -point adder, for example, the add time will depend upon the arguments and their exponents. Thus, it may be more reasonable to assume that a transition firing time is a random variable whose distribution can be represented by a rectangular distribution of the form shown in Figure 4.1.2.

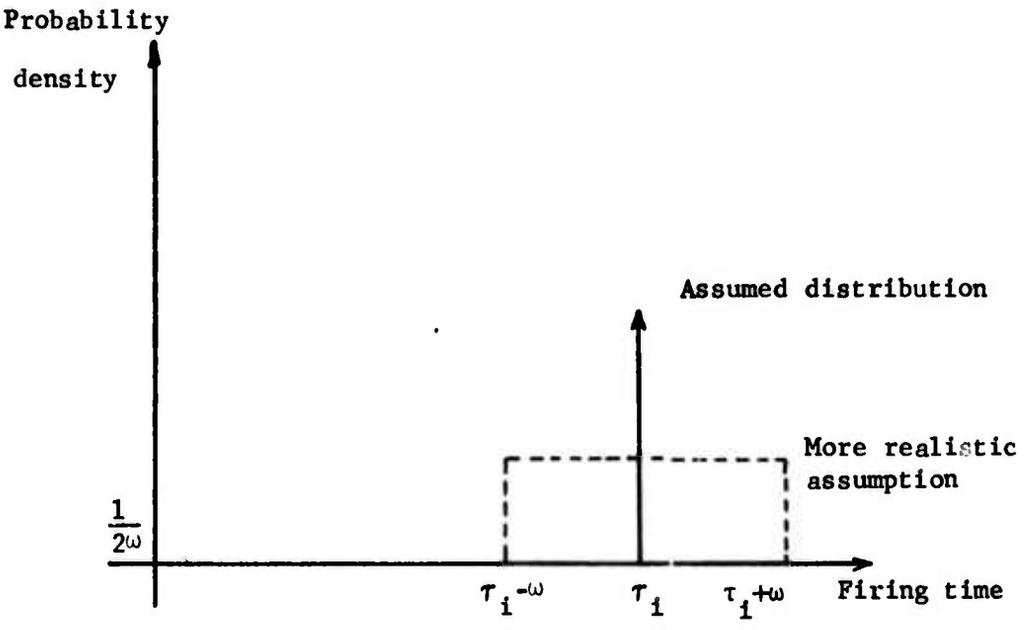


Figure 4.1.2

We will return to a consideration of statistical firing times in Section 4.5.

4.2 Dynamic Behavior of Timed Petri Nets

In Chapters 2 and 3, we introduced the ideas of firing sequence and behavior graph to characterize the action of Petri nets. For timed Petri nets, we have postulated the existence of a real-time axis against which the firing of transitions can be noted. At time zero, the net has a marking M(0). Transitions are allowed to fire, and the time at which the firing of a transition takes place is recorded in a table known as a firing schedule.

A firing schedule for a timed Petri net is a set of sequences of initiation and termination times for the transitions of a net. The firing of a transition is feasible if the transition was enabled at the instant the firing was initiated. If every firing in a firing schedule is feasible, the firing schedule is feasible. A firing schedule is infeasible (or not feasible) if it calls for the initiation or termination of an activity earlier than allowed by the termination of other activities. Figure 4.2.1 is an example of a timed Petri net and Figure 4.2.2 gives a feasible firing schedule for it.

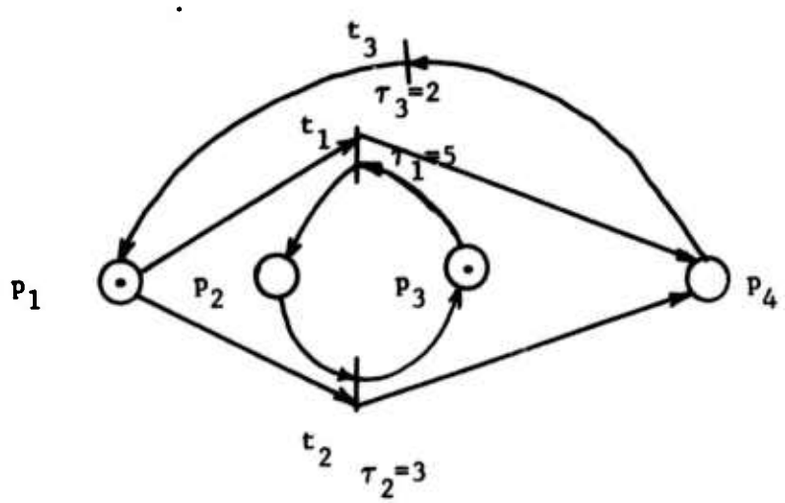


Figure 4.2.1 A Timed Petri net

Number of transition firings / Transitions	1	2	3	4	5
t_1	(0, 5)	(12, 17)	(24, 29)	---------	---------
t_2	(7, 10)	(19, 22)	(31, 34)	---------	---------
t_3	(5, 7)	(10, 12)	(22, 24)	(29, 31)	---------

Figure 4.2.2 Feasible Firing Schedule for Timed Petri Net in Figure 4.2.1.

Number of firings / Transitions	1	2	3	4	5
t_1	(0, 5)	(12, 17)	(24, 29)	---------	---------
t_2	(7, 10)	(19, 22)	(30, 33)	---------	---------
t_3	(5, 7)	(10, 12)	(17, 19)	(22, 24)	(29, 31)

Figure 4.2.3 Infeasible Firing Schedule for Timed Petri Net in Figure 4.2.1.

Figure 4.2.3 gives an infeasible firing schedule for the same net. The starred entry implies that the third initiation of transition t_2 takes place at $\tau = 30$; this is impossible, because the third initiation of t_2 cannot take place before the termination of the fifth firing of t_3, which happens at $\tau = 31$ units.

Readers will note that in the feasible firing schedule of Figure 4.2.2, the initiations of transitions t_1, t_2 and t_3 take place at regular intervals. For example, consecutive initiations of transitions t_1 and t_2 occur at intervals of 12 time units. Transition t_3 behaves in a slightly different fashion. We notice that the first, third, fifth,... transition initiations occur every 12 time units, i.e., alternate initiations occur every 12 time units.

A firing schedule with this property is termed a periodic firing schedule. If all transitions in the net can have consecutive initiations at regular intervals, we would term this a strongly-periodic firing schedule. The computation rate of a transition is the average number of firings of that transition in unit time. We can see that transitions t_1 and t_2 have a computation rate of once every 12 time units, or 1/12. Transition t_3 has a computation rate of twice every 12 time units, or 1/6. These computation rates are the maximum rates possible for the transitions.

The Petri net in Figure 4.2.1 is seen to be an LSP net, so that it represents a deterministic system, and its steady-state equivalent net is the multiply-labelled event-graph shown in Figure 4.2.4. The multiplicity of transition t_3 is 2, while that of both t_1 and t_2 is 1. A consistent current assignment for the net $\varphi_1 = \varphi_2 = 1$, $\varphi_3 = 2$.

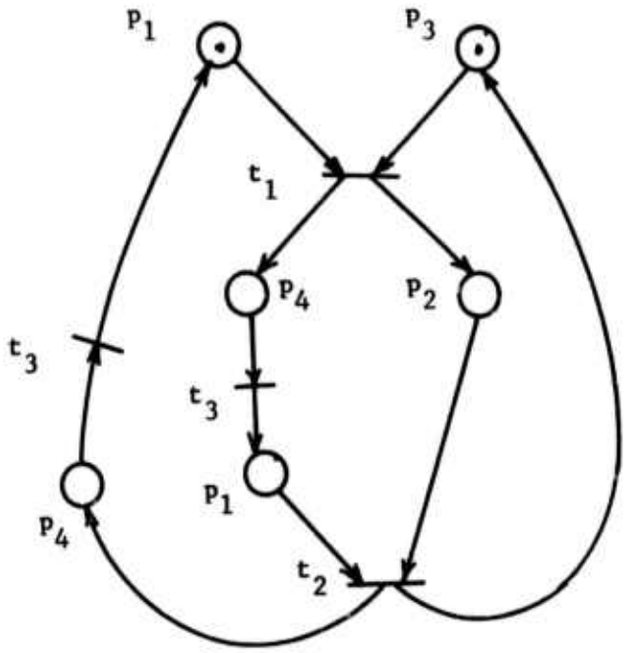


Figure 4.2.4 Steady-state equivalent net for the Petri net in Figure 4.2.1

We note that the computation rates of transitions t_1, t_2 and t_3 are in the ratio of their currents in a consistent current assignment. Before we can justify this, we look informally at the computation rate of transitions in an event graph; successive generalizations of the ideas presented below will lead us to all results of interest in this thesis, including the computation rate of transitions in timed SMD Petri nets.

Consider the simple circuit shown in Figure 4.2.5, in which each transition t_i has an associated firing time τ_i. We will begin by assuming that the circuit has one token on it, and will then extend the analysis to n tokens. With one token on the circuit, each transition can fire

in turn over and over again. Suppose that the circuit is denoted by $p_1t_1p_2t_2\cdots p_jt_j$, and suppose (without loss of generality) that initially the token is on place p_i. Let $\pi = \tau_1 + \tau_2 + \ldots + \tau_j$. Then, the token fires every transition in the circuit in turn and reappears on p_i every π seconds, assuming that no time is allowed to elapse between a transition being enabled and being initiated. Under this assumption, every transition initiates at intervals of π seconds, and π is the period of the firing schedule for the circuit that realizes the maximum computation rate. The computation rate is easily seen to be $\rho = 1/\pi$. Now suppose that the circuit has n tokens on it instead of 1. For the combined action of the n tokens, the firing rate becomes n/π, or $\frac{n}{\sum_i \tau_i}$. Every transition in the circuit has a maximum computation rate given by this expression.

Let us now consider transitions in a timed event graph. We know from Theorem 3.2.1 that in an event graph with a live marking, the token content of every simple circuit is non-zero. Thus, let the token content of a circuit C_k be n_k, where $n_k \neq 0$. Also, let π_k denote the sum of the firing times of transitions in C_k. If every circuit were by itself, the transitions in circuit C_k would have a computation rate of n_k / π_k. However, in a strongly-connected event graph, the circuits are interconnected and intuitively it is clear that they will affect each others <u>natural computation rate</u> (i.e., the computation rate the transitions in a circuit would have if the circuit were isolated from the other circuits).

Now, without loss of generality, let C_1 be the circuit with the smallest natural computation rate n_n/π_1. Clearly, all transitions $t_{11},\ldots t_{1i}$

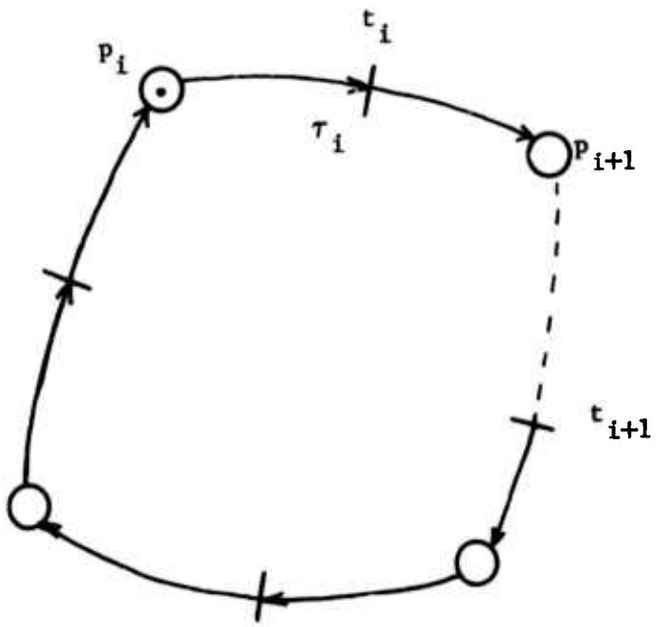


Figure 4.2.5.

$\in C_1$ have a computation rate that cannot exceed n_1/π_1 (see Figure 4.2.6).

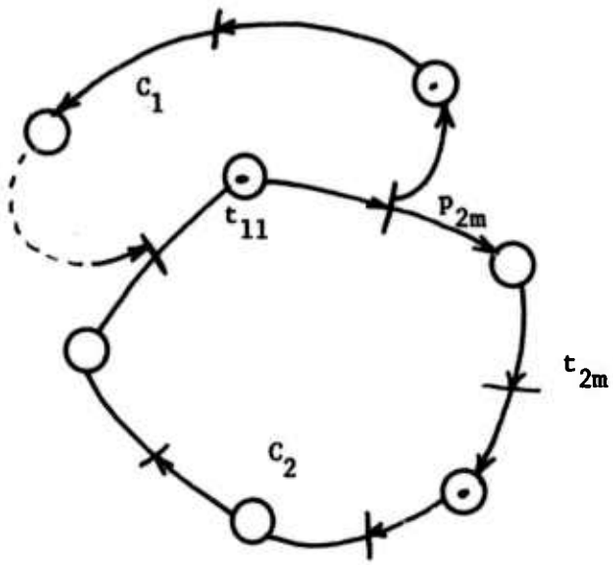


Figure 4.2.6

Extending this argument, we see that if there exists a directed path between any two transitions in the event graph, they must have the same computation rate. Since every transition is on some directed path from transitions t_{11}, t_{12}, ...t_{1i}, we conclude that all transitions in the event graph must have the same computation rate, which cannot exceed n_1/π_1. What this means is that in any timed event graph $\langle G,\Omega\rangle$, all transitions have the same computation rate, which cannot exceed

$$\min \left\{ n_1/\pi_1, \ldots n_k/\pi_k \right\}$$

where $C_1, \ldots C_k$ are the simple circuits of the graph,

n_i is the token content of circuit C_i, and

π_i is the sum of the firing times of transitions in circuit C_i.

We have been able to obtain, informally, a bound on the computation rate of transitions in a timed event graph. In the next Section, we will show that this bound can actually be realized.

4.3 Timed Event Graphs

In the previous section, we saw informally how we could obtain a bound on the steady state computation rate of a timed event graph. Here, we will show that the bound we have obtained is actually realizable; furthermore, we will show that there exists a strongly periodic firing schedule for which the transitions have a computation rate equal to the bound.

We will use the notation for event graphs given in Definition 3.3.2. To recapitulate, let $\mathcal{G}$ be a timed event graph $\langle G,\Omega \rangle$ where

G is a strongly-connected event graph $\langle V,A \rangle$.

Recall that $V = \{v_1, v_2, \ldots v_m\}$ is the set of transitions.

$A = \{a_{ij}\} \subseteq V \times V$ is the set of arcs.

a_{ij} is an arc that connects transition v_i to transition v_j.

Ω is a function that assigns to each transition v_i a firing time τ_i, i.e., $\Omega: V \rightarrow \mathbb{R}$ (where $\mathbb{R}$ is the set of non-negative real numbers). We write τ_i for $\Omega(v_i)$.

Let M(0) be the initial marking of the net. Recall that M is used interchangeably as a vector and a function. For event graphs,

$M(\tau, a_{ij})$ will denote the number of tokens on arc a_{ij} at time τ.

Arcs in event graphs will be treated in the same way as places in unrestricted Petri nets.

We now proceed to formalize the definition of a feasible firing schedule for timed event graphs.

Definition: A firing schedule S for a timed event graph $\mathcal{G}$ is a pair $\langle S_I, S_T \rangle$ such that

$$S_I : \mathbb{N} \times V \longrightarrow \mathbb{R}$$

$$S_T : \mathbb{N} \times V \longrightarrow \mathbb{R}, \qquad \mathbb{N} = \{1,2\ldots\}$$

and $S_T(n,v_i) = S_I(n,v_i) + \tau_i$.

$S_I(n,v_i)$ is the nth initiation time of transition v_i.

$S_T(n,v_i)$ is the nth termination time of transition v_i.

A firing schedule for a timed Petri net may not be realizable because some transitions are specified to fire when they are not enabled. A feasible firing schedule is one that does not violate this requirement. We now show how to determine the marking $M(\tau)$ at any instant of time τ for a timed event graph $\mathcal{G}$ which has a feasible firing schedule S.

Notation: At any instant of time τ in a firing schedule S for timed event graph $\mathcal{G}$,

$I_s(\tau,v_i)$ denotes the number of initiations of transition v_i up to and including time τ.

$T_s(\tau,v_i)$ denotes the number of terminations of transition v_i up to and including time τ.

Let $a_{ij} = (v_i,v_j)$ be an arc in the event graph G.

By token balance

$$M(\tau,a_{ij}) = M(0,a_{ij}) + T_s(\tau,v_i) - I_s(\tau,v_j).$$

Now consider the enabled transition v_j shown in Figure 4.3.1. If an initiation of v_j takes place at the instant τ, then immediately prior

to instant τ, there must have been at least one token on every input arc of v_j. We will use the notation τ^- to denote an instant $(\tau-\epsilon)$ where $\epsilon \longrightarrow 0$. τ^- will, for all practical purposes, denote the instant τ itself.

Definition: A firing schedule S for a timed event graph $\mathcal{G}$ is feasible iff for any transition v_j and for k = 1,2,3..., we have

$$\forall\, a_{ij} \in {}^{\cdot}v_j,\ M(S_I[k,v_j]^-,\ a_{ij}) \geq 1$$

where $S_I[k,v_j]^-$ denotes the instant just prior to $S_I[k,v_j]$.

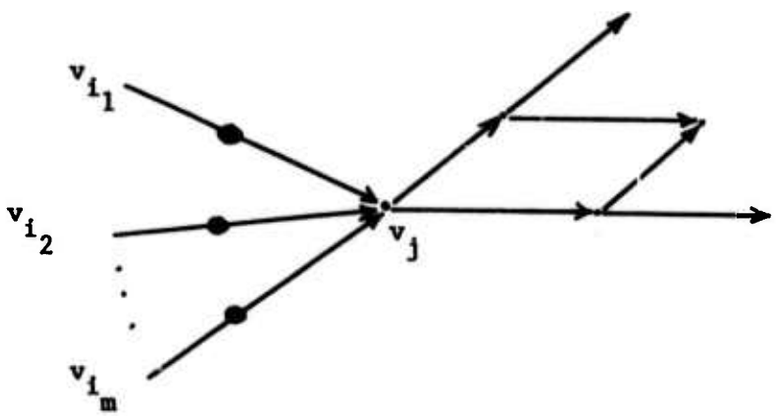


Figure 4.3.1 An instant when transition v_j can initiate.

Suppose we now consider a run of the event graph $\mathcal{G}$. Consider an arc a_{ij} that initially had $M(0,a_{ij})$ tokens on it. Suppose transition has fired n times up to and including an instant of time τ. It is

obvious that in the same interval of time, transition v_j could have fired at most $M(0,a_{ij}) + n - 1$ times without driving the token content of arc a_{ij} negative. The $(M(0,a_{ij}) + n)$th initiation of transition v_j must await the nth termination of transition v_i. We will show in the next Theorem that this is a necessary and sufficient condition for a firing schedule to be feasible.

Theorem 4.3.1: A firing schedule S for a timed event graph $\mathcal{G} = \langle G,\Omega \rangle$ is feasible iff for each arc a_{ij} in the graph G, and for n = 1,2,3...

$$S_T(n,v_i) \leq S_I(n + M(0,a_{ij}),\ v_j).$$

This can be stated equivalently as

$$S_I(n,v_i) + \tau_i \leq S_I\ (n + M(0,a_{ij}),\ v_j).$$

Proof:

Necessity: We prove this by contradiction. Suppose there exists a feasible firing schedule such that for some n and some a_{ij}

$$S_T(n,v_i) > S_I\ (n + M(0,a_{ij}),\ v_j), \quad \text{i.e.,}$$

$$S_I(n,v_i) + \tau_i > S_I(n + M(0,a_{ij}),\ v_j).$$

Let $\tau = S_I(n + M(0,a_{ij}),\ v_j)$ and therefore

$$I_S(\tau,v_j) = n + M(0,a_{ij}).$$

Since $\tau < S_T(n,v_i)$, we have $T_S(\tau,v_i) < n$.

Using token balance,

$$M(\tau,a_{ij}) = M(0,a_{ij}) + T_S(\tau,v_i) - I_S(\tau, v_j)$$

$$= M(0,a_{ij}) + T_S(\tau,v_i) - n - M(0,a_{ij})$$

$$= T_S(\tau,v_i) - n$$

i.e., $M(\tau,a_{ij}) < 0$

This is the desired contradiction of feasibility of S.

Sufficiency: Suppose S is a feasible schedule, and consider any instant of time $\tau > 0$ such that

$$S_I(n + M(0,a_{ij}), v_j) = \tau$$

We wish to show that for the instant τ^- just prior to an initiation of v_j

$$M(\tau^-,a_{ij}) \geq 1.$$

Now, at time τ^-,

$$M(\tau^-,a_{ij}) = M(0,a_{ij}) + T_S(\tau^-,v_i) - (n + M(0,a_{ij}) - 1)$$

$$= T_S(\tau^-,v_i) - n + 1$$

but

$$T_S(\tau^-,v_i) \geq n.$$

Hence,

$$M(\tau^-,a_{ij}) \geq 1$$

■

We have introduced the idea of a strongly periodic firing schedule in Section 4.2. We now give a formal definition of a strongly periodic firing schedule for a timed event graph.

Definition: A feasible firing schedule S for a timed event graph $\mathcal{G}$ is strongly periodic with period $\pi > 0$ iff there exist real numbers $x_i \geq 0$ such that

$$S_I(n,v_i) = x_i + (n-1)\pi \qquad n = 1,2,\ldots \quad i = 1,2,\ldots m.$$

The real numbers $x_1,\ldots x_m$ are the displacements of the firing times of transitions from the instants $0, \pi, 2\pi,\ldots$

Corollary 4.3.1.1: Let a_{ij} be an arc in a timed event graph $\mathcal{G}$. Then $\mathcal{G}$ has a strongly periodic firing schedule with period $\pi > 0$ and with displacements $x_1,\ldots x_m$ iff $\forall a_{ij} \in G$, we have

$$x_i + \tau_i \leq x_j + M(0,a_{ij}).$$

Proof: Theorem 4.3.1 states that $\forall a_{ij} \in G$ and for $n = 1,2,\ldots$

$$S_I(n,v_i) + \tau_i \leq S_I(n + M(0,a_{ij}), v_j) \ldots [4.1].$$

By the definition of a strongly periodic firing schedule we have

$$S_I(n,v_i) = x_i + (n-1)\pi$$

and $S_I(n + M(0,a_{ij}), v_j) = x_j + (n + M(0,a_{ij}) - 1).$

Substituting into inequality 4.1, we have

$$x_i + (n-1) + \tau_i \leq x_j + (n + M(0,a_{ij}) - 1)$$

or $$x_i + \tau_i \leq x_j + M(0,a_{ij})$$

■

The inequalities of Corollary 4.3.1.1 can be rewritten

$$x_j - x_i \geq \tau_i - M(0,a_{ij}) \quad \forall a_{ij} \in G.$$

The quantity $\tau_i - \pi M(0,a_{ij})$ is a constant for each arc a_{ij}.

Let us write

$$\tau_i - \pi M(0,a_{ij}) = c_{ij}.$$

Corollary 4.3.1.1 can be rewritten as follows:

A timed event graph $\mathcal{G}$ has a strongly periodic firing schedule for displacements $x_1,\ldots x_m$ iff

$$\forall\, a_{ij} \in G, \text{ we have } x_j - x_i \geq c_{ij} \ldots [4.2]$$

where $c_{ij} = \tau_i - \pi M(0,a_{ij})$.

We would now like to investigate under what conditions the set of inequalities [4.2] is true. In order to do this, we make a brief excursion into the theory of potentials for directed graphs. The development given here follows very closely the material in Berge and Ghouila-Houri, pages 144-145 and pages 155-157 [B3].

4.3.2 Existence of Potential Differences for event graphs

Suppose with each arc a_{ij} of an event graph G we associate a real number θ_{ij} which satisfies the condition that for every cycle $\xi \in G$,

$$\sum_{a_{ij} \in \xi^+} \theta_{ij} - \sum_{a_{ij} \in \xi^-} \theta_{ij} = 0.$$

where ξ^+ denote the arcs of the cycle oriented in a given sense, and ξ^- the arcs oriented in the opposite sense (see Figure 4.3.2). Then, we will say each θ_{ij} represents a potential difference across its corresponding arc a_{ij}. Readers familiar with circuit theory will notice the similarity

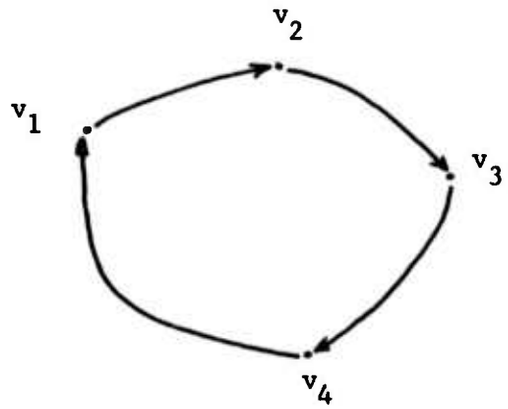


Figure 4.3.2 A Cycle

of this notion of potential difference to that used for electrical networks. In networks we can choose any arbitrary node to be a reference node and assign it a potential of zero. The potential of any node in the network is then equal to the potential difference between it and the reference node. We will now show that if the arcs of an event graph can be given a potential difference assignment, then it is equivalent to saying that each transition v_i can be assigned a potential x_i such that the potential difference $\theta_{ij} = x_j - x_i$. We have used x_i to denote the potential associated with transition v_i, and have used x_i earlier to denote the displacement associated with v_i. This has been done intentionally.

Theorem 4.3.2: A function θ: A $\longrightarrow \mathbb{R}$ which assigns a real number θ_{ij} to each arc a_{ij} is a potential difference assignment iff there exists a function X: V $\longrightarrow \mathbb{R}$ which associates with each vertex v_i in the event

graph a real number x_i such that for every arc a_{ij} we have

$$\theta_{ij} = x_j - x_i.$$

Sufficiency: If θ_{ij} is defined as given, then consider a cycle

$$\xi = (v_1, v_2, \ldots v_k, v_1).$$

Define $\xi_{a_{ij}} = +1$ if a_{ij} is directed in one sense

$\xi_{a_{ij}} = -1$ if a_{ij} is directed in the opposite sense.

Then, $\xi_{a_{12}} \theta_{12} = x_2 - x_1$

$\xi_{a_{23}} \theta_{23} = x_3 - x_2$

$\vdots \quad \vdots \quad \vdots$

$\xi_{a_{k1}} \theta_{k1} = x_1 - x_k.$

Summing, we get

$$\sum_{a_{ij} \in \xi^+} \theta_{ij} - \sum_{a_{ij} \in \xi^-} \theta_{ij} = 0.$$

Thus, the assignment θ is a potential difference assignment.

Necessity: If θ is a potential difference assignment, let us define the potentials x_i step by step.

Take an arbitrary vertex v_1 and assign the coefficient $x_1 = 0$ to it. If v_i has been labelled, and v_j has not yet been labelled, and if a_{ij} is an arc in A, then we write

$$x_j = x_i + \theta_{ij}.$$

Similarly, if a_{ji} is an arc in A, then we write

$$x_i = x_j + \theta_{ij}$$

or $x_j = x_i - \theta_{ij}$.

Since the event graphs we are interested in are strongly connected, the potential of every vertex can be assigned this way.

The potential assigned to a vertex v_i is uniquely defined; for otherwise there would exist two chains ζ_1 and ζ_2 going from v_1 to v_2 such that they form a cycle around which the potential differences do not sum to zero (see Figure 4.3.3). This would violate the definition of potential difference.

Figure 4.3.3

We now examine the conditions under which a potential difference assignment exists for directed graphs. The following Theorem holds for all <u>connected</u> graphs (i.e., not just strongly connected graphs).

<u>Theorem 4.3.3</u>: Let $G = (V,A)$ be a connected graph, and let us associate with each arc $a_{ij} \in A$ an interval $[k_{ij}, l_{ij}]$, where $k_{ij} \leq l_{ij}$. A necessary and sufficient condition for the existence of a potential difference assignment

$$\Theta = \{(a_{ij}, \theta_{ij}) \mid a_{ij} \in A\}$$

such that

$$k_{ij} \leq \theta_{ij} \leq l_{ij}$$

is that for every simple cycle ξ,

$$\left.\begin{aligned} \sum_{a_{ij} \in \xi^-} l_{ij} &\geq \sum_{a_{ij} \in \xi^+} k_{ij} \\ \sum_{a_{ij} \in \xi^+} l_{ij} &\geq \sum_{a_{ij} \in \xi^-} k_{ij} \end{aligned}\right\}$$

Proof: (Suggested by Jack Dennis; we disagree with the proof in B3).

Necessity: The condition is necessary; for, suppose such a potential difference assignment θ exists. Then, we have, for a cycle ξ,

$$\sum_{a_{ij} \in \xi^+} \theta_{ij} - \sum_{a_{ij} \in \xi^-} \theta_{ij} = 0$$

If we use the lower bound of k_{ij} for each arc in the first term and the upper bound l_{ij} for each arc in the second, we have

$$\sum_{a_{ij} \in \xi^+} k_{ij} - \sum_{a_{ij} \in \xi^-} l_{ij} \leq 0$$

i.e.,

$$\sum_{a_{ij} \in \xi^-} l_{ij} \geq \sum_{a_{ij} \in \xi^+} k_{ij}$$

By doing the converse, we get the second inequality.

Sufficiency: Here, we shall use the concept that each interval represents

a set of points on the real line. We will write

$$Y_{ij} = [k_{ij}, l_{ij}] = \{x | x \geq k_{ij}, x \leq l_{ij}\}.$$

We will use the concept of the interval sum of two intervals A and B, defined as

$$A + B = \{a + b | a \in A \text{ and } b \in B\}.$$

Also, the interval difference of two intervals A and B is defined as

$$A - B = \{a - b | a \in A \text{ and } b \in B\}.$$

The conditions of the Theorem can be written as

$$\sum_{a_{ij} \in \xi^-} l_{ij} - \sum_{a_{ij} \in \xi^+} k_{ij} \geq 0 \quad \ldots [4.3]$$

and

$$\sum_{a_{ij} \in \xi^-} k_{ij} - \sum_{a_{ij} \in \xi^+} l_{ij} \leq 0 \quad \ldots [4.4]$$

Now consider the interval

$$J = \left[\sum_{a_{ij} \in \xi^-} k_{ij} - \sum_{a_{ij} \in \xi^+} l_{ij} , \sum_{a_{ij} \in \xi^-} l_{ij} - \sum_{a_{ij} \in \xi^+} k_{ij} \right]$$

From [4.3] and [4.4],

$$0 \in J \quad \ldots [4.5].$$

But J can be rewritten as

$$J = \left[\sum_{a_{ij} \in \xi^-} k_{ij} , \sum_{a_{ij} \in \xi^-} l_{ij} \right] - \left[\sum_{a_{ij} \in \xi^+} k_{ij} , \sum_{a_{ij} \in \xi^+} l_{ij} \right]$$

$$= \sum_{a_{ij} \in \xi^-} Y_{ij} - \sum_{a_{ij} \in \xi^+} Y_{ij} \quad \ldots [4.6]$$

From [4.5] and [4.6],

$$0 \in \sum_{a_{ij} \in \xi^-} Y_{ij} - \sum_{a_{ij} \in \xi^+} Y_{ij} \quad \ldots [4.7]$$

Thus, the conditions of the Theorem can be written as in [4.7]. We shall show by induction that [4.7] implies the existence of a potential difference assignment Θ such that $k_i \leq \theta_i \leq l_i$. This is true for a graph with one arc; we shall suppose that it is true for every graph with (m - 1) arcs and show that it holds for a graph G with m arcs (labelled 1,...m) with the intervals $Y_1,\ldots Y_m$ subject to condition [4.7]. Let $\bar{G}$ be a graph with (m - 1) arcs that is obtained by deleting arc $h = (v_i,v_j)$ from G, and let $\bar{\Theta}$ be the potential difference assignment for $\bar{G}$. Let δ_h be the potential difference across the vertices v_i,v_j in the graph $\bar{G}$ under the potential difference assignment $\bar{\Theta}$. If $\delta_h \in Y_h$, then by adding arc h to $\bar{G}$ and assigning it a potential difference $\theta_h = \delta_h$, we have constructed for the graph G a potential difference assignment

$$\Theta = (\bar{\theta}_1,\ldots,\bar{\theta}_{m-1}, \delta_h).$$

If S_h does not lie within the limits $[k_h,l_h]$, then there are two possibilities:

Either (a) $\delta_h < k_h$

or (b) $\delta_h > l_h$.

We will assume without loss of generality that for all simple cycles

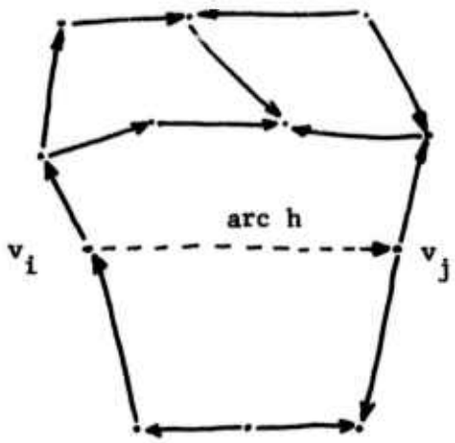


Figure 4.3.4.

$\xi \in G$ which contain the arc h, $h \in \xi^-$. In $\bar{G}$ these cycles correspond to simple paths from v_i to v_j. Thus, for each simple path S from v_i to v_j have

$$0 \in \sum_{\substack{a_{pq} \in \xi^- \\ a_{pq} \neq h}} Y_{pq} - \sum_{a_{pq} \in \xi^+} Y_{pq} + \delta_h$$

$$\text{or} \quad \delta_h \in -\sum_{\substack{a_{pq} \in \xi^- \\ a_{pq} \neq h}} Y_{pq} + \sum_{a_{pq} \in \xi^+} Y_{pq}$$

We will refer to the set of simple paths from v_i to v_j as the <u>hammock</u> $\mathcal{H}$. $\mathcal{H} = \{s | s \text{ is a simple path from } v_i \text{ to } v_j\}$.

Case (a) $\delta_h < k_h$. We will show that it must be possible to reassign potential differences to arcs in the hammock # so that $\delta_h \geq k_h$. We do this by finding a cut set C_n of the hammock #(see Figure 4.3.5).

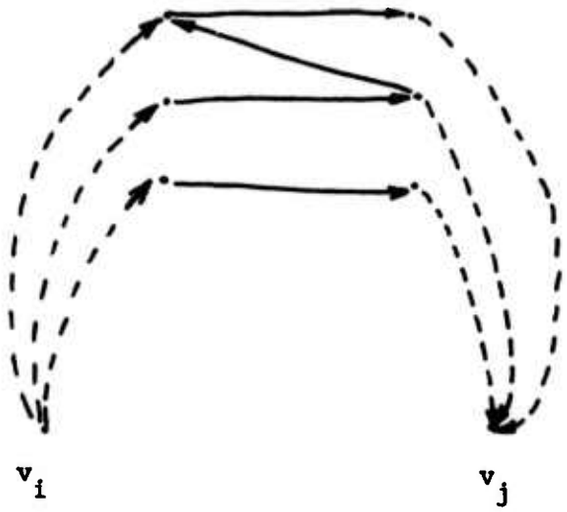


Figure 4.3.5. Hammock #.

With each arc $m = a_{pq}$ in the cut set C_n, we associate a <u>slack</u> ∂_m defined by

$$\partial_m = k_m - \theta_m \quad \text{if } m \in \xi^+.$$

$$\partial_m = \theta_m - l_m \quad \text{if } m \in \xi^-.$$

Let $\partial^n = \min [\partial_1, \ldots, \partial_r]$

$1, \ldots r \in C_n$

We increase the potential difference θ_m of each arc $m \in \xi^+$ by ∂^n and decrease the potential difference θ_m of each arc $m \in \xi^-$ by ∂^n.

Thus δ_h has been increased by ∂^n. If $\delta_h + \partial^n > k_h$, we are done. Otherwise, repeat the above procedure for some other cut set, and so on.

If no more cut sets are found for which this potential difference reassignment can be done, then there must be some path s in which the following is true:

$$\theta_m = k_m \quad \forall\ m \in \xi^+$$

$$\theta_m = l_m \quad \forall\ m \in \xi^-.$$

For as long as there is no such path, there must exist a cut set whose arcs do not have potential differences equal to either limit.

Now consider the cycle consisting of the path s and arc k.

$$\delta_k = \sum_{m \in \xi^+} l_m - \sum_{\substack{m \in \xi^- \\ m \neq k}} k_m < k_h$$

Recalling that k was defined to be in ξ^-, we see that

$$\sum_{m \in \xi^+} l_m - \sum_{m \in \xi^-} k_m < 0$$

which contradicts the first condition of the Theorem, namely that

$$\sum_{a_{ij} \in \xi^-} l_{ij} \geq \sum_{a_{ij} \in \xi^+} k_{ij}$$

Similarly, we show that for Case (b), i.e., $S_h > l_h$, it must be possible to reassign the potentials in the hammock # so that $\delta_h \leq l_h$. If not, we can show by a method similar to that for case (a) that there must exist a path for which

$$\sum_{a_{ij} \in \xi^+} l_{ij} < \sum_{a_{ij} \in \xi^-} k_{ij}$$

which contradicts the second condition of the Theorem.

This proves the Theorem. ∎

Corollary 4.3.3.1

A potential difference assignment θ such that $\theta_{ij} \geq k_{ij}$ (for all $a_{ij} \in G$) exists if and only if, for every circuit ξ,

$$\sum_{a_{ij} \in \xi} k_{ij} \leq 0$$

Proof: Set $l_{ij} = +\infty$ for every a_{ij}. There are two types of cycles:

(a) circuits, i.e., cycles in which all arcs are oriented in the same direction.

(b) cycles which are not circuits.

For every circuit ξ, we see that

$$\sum_{a_{ij} \in \xi.} k_{ij} \leq 0.$$

For cycles which are not circuits, we see that

$\sum k_{ij} \leq \infty$ which is vacuously true if all k_{ij}'s are finite. ∎

We are now in a position to show that the bound we obtained on the computation rate of transitions in a timed event graph is actually attainable. This forms the subject of our next theorem.

Theorem 4.3.4: Let $\mathcal{G} = (G,\Omega)$ be a timed event graph. $\mathcal{G}$ has a strongly periodic firing schedule with period $\pi > 0$ iff

$$\pi \geq \max_{\forall k} \frac{\sum_{v_i \in C_k} \tau_i}{\sum_{a_{ij} \in C_k} M(0,a_{ij})}$$

where $\{C_1, C_2, \ldots C_k\}$ is the set of all circuits in G.

Proof: From Corollary 4.3.1.1 we have the result that a timed event graph $\mathcal{G}$ has a strongly periodic firing schedule with displacements $x_1, \ldots x_m$ and period π iff

$$\forall\, a_{ij} \in G, \text{ we have } x_j - x_i \geq c_{ij}$$

where $c_{ij} = \tau_i - \pi M(0,a_{ij})$. [4.2]

Now with each arc a_{ij}, we associate $\theta_{ij} = x_j - x_i$.

By Theorem 4.3.2, a function that assigns such a θ_{ij} to each arc a_{ij} must be a potential difference assignment. The set of inequalities [4.2] now becomes

$$\theta_{ij} \geq c_{ij} \quad \forall\, a_{ij} \in G \qquad [4.8]$$

By Corollary 4.3.3.1, this system of inequalities has a solution iff, for every circuit $\xi \in G$,

$$\sum_{a_{ij} \in \xi} c_{ij} \leq 0 \qquad [4.9]$$

[4.9] now becomes

$$\forall\, \xi \in G, \quad \sum_{a_{ij} \in \xi} \left(\tau_i - \pi\, M(0,a_{ij}) \right) \leq 0$$

$$\text{or} \quad \sum_{v_i \in \xi} \tau_i \; - \; \pi \sum_{a_{ij} \in \xi} M(0,a_{ij}) \leq 0 \qquad \forall\, \xi \in G,$$

$$\text{or} \quad \pi \geq \frac{\sum_{v_i \in \xi} \tau_i}{\sum_{a_{ij} \in \xi} M(0,a_{ij})} \qquad \forall\, \xi \in G$$

This is true iff

$$\pi \geq \max_{\xi \in G} \left\{ \frac{\sum_{v_i \in \xi} \tau_i}{\sum_{a_{ij} \in \xi} M(0,a_{ij})} \right\} \qquad . \quad . \quad . \qquad [4.10]$$

■

Comments on Theorem 4.3.4: We note that there exists a strongly periodic firing schedule with period

$$\pi = \max_{\xi \in G} \left\{ \frac{\sum_{v_i \in \xi} \tau_i}{\sum_{a_{ij} \in \xi} M(o,a_{ij})} \right\}$$

The computation rate p of transitions in G is given by $p = \frac{1}{\pi}$.

Hence,

$$\rho \leq \rho' = \min_{\xi \in G} \left\{ \frac{\sum_{a_{ij} \in \xi} M(0, a_{ij})}{\sum_{v_i \in \xi} \tau_i} \right\} \qquad [4.11]$$

It is clear from the inequality [4.7] that there exists a strongly periodic firing schedule which realizes a computation rate

$$\rho = \rho'$$

Thus, the timed event graph can be allowed to "run" at any "speed" less than or equal to ρ'. In Section 4.2, we argued that the computation rate of transitions in a timed event graph could not exceed that of transitions in the circuit with the minimum ratio of

$$\frac{\sum_{a_{ij} \in \xi} M(0, a_{ij})}{\sum_{v_i \in \xi} \tau_i}$$

but our arguments were non-rigorous. In this section we have substantiated our claim in a rigorous fashion. The circuit with the minimum ratio of token content to sum of transition firing times is termed the critical circuit, and as we have seen, this is the one which determines the maximum computation rate of all transitions in the net.

Section 4.4 Timed LSP Nets

We now proceed to apply the result obtained in the previous Section to determining the maximum computation rate of transitions in a timed LSP Petri net. Our starting point will be to decide if a given marked Petri net is indeed LSP. This can be done by applying the results developed in Section 2.3.

Consider a timed LSP Petri net $\mathcal{L} = \langle \mathcal{P}, \Omega \rangle$. The timed Petri net in Figure 4.2.1 is an LSP net, and we saw that we were able to construct a feasible periodic schedule for it. We now show that every LSP Petri net has a feasible periodic firing schedule, and we show how to find the period and the computation rate of every transition in the net.

Recall from Section 3.4 that the steady state behavior of an LSP net $\mathcal{P}$ can be represented by the steady-state equivalent net $\mathcal{S}$. Furthermore, recall that $\mathcal{S}$ has the structure of a multiply-labelled event graph. The only difference between event-graphs and multiply-labelled event-graphs is that that the latter may have multiple instances of certain transitions. We will begin by assuming that all transitions in a multiply-labelled event graph have distinct labels, and the steady-state equivalent net $\mathcal{S}$ can be regarded as an event graph. Each transition is assigned the firing time of the transition it corresponds to before the relabelling was done. The computation rate of transitions in the resulting event graph can be found by applying the result of [4.11].

We showed in Chapter 3 that the steady state equivalent net of a consistent LSP Petri net must have a number of occurrences of each

transition equal to its current in a consistent current assignment. We now show that the computation rate of transitions in a timed LSP Petri net must be proportional to their currents in a consistent current assignment. To show this, let us consider the steady-state equivalent net $\mathcal{S}$. We have seen that we can find the computation rate of transitions in the relabelled net by treating it as an event graph. We will call this the fundamental computation rate of the LSP net, and will denote it by ρ. If the multiplicity of any transition v_i is $\mu(v_i)$, it means that in unit time, transition v_i in the timed LSP net $\mathcal{L}$ fires $\mu(v_i) \times \rho$ times, because each firing of an instance of transition of transition v_i in $\mathcal{S}$ is also a firing of transition v_i in $\mathcal{P}$. Hence, we can find the computation rate of any transition v_i by simply multiplying ρ by the multiplicity of that transition in $\mathcal{S}$.

Summarizing, the following steps are are involved in finding the computation rate of transitions in a timed LSP Petri net $\mathcal{L}$:

Step 1 Find the steady state equivalent net $\mathcal{S}$ of the underlying LSP Petri net $\mathcal{P}$.

Step 2 Treating $\mathcal{S}$ as an event graph, find its maximum computation rate ρ by applying [4.11]. This gives the maximum fundamental computation rate of the timed LSP Petri net $\mathcal{L}$.

Step 3 If ρ_i is the maximum computation rate of transition v_i in $\mathcal{L}$, and $\mu(v_i)$ is its multiplicity in $\mathcal{S}$, then $\rho_i = \rho \times \mu(v_i)$.

We illustrate this method by an example.

Example Consider the timed LSP Petri net of Figure 4.2.1 (reproduced in Figure 4.4.1). In order to make our example as general as possible,

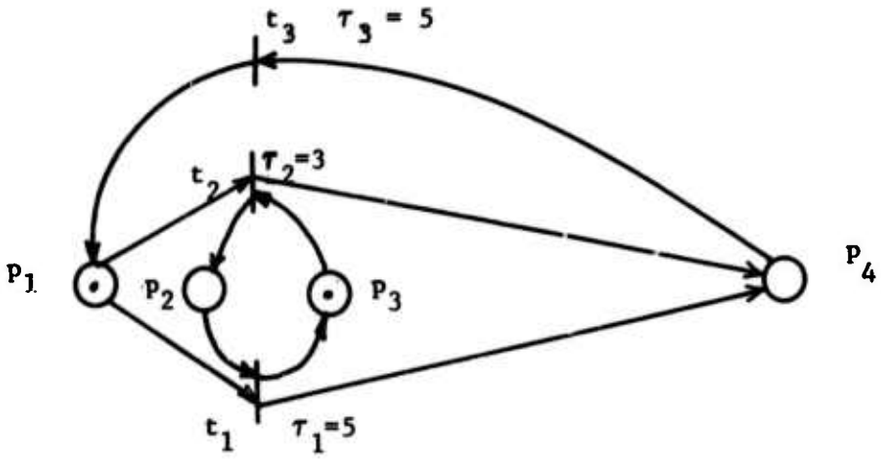


Figure 4.4.1 Timed LSP Petri net $\mathcal{L} = \langle \mathcal{P}, \Omega \rangle$ with a live, bounded marking M.

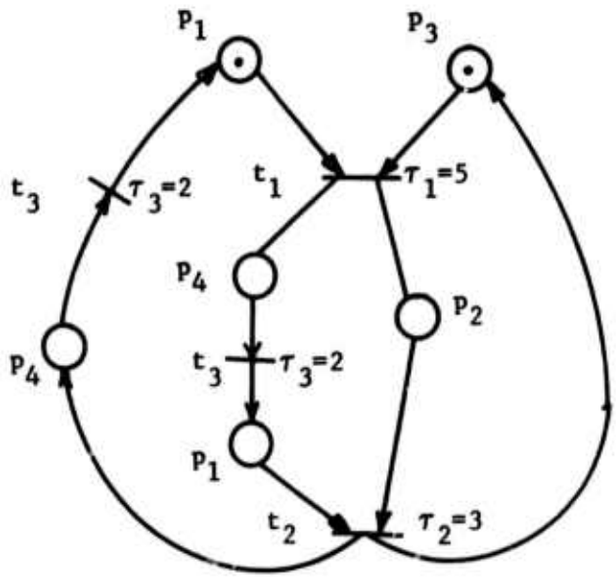


Figure 4.4.2 Steady state Equivalent net $\mathcal{S}$ for LSP Petri net $\mathcal{P}$ with marking M.

the net has a marking M that is not safe, but which is live, bounded and persistent. In Figure 4.4.2, we show the steady state equivalent $\mathcal{S}$ of the net $\mathcal{P}$ with marking M. $\mathcal{S}$ can be decomposed into four circuits :

C_1: $p_1 t_1 p_4 t_3 p_1 t_2 p_4 t_3 p_1$

C_2: $p_3 t_1 p_2 t_2 p_3$

C_3: $p_1 t_1 p_2 t_2 p_4 t_3 p_1$

C_4: $p_3 t_1 p_4 t_3 p_1 t_2 p_3$

The fundamental computation rate ρ is seen to be

$$\rho = \min[\, 2/12,\ 1/8,\ 2/10,\ 1/10]$$

or $\rho = 1/10.$

Now, $\mu(t_1) = 1$

$\mu(t_2) = 1$

$\mu(t_3) = 2$

Thus the computation rates of transitions t_1, t_2 and t_3 are as follows:

$\rho_1 = 1/10$

$\rho_2 = 1/10$

$\rho_3 = 2/10.$

A note on the Structure of LSP Petri nets

Although it is true that the LSP Petri nets that arise in the course of modelling practical systems will all be SMD there do exist Petri nets which are LSP but not SMD. The net in Figure 4.4.3 is an LSP Petri net, but it is not SMD as the reader can easily verify. The method we have given for finding the computation rate for timed LSP Petri nets holds for all LBP Petri nets which have a live, safe marking.

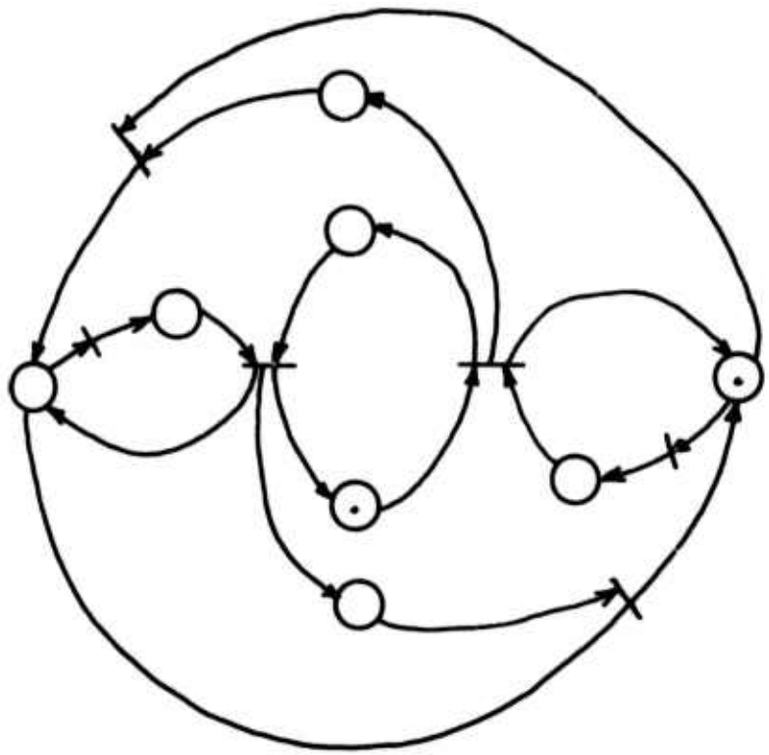


Figure 4.4.3 An LSP Petri net that is not SMD

4.5 Timed Petri nets with random firing times

The assumption of deterministic firing times for Petri nets has given us a great deal of insight into the action of practical asynchronous systems. We have been able to find the computation rate of transitions in timed event graphs and LSP Petri nets, and in each case we have been able to identify system components that limit the fundamental computation rate of the net (i.e., the critical circuit). The critical circuit represents a bottleneck in the system being modelled, and readers familiar with PERT networks will see the correspondence between critical circuits in timed event graphs and critical paths in PERT networks.

The assumption of deterministic firing times may not be sufficiently accurate for practical systems. In practice, an action in an asynchronous system like an addition will be a random variable , and in a large number of cases this can be approximated by either a rectangular or a Gaussian distribution (see Figures 4.5.1 and 4.5.2 respectively). A Gaussian distribution has the problem that it has a "tail" which extends for negative values of time. Since we cannot visualize an action taking less than zero time to occur, we would prefer to use the rectangular distribution shown in Figure 4.5.1.

It is possible to find out the mean computation rate of a timed event graph when all firing times are given by a distribution like the rectangular distribution. However, the process is very tedious, and we will work with the means of each of the firing times. By doing this, we get a timed event graph whose computation rate can be found by applying Equation [4.11]. It can be shown by using the results of Clark[C3] that the computation rate so obtained is an upper bound on the true computation rate of the timed event graph whose firing times are random variables.

The deviation of the true mean computation rate from this bound depends upon on the standard deviations of the random variables, but there does not appear to be any simple expression that relates these quantities.

We now turn to Chapter 5, where we find a bound on the computation rate of timed SMD Petri nets.

CHAPTER 5

COMPUTATION RATE OF ASYNCHRONOUS SYSTEMS WITH DECISIONS

5.1 Timed SMD Petri Nets

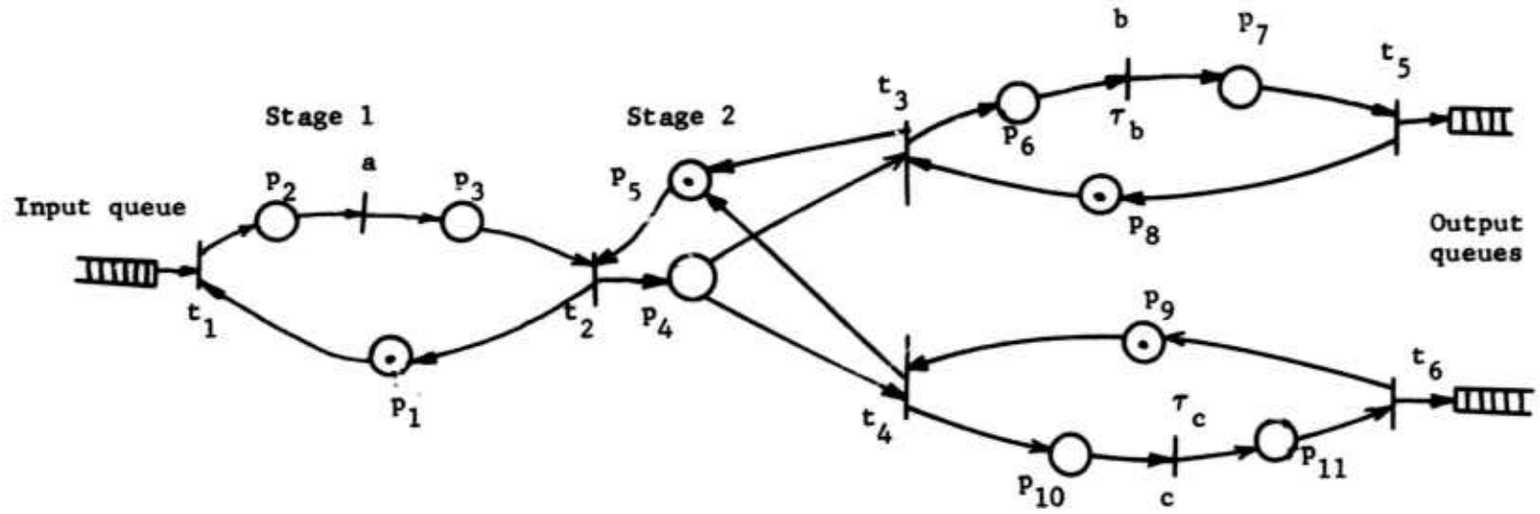
In Chapter 3, we illustrated the power of SMD Petri nets in modelling asynchronous systems with decisions. We also established two important results about SMD Petri nets:

(a) It is decidable if any given marking for an SMD Petri net is live.

(b) For SMA Petri nets, any marking which puts at least one token on every state machine is live.

Thus, using the results presented in Chapter 3, we can design asynchronous processing systems for possible implementation by the techniques developed by Dennis and Patil [D1, D2]. We now show how we can estimate the computation rate of this more general class of systems. Recall that in Chapter 4 we developed a technique for finding the maximum computation rate of a large class of deterministic systems. In this chapter, we look at non-deterministic systems and see if we can obtain a bound on their computation rate. We do this by finding a bound on the computation rate of transitions in a timed SMD Petri net.

We have defined timed Petri nets in Section 4.1 (see definition 4.1.1). Thus, a timed SMD Petri net is an SMD net in which every transition has a fixed, non-zero firing time. For details of the firing mechanism and notation for the instantaneous marking of a timed SMD net, the reader is urged to re-read Section 4.1.



Decision made at place p_4.

Figure 5.1.1.

Let us now consider a timed SMD Petri net and examine its behavior. We do this by constructing a firing schedule for it, and interpreting the notion of computation rate of a transition in connection with the firing schedule.

As a simple example to motivate this chapter, consider the SMD Petri net model of an assembly line with decisions of the type discussed in Section 1.2. Figure 5.1.1 is such a Petri net model. Let transitions a, b and c represent assembly operations, and let p be a place which represents a bay at which a decision is made about which engine should be attached to a chassis. The engines are available in the input queues connected to transitions b and c, and completely assembled autos are output into the queues marked "output queues." Each of the assembly operations represented by the transitions a, b and c have associated time durations τ_a, τ_b, and τ_c respectively. A partially assembled auto that appears at place p can be routed one of two ways, by firing either transition d or transition e. We said in Section 1.2 that this routing can be done in several ways, and the exact mechanism of making this routing decision does not concern us here. We have pointed out some of the ways in which this decision can be made for a practical assembly line. From a performance standpoint, we must know the relative numbers of each type of automobile which are produced by the assembly line. Let us see why. Suppose a large proportion of all cars produced by the assembly line are 200 HP cars, and suppose that the final assembly process for attaching a 200 HP engine to a chassis is extremely slow (i.e., transition f has a long firing time). This assembly stage will not be able to handle the load imposed by it,

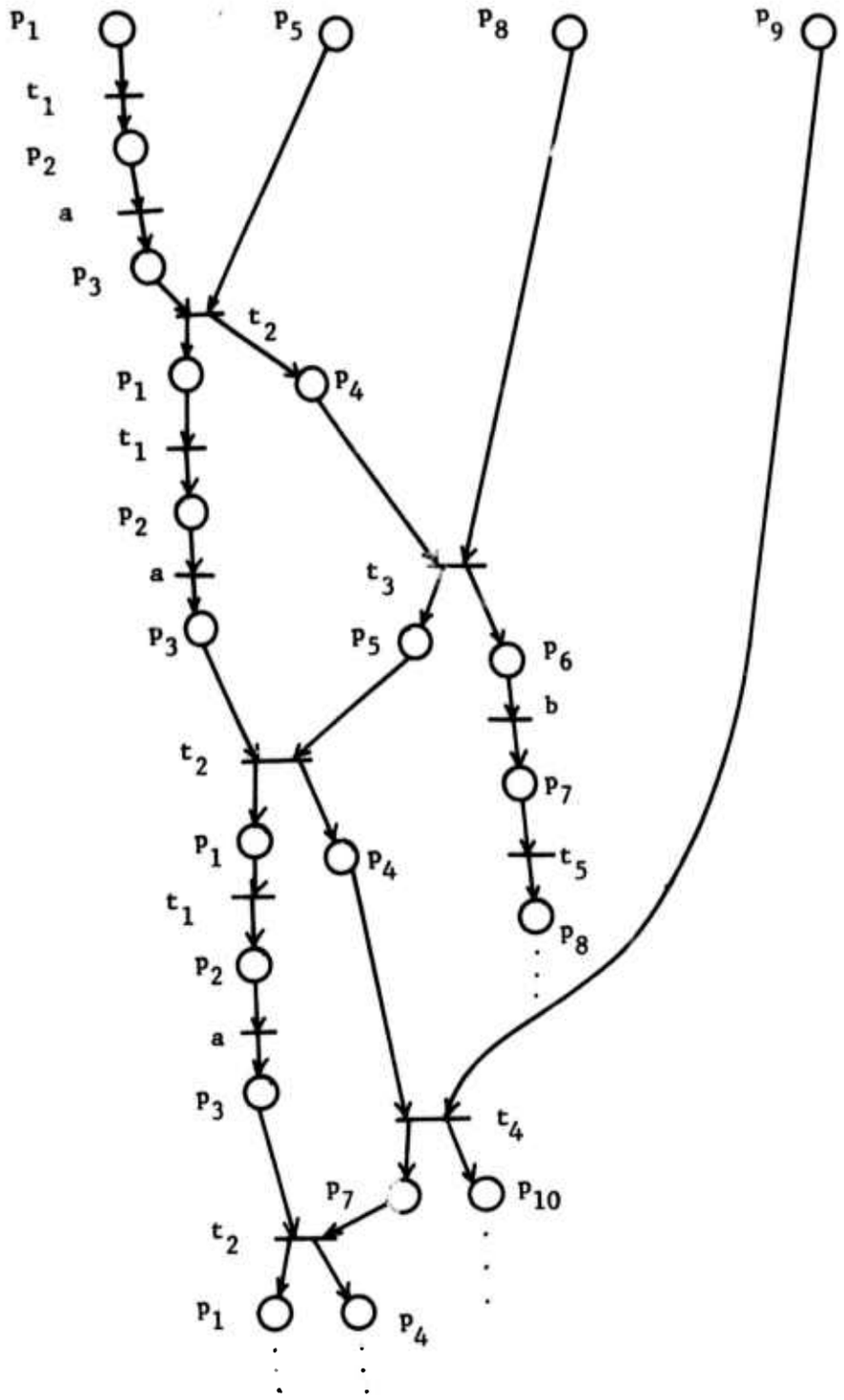


Figure 5.1.2 Occurrence Graph for Petri Net in Figure 5.1.1.

and will slow down the operation of the whole line by choking off the absorption of subassemblies from the input queues. It is intuitively obvious, therefore, that this assembly line can be balanced by matching the speed of operation of an assembly station to its expected frequency of use. It can also be seen that there is a tradeoff between the number of physical processing units at an assembly station and the speed of each processing unit - a number of slow processing units can be used instead of a few fast ones, and vice versa. Our question in connection with this assembly line is - how do we estimate the rate at which it produces assembled autos?

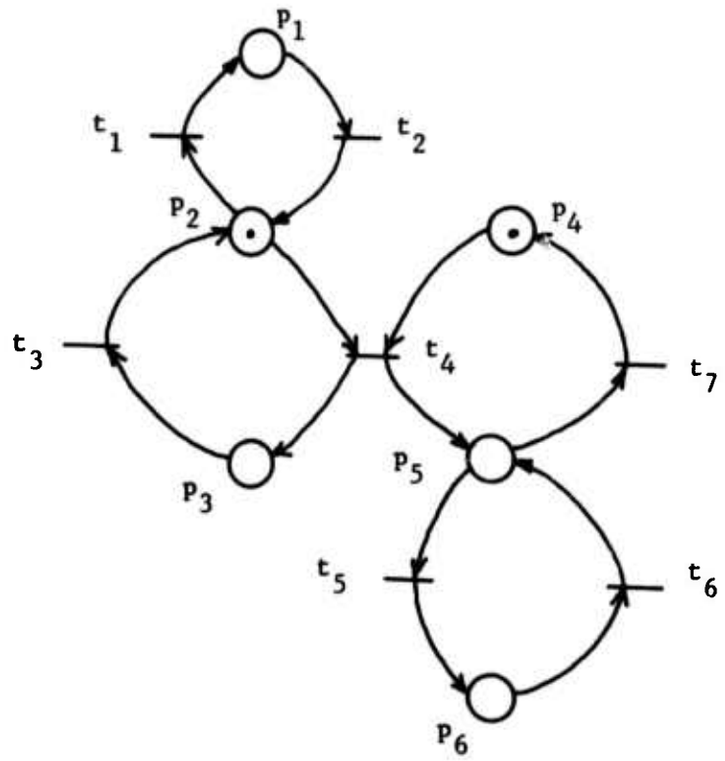
Let us examine our model carefully and see what information is needed to specify the system in a way which is complete for our purpose. We have shown input queues connected to some of the transitions in the net in Figure 5.1.1. We pointed out in Section 1.2 that we will assume that there is always at least one item in each input queue, so that it is not necessary to indicate the queues, but we have shown the queues for completeness. Let us now see how to incorporate the relative frequency of use of the two alternative assembly stations into out model.

The action of the Petri net in Figure 5.1.1 can be represented by means of an occurrence graph (see Section 3.5.3). In the occurrence graph shown in Figure 5.1.2, the probability or the relative frequency of occurrence of each of the outcomes of the decision made at place p_4 will be reflected in the number of occurrences of the corresponding transition in a long frustrum of the occurrence graph. For example, if the probabilities of occurrence of transitions d and e are $\frac{1}{3}$ and $\frac{2}{3}$ respectively, then, in a long frustrum of the occurrence graph there

will on an average, be twice as many occurrences of transition e as of transition d. This means that in a consistent current assignment for the Petri net, the current assigned to transition e will be twice that assigned to transition d. Our problem is to find the maximum computation rates of transitions t_5 and t_6 in the net. In general, we would like to find the maximum computation rate of any transition in a timed SMD net, for which a consistent current assignment is given. We wish to find a computation rate assignment such that the computation rate of each transition is proportional to its current in the given consistent current assignment.

The problem can be tackled by simulation. In such a method the computation rate of any transition can be found by letting the net "run" for a long time, and then dividing the number of times that transition has fired by the total amount of time that has elapsed since time zero. The method is extremely time consuming and we would like an analytical technique which enables us to find a good bound on the computation rate of transitions in the timed net.

Consider a timed SMD net $\chi = \langle \mathcal{P}, \Omega \rangle$, having a minimal integer consistent current assignment Φ. We know from Section 3.5 that several c-equivalent nets are, in general, possible for χ with consistent current assignment Φ. In Figure 5.1.3(a) we reproduce the net shown in Figure 3.5.4. For the given minimal consistent current assignment, several c-equivalent nets are possible, and we give two of them (see Figures 5.1.3(b) and 5.1.3(c). Each timed c-equivalent net defines a periodic schedule. We compute their fundamental computation rates, and get the following:



The SMD net χ.

Minimal integer consistent current assignment:

$\varphi_1 = \varphi_2 = 1$ $\varphi_5 = \varphi_6 = 2$

$\varphi_3 = \varphi_4 = \varphi_7 = 2$

Firing time assignment:

$\tau_1 = 2, \tau_2 = 3, \tau_3 = 4, \tau_4 = 1, \tau_5 = 3, \tau_6 = 5, \tau_7 = 2$

Figure 5.1.3(a)

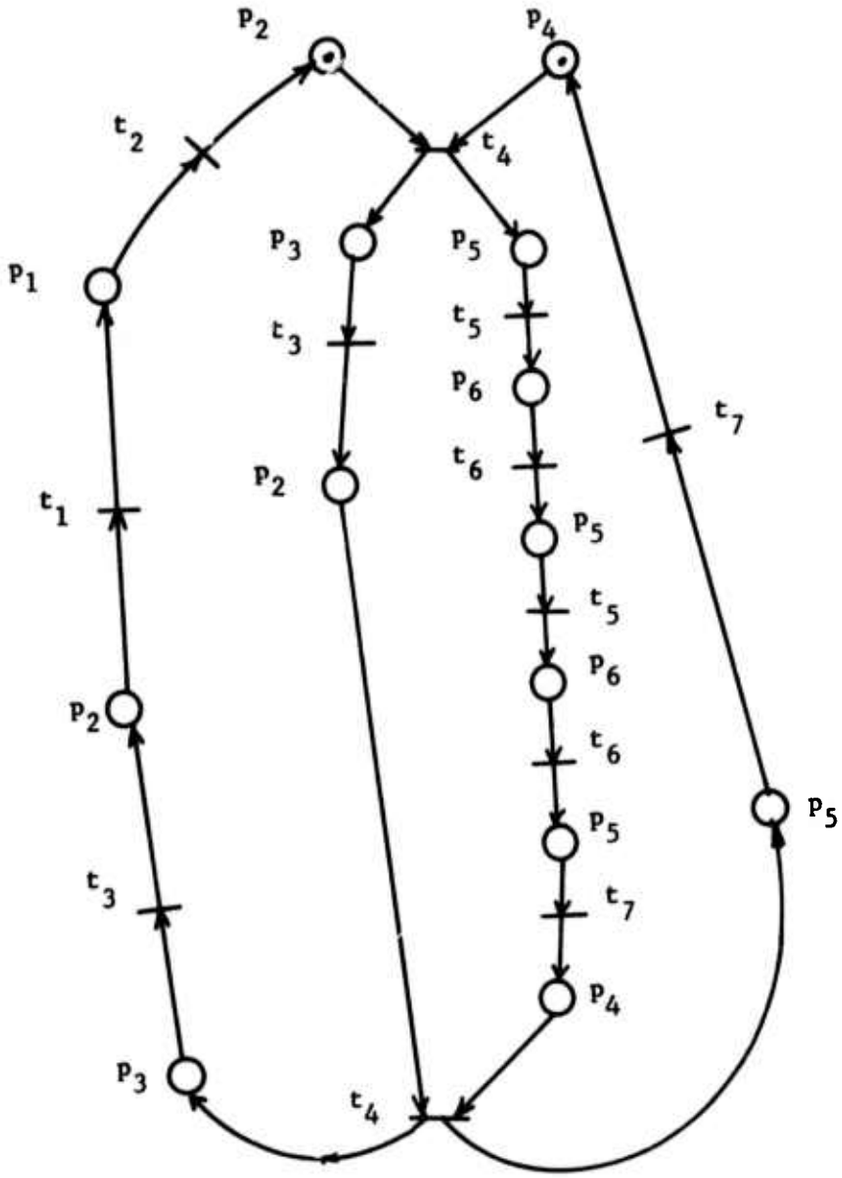

p2
p4
t2
t4
p1
p3
p5
t3
t5
p6
t7
t6
p2
t1
p5
t5
p6
p2
t6
p5
p5
t3
t7
p4
p3
t4

Figure 5.1.3(b)

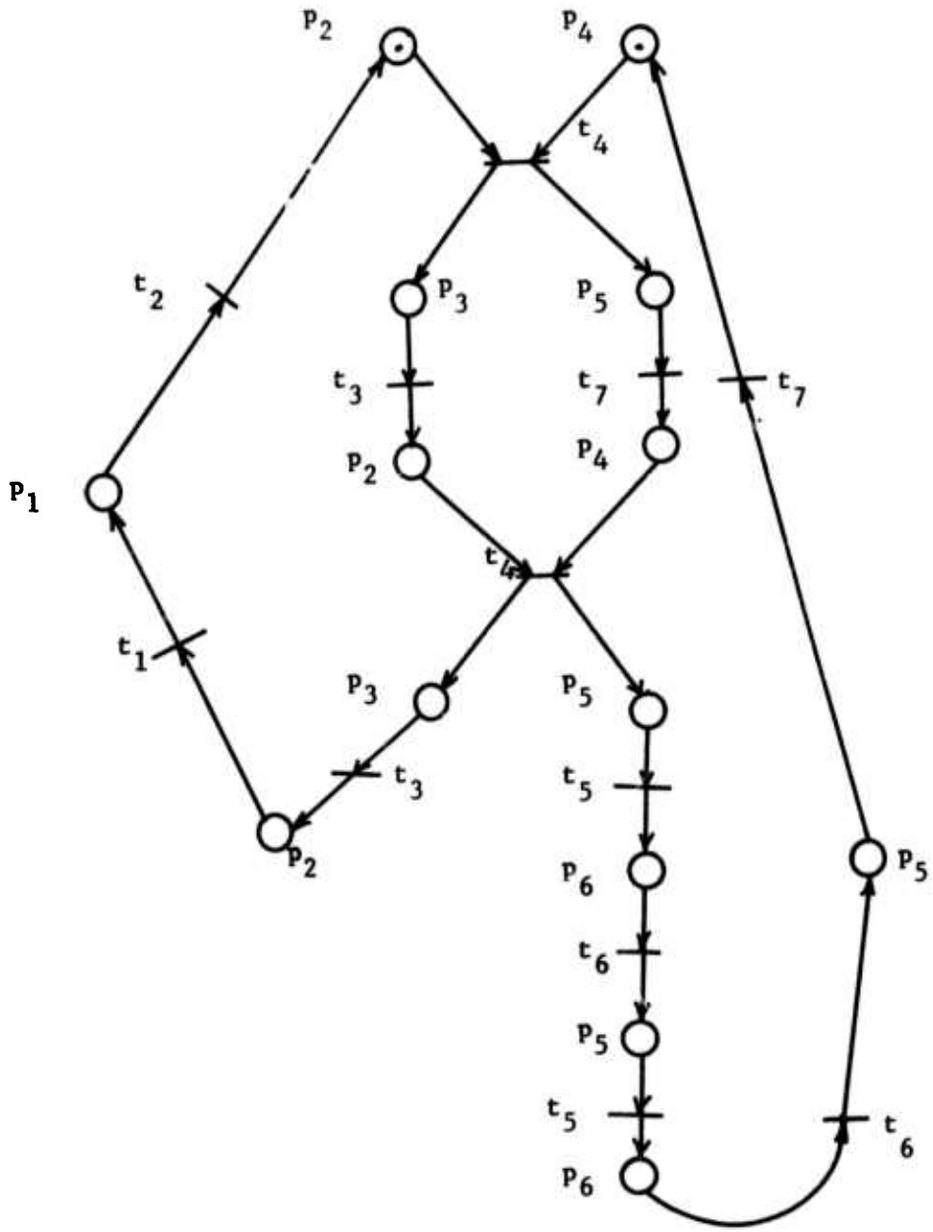

p2
p4
t4
t2
p3
p5
t3
t7
t7
p2
p4
p1
t4
t1
p3
p5
t3
t5
p2
p6
p5
t6
p5
t5
t6
p6

Figure 5.1.3(c)

Figure 5.1.3 (b)

Applying the results of Section 4.4,

Fundamental Computation Rate $\rho' = \min\left[\frac{1}{15}, \frac{1}{29}, \frac{1}{8}, \frac{1}{22}\right] = \frac{1}{29}$.

Figure 5.1.3 (c)

Fundamental Computation Rate $\rho' = \min\left[\frac{1}{15}, \frac{1}{22}, \frac{1}{13}, \frac{1}{24}\right] = \frac{1}{24}$.

Thus, the maximum Fundamental Computation Rates of the two c-equivalent nets are different. In order to find a bound on the maximum fundamental computation rate of the timed net χ, we must find the timed c-equivalent net which has the largest computation rate.

The maximum fundamental computation rate of a c-equivalent net for the timed net χ represents the maximum fundamental computation rate of transitions in the net χ for the behavior specified by that c-equivalent net. This leads to the following definition:

Definition 5.1.1 The maximum fundamental computation rate of transitions in a timed SMD net $\chi = \langle \mathcal{P}, \Omega \rangle$ for a minimal integer consistent current assignment Φ is given by the fundamental computation rate of the c-equivalent net which has the largest fundamental computation rate.

Definition 5.1.2 The maximum computation rate of a transition t_i belonging to a timed SMD Petri net $\chi = \langle \mathcal{P}, \Omega \rangle$ with a minimal integer consistent current assignment Φ is given by

$$\rho_i = \varphi_i \times \rho$$

where φ_i is the current assigned to t_i by Φ and ρ is the maximum funda-

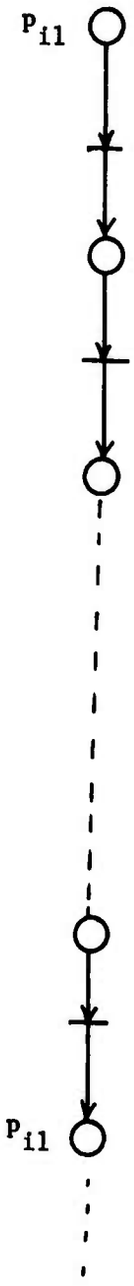


Figure 5.1.4

mental computation rate of χ.

We see that to obtain an exact value of the maximum fundamental computation rate of χ, we must find all the c-equivalent nets for χ and find the one with the largest fundamental computation rate. This is a very tedious process, and we would like to find a simple method which gives a bound on the fundamental maximum computation rate of χ.

Consider the timed SMD Petri net $\chi = (\mathcal{P},\Omega)$ in which the underlying Petri net $\mathcal{P}$ has a live marking M. Let $S_1, S_2, \ldots S_m$ be the state machine components of $\mathcal{P}$. Let the transitions of any state machine component S_i be $t_{i1}, t_{i2}, \ldots t_{ik}$, their firing times be $\tau_{i1}, \tau_{i2}, \ldots \tau_{ik}$ and their currents be $\varphi_{i1}, \varphi_{i2}, \ldots \varphi_{ik}$ respectively. Also, let $p_{i1}, p_{i2}, \ldots p_{ik}$ be the places belonging to the state machine S_i. Let n_i denote the number of tokens on state machine S_i.

Consider the occurrence graph associated with one of the tokens on the state machine S_i (see Figure 5.1.4).

Since there are n_{i1} tokens on the state machine S_i, the maximum natural computation rate of transition t_{i1} is thus $n_i \Psi_{i1}$ or $n_i \varphi_{i1} \Psi_i$. The quantity Ψ_i is termed the maximum natural fundamental computation rate of the state machine, and to obtain the maximum natural computation rate of transition k_{i1}, we multiply Ψ_i by φ_{i1}. The term maximum natural computation rate of a transition t_{i1} refers to the computation rate it would have if the state machine S_i were isolated from the other components of the SMD Petri net $\mathcal{P}$. However, as we saw in the case of timed event graphs, the components of a timed net affect each other's

fundamental computation rate. We now show that if two state machines S_i and S_j have some transition t in common, then the fundamental computation rate of <u>both</u> state machines is min $[\Psi_i, \Psi j]$. Let φ be the current associated with transition t in a consistent current assignment. Then, it is clear that the computation rate of t cannot exceed $\varphi \times \min [\Psi_i, \Psi_j]$. For suppose without loss of generality that $\Psi_i < \Psi_j$. Then, the computation rate of t cannot be greater than $\varphi \times \Psi_i$. If it could be greater than $\varphi \times \Psi_i$, it would violate the definition of the fundamental computation rate of S_i. Thus, the computation rate of t cannot exceed $\varphi \times \min [\Psi_i, \Psi_j]$. Also, by consistency, the computation rate of any transition $t_{il} \in S_i$ must equal

$$\frac{\varphi_{il}}{\varphi} \times \varphi \times \min [\Psi_i, \Psi_j] = \varphi_{il} \times \min [\Psi_i, \Psi_j].$$

Similarly, the computation rate of any transition t_{jm} in S_j equals

$$\frac{\varphi_{jm}}{\varphi} \times \varphi \times \min [\Psi_i, \Psi_j] = \varphi_{jm} \times \min [\Psi_i, \Psi_j].$$

We conclude that if two state machine components S_i and S_j in a timed SMD Petri net have a transition in common, each state machine acquires a fundamental computation rate = min $[\Psi_i, \Psi_j]$.

Let $\rho = \min [\Psi_1, \Psi_2, \ldots \Psi_m]$. Without loss of generality, let $\rho = \Psi_1$. We now construct sets $\lambda_1, \lambda_2, \ldots \lambda_r$ of state machines as follows:

If λ_p is a set of state machines, then define λ_{p+1} to be the set $\lambda_p \cup \{S_{p1}, S_{p2}, \ldots S_{pr}\}$ where $S_{p1}, S_{p2}, \ldots S_{pr}$ are state machines which have at least one transition in common with the state machines in λ_p.

Also, $\lambda_r = \{S_1, S_2, \ldots S_m\}$, i.e., all the state machine components of $\mathcal{P}$.

Construct λ_{p+1} from λ_p until $\lambda_{p+1} = \lambda_p$. Then λ_p must include all the state machines because $\mathcal{P}$ is strongly-connected.

The reader can now see that if the maximum fundamental computation rate of $S_1 = \Psi_1$, then the maximum fundamental computation rate of all state machines in λ_2 is also Ψ_1, and so on until we have exhausted all state machines in the net $\mathcal{P}$. Thus, all state machines in χ must have a maximum fundamental computation rate $\rho = \Psi_1 = \min [\Psi_1, \Psi_2, \ldots \Psi_m]$ where $S_1, S_2, \ldots S_m$ are all the state machine components of $\mathcal{P}$.

The maximum computation rate of any transition t_i in the net is thus

$\varphi_i \times p$.

We have thus established the following result:

In a timed SMD Petri net $\chi = \langle \mathcal{P}, \Omega \rangle$, the maximum computation rate ρ_i of any transition t_i is given by

$$\rho_i \leq \varphi_i \times \rho'$$

where $\rho' = \min [\Psi_1, \Psi_2, \ldots \Psi_m]$

where $\Psi_1, \Psi_2, \ldots \Psi_m$ are the maximum fundamental computation rates of $S_1, S_2, \ldots S_m$. The maximum fundamental computation rate Ψ_k of state machine S_k is given by

$$\Psi_k = \frac{n_k}{\sum_{j=1}^{r} \varphi_{kj} \tau_{kj}}$$

where n_k = number of tokens on state machine S_k. $t_{k1}, t_{k2}, \ldots t_{kr}$ are the transitions of state machine S_k. φ_{kj}, τ_{kj} are, respectively, the current in a consistent current assignment for $\mathcal{P}$ and firing time of

transition $t_{kj} \in S_k$.

The bound we have obtained here is based on the assumption that when a token appears at a place with conflict, then its further routing (i.e., which output transition to fire) is done statistically, using an a-priori probability measure proportional to the currents associated with the respective transitions. In the next section, we derive this bound by considering the c-equivalent nets for X.

5.2 Rigorous Derivation of Bound on Fundamental Computation Rate for Timed SMD Petri Nets

In the previous section, we saw how we can find the upper bound on the maximum fundamental computation rate of a timed SMD Petri net χ with a minimal integer consistent current assignment. This involved drawing all the timed c-equivalent nets for χ and finding the one with the largest maximum fundamental computation rate. As we pointed out, this is a tedious process. The following Theorem gives an upper bound on the maximum fundamental computation rate of a timed SMD Petri net in terms of its structure, marking, minimal integer consistent current assignment and firing time assignment.

Theorem 5.2.1 In a timed SMD Petri net $\chi = \langle \mathcal{P}, \Omega \rangle$ with a minimal integer consistent current assignment Φ, the maximum fundamental computation rate is given by $\rho' = \min [\Psi_1, \ldots \Psi_m]$. $\Psi_1 \ldots \Psi_m$ are the fundamental computation rates of the state machine components of χ. The fundamental computation rate Ψ_k of state machine S_k is given by

$$\Psi_k = \frac{n_k}{\sum_{j=1}^{r} \varphi_{kj} \tau_{kj}}$$

where n_k = number of tokens on state machine S_k.

$t_{k1}, \ldots t_{kr}$ are the transitions of state machine S_k.

φ_{kj}, τ_{kj} are the current and firing time respectively of transition $t_{kj} \in S_k$.

Proof: Consider the c-equivalent net $\mathcal{C}$ of the SMD net $\mathcal{P}$ for the consistent current assignment Φ. We showed in Section 3.5 that for every state machine component S_k in $\mathcal{P}$, there exists a corresponding circuit $C_k \in \mathcal{C}$ with the following property:

Every transition t_i in C_k has a multiplicity $\mu(t_i)$ equal to the current φ_i assigned to transition t_i in $\mathcal{P}$ by the minimal consistent current assignment Φ. Now let $C_1,\ldots,C_m$, $C_{m+1},\ldots C_r$ be the simple circuits in $\mathcal{C}$, where $C_1,\ldots C_m$ correspond to state machines of $\mathcal{P}$ and $C_{m+1},\ldots C_r$ are simple circuits in $\mathcal{C}$ that do not correspond to state machines of $\mathcal{P}$. Let $\Psi_1,\ldots\Psi_m$, $\Psi_{m+1},\ldots,\Psi_r$ be their respective fundamental computation rates. For any circuit $C_i \in \{C_1,\ldots C_m\}$

$$\Psi_i = \frac{n_i}{\sum \varphi_{ij}\tau_{ij}}$$

$$\forall\, t_{ij} \in C_i$$

Now, the fundamental computation rate ρ of the timed net χ is given by

$$\rho' = \min\,[\Psi_1,\ldots\Psi_r]$$

$$= \min\,[\Psi_1,\ldots\Psi_m,\ \Psi_{m+1},\ldots,\Psi_r]$$

or $\rho' \leq \min\,[\Psi_1,\ldots\Psi_m]$

■

Note:

There may exist some $\psi_s \in \{\Psi_{m+1},\ldots,\Psi_r\}$ such that

$\psi_s < \min\,[\Psi_1,\ldots\Psi_m]$.

Thus, while $\rho' = \min\,[\Psi_1,\ldots\Psi_m]$ is certainly a bound, this bound may not be achievable. The computation rate ρ_i of any transition t_i is defined

to be

$$\rho_i = \rho' \times \varphi_i.$$

Once again, this value of ρ_i is a bound, but this bound may not be achievable. Theorem 5.2.1 enables us to find a bound on the computation rate of any transition in a timed SMD Petri net by finding for each state machine component S_i the corresponding fundamental computation rate ψ_i. This is simpler than finding the maximum fundamental computation rate of all the c-equivalent nets for the SMD Petri net.

Example: Let us apply this result to the timed SMD Petri net of Figure 5.1.3. Theorem 5.2.1 gives a bound on the maximum fundamental computation rate equal to

$$\min [\ 1/15,\ 1/22] = 1/22.$$

By drawing all the c-equivalent nets for the SMD net, the reader can verify that the net of Figure 5.1.3(c) has the largest maximum fundamental computation rate of 1/24. The bound of Theorem 5.2.1 cannot actually be realized in any consistent behavior of the SMD net, but we see that it is certainly a reasonable bound. The reason that the bound is not achievable is that there is no state machine component in $\mathcal{P}$ that corresponds to the multiply-labelled circuit $p_4t_4p_3t_3p_2t_4p_5t_5\ldots t_7p_4$. This circuit has a maximum natural fundamental computation rate of 1/24.

5.3 Achievability of Bound on Computation Rate for timed SMD Petri nets

The bound on the computation rate of transitions in a timed SMD Petri net that we have presented in the previous two Sections has been shown to be an upper bound on the computation rate for a given SMD Petri net with a minimal integer consistent current assignment. We have also shown that this bound may not be achievable, and in this Section we look at the issue of achievability.

When the reader examines the expression for the bound, he will notice that it is always achievable for event graphs. Since the multiplicity of every transition in an event graph is one, this means that there exists a minimal integer consistent current assignment in which the current assigned to each transition is unity. Also, since a simple circuit in an event graph corresponds to a state machine component if we view the event graph as an SMD Petri net, we can rewrite Theorem 5.2.1 for event graphs to read as follows:

In a timed event graph $\mathcal{G} = (G,\Omega)$, the computation rate of all transitions in the net is the same and is given by

$$\rho \leq \min [\psi_1,\ldots\psi_m],$$

where $C_1,\ldots C_m$ are all simple circuits of the event graph and

$$\psi_k = \frac{\sum_{a_{ij} \in C_k} M(0, a_{ij})}{\sum_{\forall t_{k1} \in C_k} \tau_{k1}}$$

This is the same bound as the one we obtained in Expression [4.11] in Section 4.3. We also showed in Section 4.3 that this bound is achie-

vable. Thus, the result in Theorem 5.2.1 represents an achievable bound if the SMD Petri nets we consider are restricted to being event graphs. Our question is - how good a bound is it for more general SMD Petri nets? The largest class of SMD Petri nets we have been able to prove achievability for is the class we term α-SMD Petri nets.

Definition 5.3.1 An α-minimal integer consistent current assignment is one in which all transitions with more than one input place are assigned unit current.

Definition 5.3.2 An SMA Petri net is α-SMA iff there exists an α-minimal integer consistent current assignment for it.

Lemma 5.3.1 Let $\mathcal{C}$ be the c-equivalent net of an α-SMA Petri net $\mathcal{P}$ for the α-minimal integer consistent current assignment Φ. Then, every circuit C_k in $\mathcal{C}$ corresponds to a state machine component S_k in $\mathcal{P}$ and vice versa.

Proof $\Leftarrow$

Consider a state machine component S_k in $\mathcal{P}$. Let $\mathcal{F}$ be a cyclic frustrum for $\mathcal{C}$. We know that the initial slice of $\mathcal{F}$ must contain at least one place from every state machine component (follows from Theorem 3.5.1). Choose some place $p_{ki} \in S_k$ from the places in the initial slice of $\mathcal{F}$. When two consecutive instances of this place in a cyclic frustrum for the given current assignment are considered, every state machine containing p_{ki} must unfold into a chain that begins and ends at p_{ki}. In the c-equivalent net constructed from the cyclic frustrum, each chain corresponds to a circuit.

$\Rightarrow$. Consider the cyclic frustrum for the given c-equivalent net. Since $\mathcal{P}$ is SMA, every closed simple directed path is an SSM. Also, since every transition which has multiple input places has unit current, there is exactly one instance of each such transition in a cyclic frustrum. Every allocation reduction on the c-equivalent net results in exactly one SSM (a multiply-labelled circuit) for each SSM in $\mathcal{P}$.

Suppose some closed directed simple path does not correspond to any SSM. Then it must correspond to some closed structure in which there exists a transition t with multiple input places. The only way in which such a closed structure could map into a circuit is for there to be more than one instance of it in a cyclic frustum, which is the desired contradiction (see Figure 5.3.1).

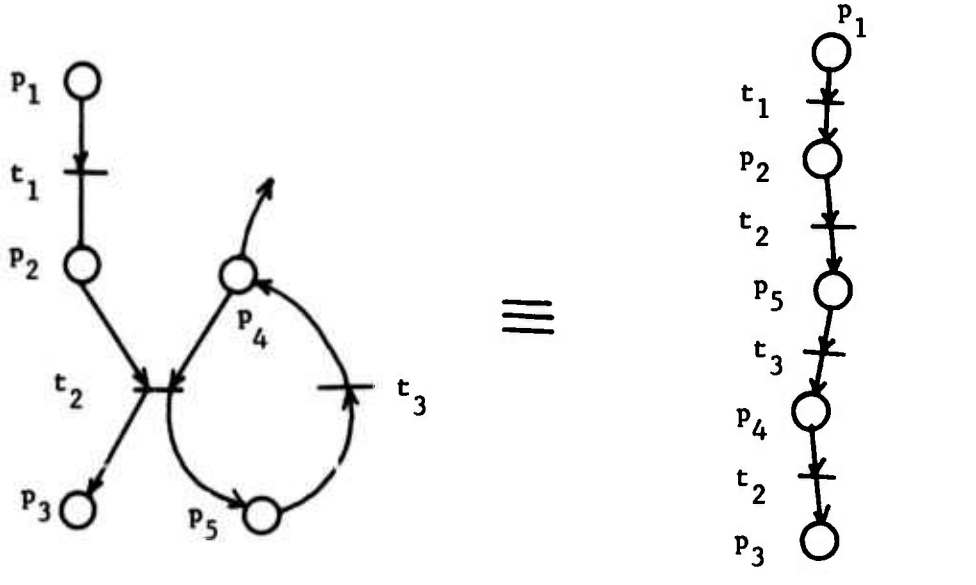


Figure 5.3.1

■

As a consequence of Lemma 5.3.1, we have the following theorem:

Theorem 5.3.1: The bound on the computation rate of transitions attained in Theorem 5.2.1 holds for every timed α - SMD Petri net $\chi = \langle \mathcal{P}, \Omega \rangle$

Proof: By Lemma 5.3.1, the c-equivalent net $\mathcal{C}$ for an α - SMD Petri net $\mathcal{P}$ is a multiply labelled event graph in which every simple circuit corresponds to a state machine component of $\mathcal{P}$ and vice versa. By attaching firing times to each transition in $\mathcal{P}$, the desired result follows.

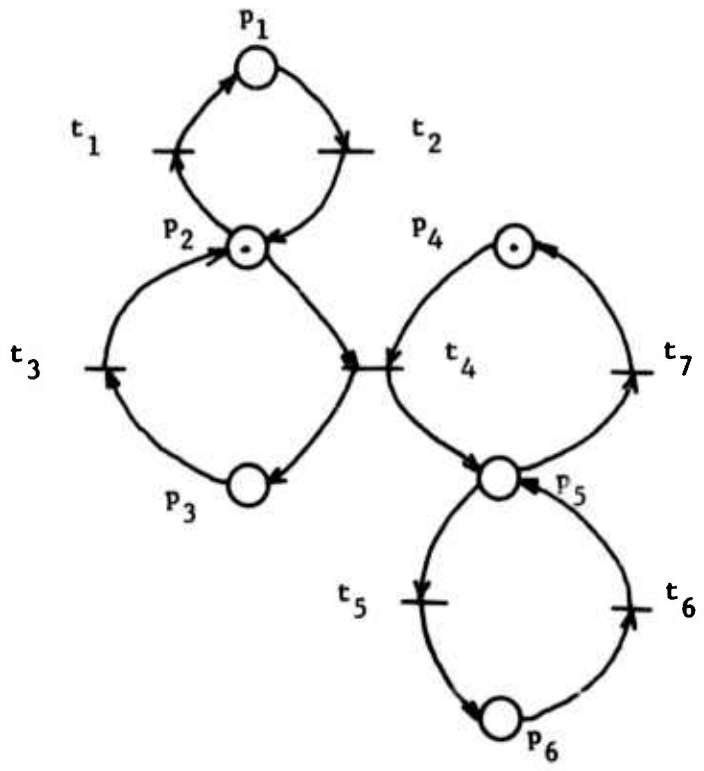
■

Example: Figure 5.3.2(a) gives an example of a timed α - SMA net, shown with a consistent current assignment. Figure 5.3.2(b) gives a c-equivalent net which realizes the computation rate given by the expression of Theorem 5.2.1. The reader will note that each state machine in the α - SMD Petri net of Figure 5.3.2(a) maps into a multiply labelled circuit in Figure 5.3.2(b) and vice versa.

In Figure 5.3.3(a) the timed SMD Petri net of Figure 5.3.2(a) is shown with a minimal integer consistent current assignment in which transition t_4 has a current equal to 2. This net is no longer α - SMA, and the bound of Theorem 5.2.1 can no longer be achieved, as can be verified by examining its c-equivalent net which has the largest maximum fundamental computation rate.

Bound from Theorem 5.2.1

$$\rho' = \min\left[\frac{1}{20}, \frac{1}{25}\right] = \frac{1}{25}$$



Current Assignment	Firing Time Assignment
$\varphi_1 = 2$	$\tau_1 = 3$
$\varphi_2 = 2$	$\tau_2 = 5$
$\varphi_3 = 1$	$\tau_3 = 4$
$\varphi_4 = 1$	$\tau_4 = 2$
$\varphi_5 = 4$	$\tau_5 = 7$
$\varphi_6 = 4$	$\tau_6 = 8$
$\varphi_7 = 1$	$\tau_7 = 3$

Figure 5.3.2(a)

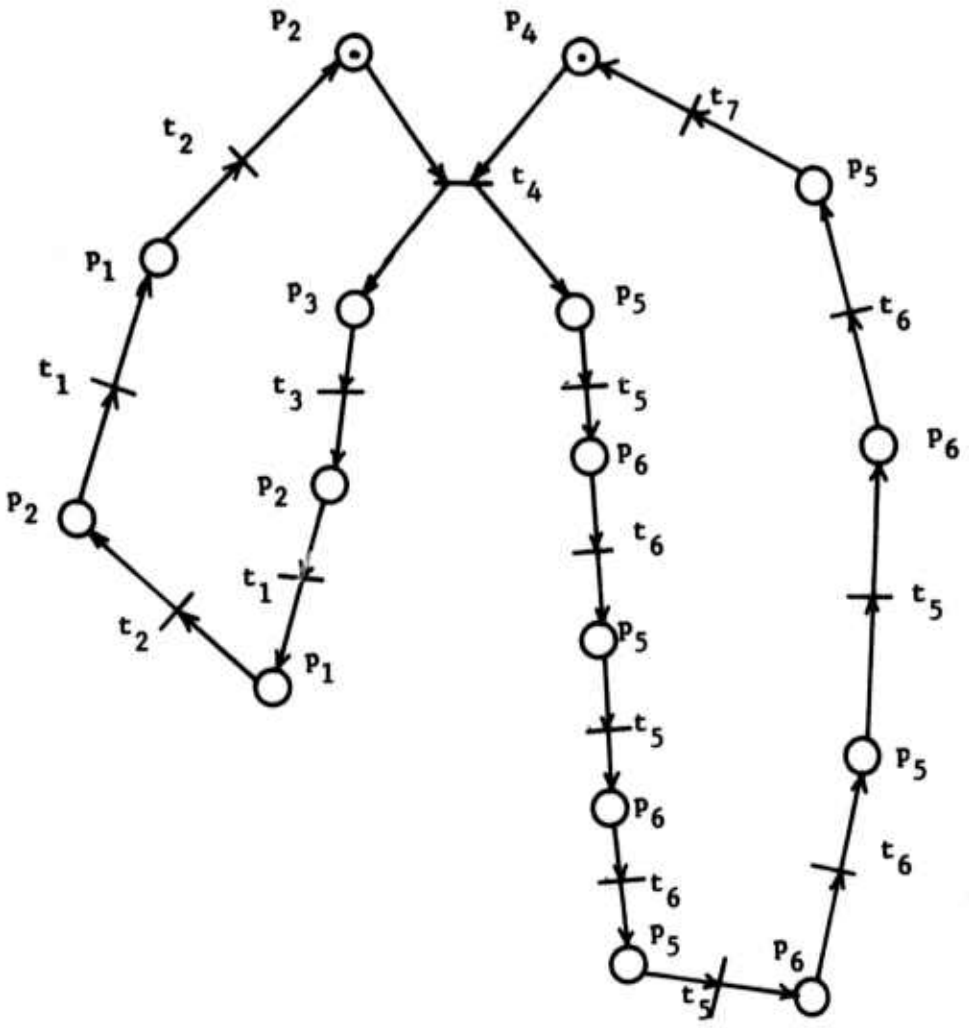

p2
p4
t7
t2
t4
p5
p1
p3
p5
t6
t1
t3
t5
p6
p6
p2
t6
t1
t5
t2
p5
p1
t5
p5
p6
t6
t6
p5
p6
t5

Figure 5.3.2(b)

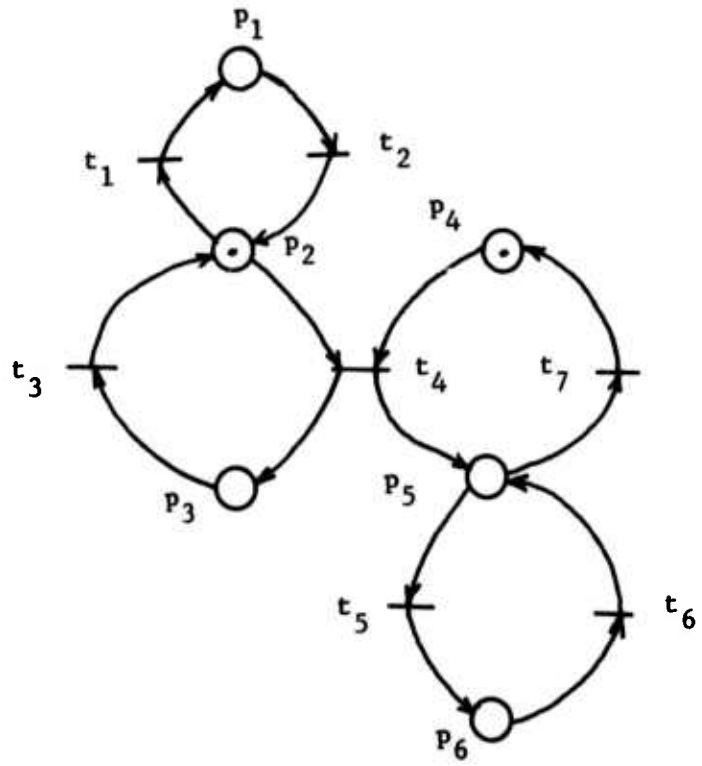


Figure 5.3.3(a).

Current Assignment	Firing Time Assignment
$\varphi_1 = 1$	$\tau_1 = 3$
$\varphi_2 = 1$	$\tau_2 = 5$
$\varphi_3 = 2$	$\tau_3 = 4$
$\varphi_4 = 2$	$\tau_4 = 2$
$\varphi_7 = 2$	$\tau_5 = 7$
$\varphi_5 = 1$	$\tau_6 = 8$
$\varphi_6 = 1$	$\tau_7 = 3$

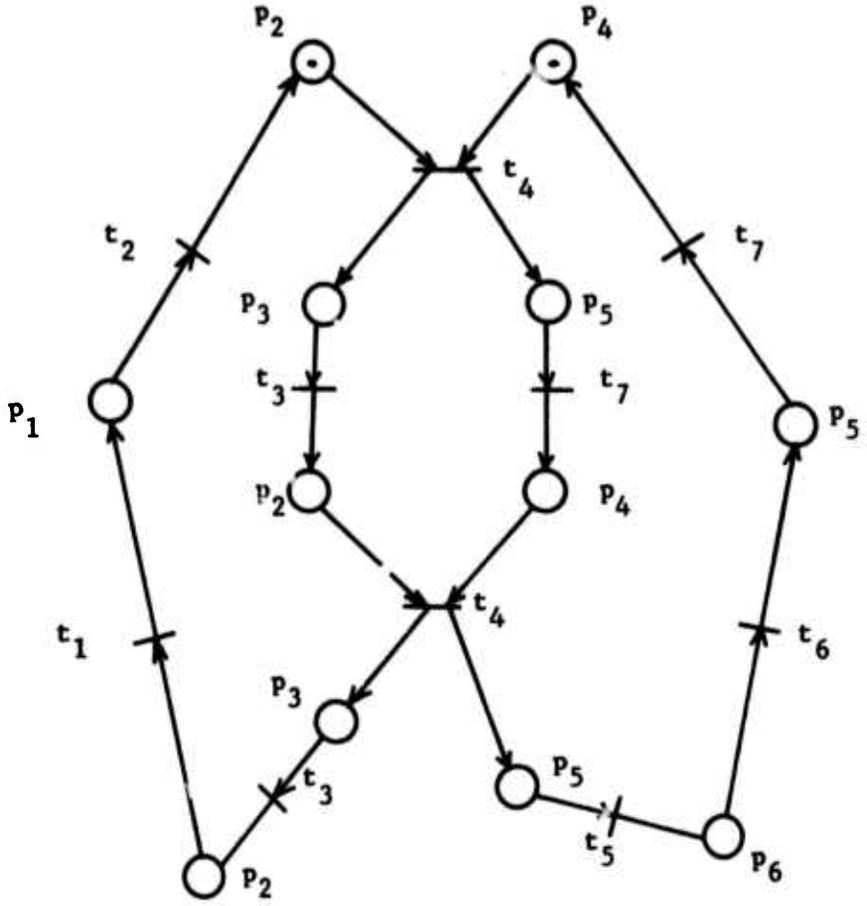


Figure 5.3.3.(b)

Bound for c-equivalent net of Figure 5.3.3(b)

$$\rho' = \min\left[\frac{1}{20}, \frac{1}{25}, \frac{1}{26}, \frac{1}{19}\right] = \frac{1}{26}.$$

Thus, the bound from Theorem 5.2.1 is not achievable by any c-equivalent net for the given timed SMD Petri net.

Section 5.4 Review of Results Obtained

In this Section we will put our work in perspective and will point out its relationship to PERT networks [F1] and the program flowcharts of Martin and Estrin [M1, M2, M3].

In Chapter 2 we have argued that practical systems can be viewed as an ensemble of interacting components and that each system component can be viewed as a state machine. Subsequently, we investigated the modelling of such systems using Petri nets. We introduced SMD Petri nets as being the class of Petri nets which can model practical systems. The type of systems we wish to model do not have any redundant functional operators, and are free of deadlock. This translates to the problem of choosing only those SMD Petri nets which have a live marking. We have pointed out that we still do not understand how the state machine components of an SMD Petri net should be interconnected to ensure that the net has a live marking. Currently, the largest subclass of SMD Petri nets we know of with this property is the class of SMA Petri nets.

Since any live marking for an SMD Petri net $\hat{P}$ is bounded, a consistent current assignment can be made to the transitions in $\hat{P}$. By multiplying all currents in a consistent current by the least common multiple of their denominators and dividing them by the greatest common divisor of their numerators, we get the minimal integer consistent current assignment. The current associated with a transition in a minimal integer consistent current assignment is the multiplicity of the transition in a c-equivalent net of the SMD Petri net for the given current assignment.

We have also looked at the entire class of LSP Petri Nets (i.e., Petri nets which have a Live, Safe, Persistent marking). Even though

the LSP Petri nets of interest are likely to be SMD, there do exist LSP Petri nets that are not SMD. A steady state equivalent net exists for any LSP Petri net. Furthermore, the steady state equivalent net of an LSP Petri net is unique, and corresponds to the c-equivalent net for an LB SMD Petri net with a minimal integer consistent current assignment.

Figures 5.4.1 and 5.4.2 are Venn diagrams which exhibit the relationship between the various subclasses of Petri nets that have been considered in this thesis.

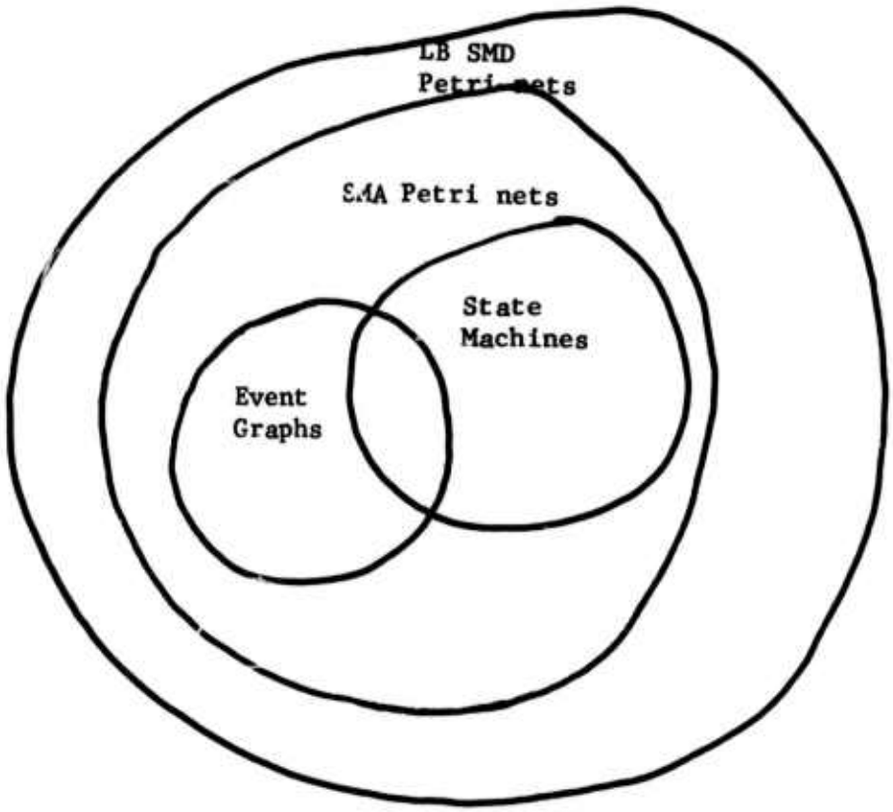
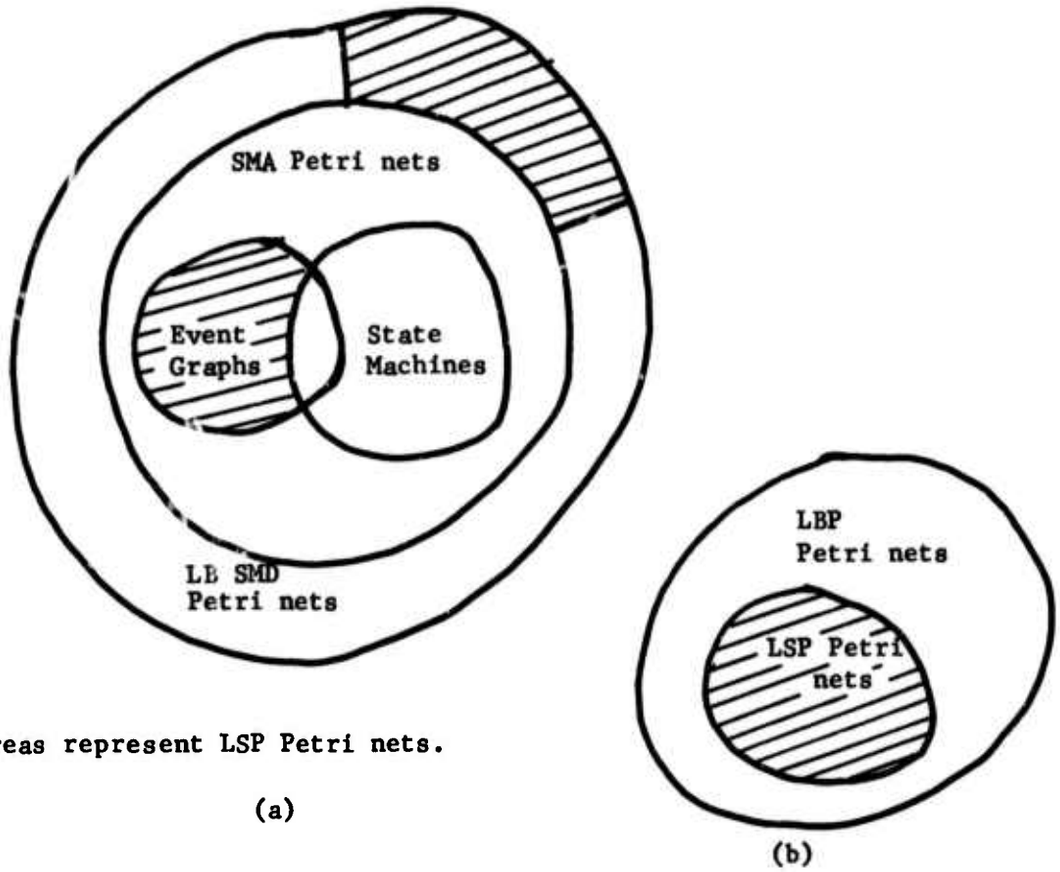


Figure 5.4.1



Shaded areas represent LSP Petri nets.

Figure 5.4.2

We have introduced timed Petri nets in order to model the finite speed of operation of practical systems and have shown how to find the computation rate of transitions in LB SMD Petri nets and LSP Petri nets. For both kinds of timed Petri nets, the maximum computation rate of a transition is the fundamental computation rate of the timed c-equivalent net for the timed Petri net multiplied by the multiplicity of the transition in the c-equivalent net.

In order to find the maximum fundamental computation rate of a timed c-equivalent net (or a steady-state equivalent net for an LSP Petri net),

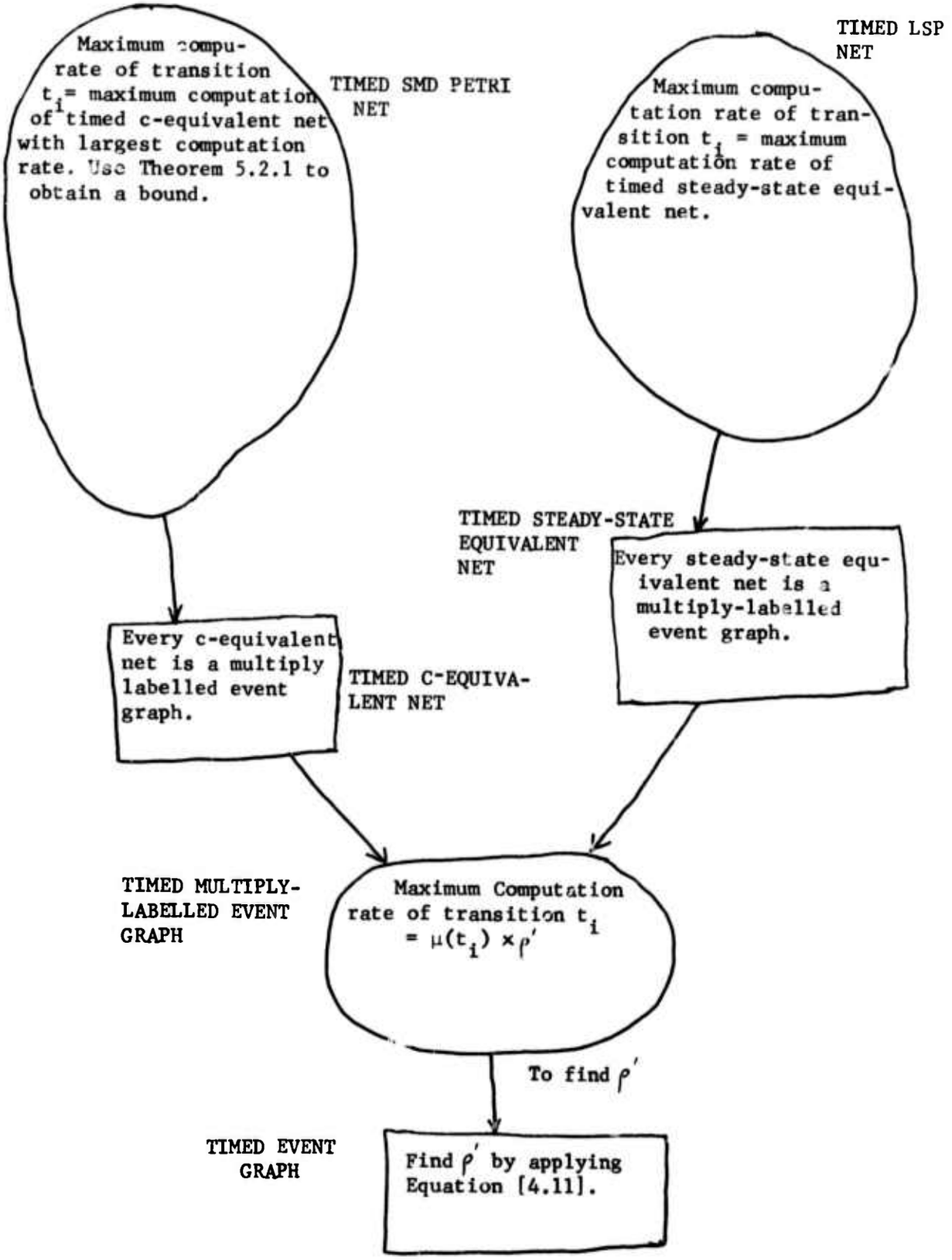

Maximum compu-
rate of transition
t_i = maximum computation
of timed c-equivalent net
with largest computation
rate. Use Theorem 5.2.1 to
obtain a bound.
TIMED SMD PETRI
NET
TIMED LSP
NET
Maximum compu-
tation rate of tran-
sition t_i = maximum
computation rate of
timed steady-state equi-
valent net.
TIMED STEADY-STATE
EQUIVALENT
NET
Every steady-state equi-
valent net is a
multiply-labelled
event graph.
Every c-equivalent
net is a multiply
labelled event
graph.
TIMED C-EQUIVA-
LENT NET
TIMED MULTIPLY-
LABELLED EVENT
GRAPH
Maximum Computation
rate of transition t_i
= $\mu(t_i) \times \rho'$
To find ρ'
TIMED EVENT
GRAPH
Find ρ' by applying
Equation [4.11].

Figure 5.4.3

we assume that all transitions are dictinctly labelled, and we find the maximum computation rate of the resulting timed event graph. The maximum computation rate of a timed event graph can be found by applying Equation [4.11].

We illustrate the various terms used and the relationship between the computation rates of timed event graphs, multiply-labelled event graphs, LSP Petri nets and SMD Petri nets by means of Figure 5.4.3. The diagram is self-explanatory and shows that the bound of Equation [4.11] forms the cornerstone of our work, the fundamental computation rate of timed SMD and LSP Petri nets being obtained by finding the fundamental computation rate of an equivalent multiply-labelled event graph.

We are now in a position to point out how our work relates to the following models of parallel processing and parallel computation:

(a) PERT networks

(b) Martin and Estrin Flowcharts.

Let us begin with PERT networks [F1]. A PERT network consists of an acyclic directed graph with an input vertex and an output vertex. All arcs in the network lie on paths from the input vertex to the output vertex. Each arc denotes an activity in the project being modelled by the network, and each activity takes a certain amount of time to occur. We can model a PERT network as a timed acyclic event graph by replacing each arc with its two end vertices by the structure shown in Figure 5.4.4. The time associated with the activity ab is now associated with the timed transition t_{ab} introduced into the arc ab. Transitions a and b are assumed to have zero firing time. In Figure 5.4.5 we show this transformation carried out on an example PERT network.

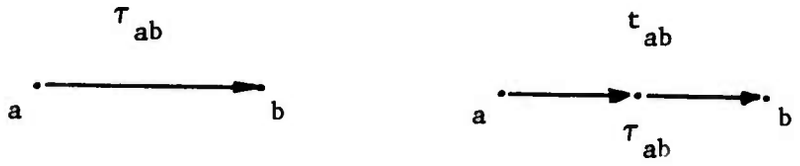
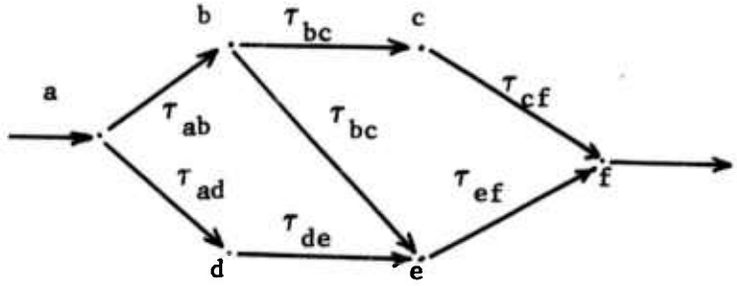
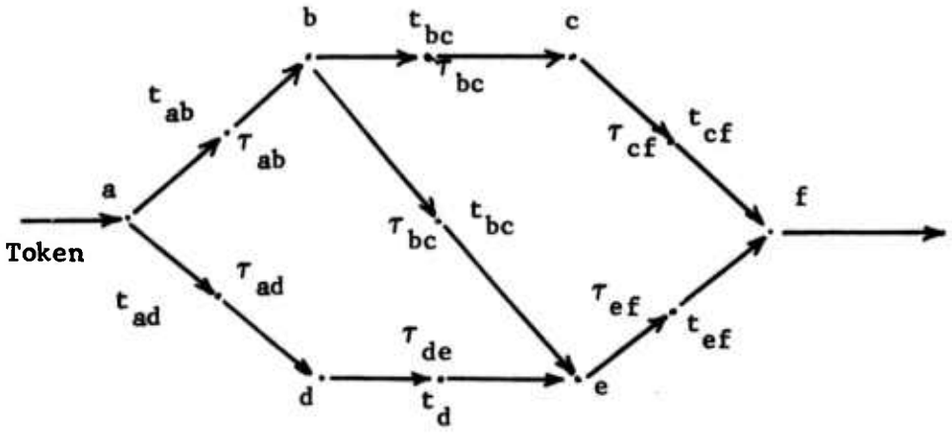


Figure 5.4.4.



(a) A PERT Network.



(b) Equivalent Timed Acyclic Event Graph.

Figure 5.4.5

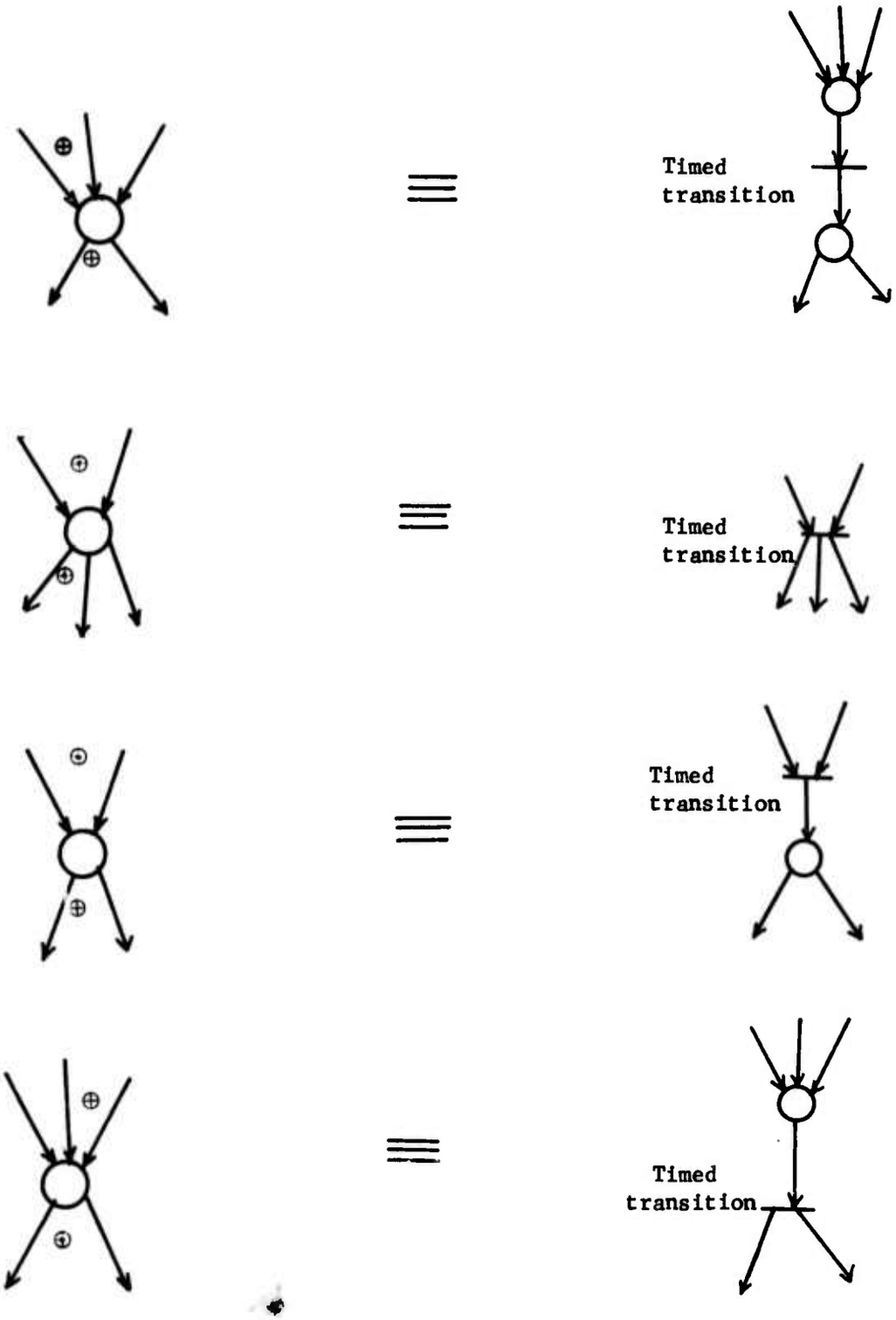

Timed
transition
Timed
transition
Timed
transition
Timed
transition

Figure 5.4.6.

Suppose we draw an arc from f to a in Figure 5.4.5. The resulting graph is a strongly-connected event graph, whose critical circuit corresponds to the critical path of the PERT network. In Chapter 6 we will see how strongly-connected timed event graphs can be applied to many practical situations where PERT networks are inadequate.

We now turn to the program flowcharts of Martin and Estrin [M1, M2, M3]. These are directed graphs consisting of arcs and nodes. Nodes represent operations in a computer program and arcs represent data paths between them. The presence of a data value on a data path is represented by a marker. Each node has logical conditions (and, exclusive-or) on data values on the input arcs that must be satisfied before the node can act. The action of a node consists of removing a data value from the specified logical combination of input arcs, performing a computation, and then depositing data values on the specified logical combination of output arcs. The action of a node is assumed to take some finite amount of time. In Figure 5.4.6 we show how to model these timed nodes by means timed acyclic Petri net structures. A Martin and Estrin flowchart models program constructs like decisions and iteration together with aspects of parallel programs, like the fork and join operations [D6]. A test is performed to check if the operation of the flowchart can proceed in a deadlock-free manner; if an arc is drawn from the output node to the input node, this test turns out to be similar to the one for deciding if an SMD Petri net is SMA. Our work can be viewed as modelling cyclic or recurrently acting systems, while the work of Martin and Estrin is concerned with finding the mean execution time of parallel programs. The two pieces of work taken should provide

a good bag of tools for the analysis and design of asynchronous computer systems.

CHAPTER 6

APPLICATIONS OF TIMED PETRI NETS TO THE MODELLING OF ASYNCHRONOUS CONCURRENT SYSTEMS

Deterministic Systems

We will now consider applications for our work drawn from diverse disciplines such as computer systems modelling and operations research. Throughout this discussion, we will keep in mind the distinction we have made between deterministic and non-deterministic systems. We begin by presenting several types of deterministic systems. The simplest system we wish to consider is a set of adder units whose action can be represented by the timed event graph shown in Figure 6.1.1.

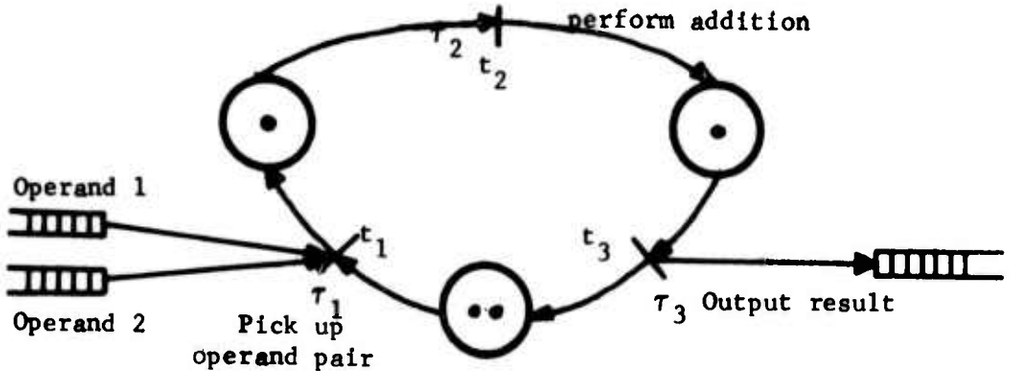


Figure 6.1.1

Each token on the circuit represents a physical hardware adder unit which can be in one of three states shown (i.e., "ready to pick up operand pair", "ready to perform addition" and "ready to output result"). We will make the assumption that there is always an operand pair available in the

input queues, so that a hardware adder unit never has to be idle for want of operands, i.e., the environment does not introduce delays into the operation of the system. Applying Equation [4.11], the maximum rate at which results appear in the output queue is given by $n / (\tau_1 + \tau_2 + \tau_3)$ where n is the number of physical hardware adder units,

τ_1, τ_2 and τ_3 are the times required to perform the actions modelled by transitions t_1, t_2 and t_3.

Let us assume some values for the above parameters.

Let $n = 4$.

$\tau_1 =$ 500 nsec.

$\tau_2 =$ 2 μsec.

$\tau_3 =$ 500 nsec.

The maximum throughput rate of the adder is then 4/3 = 1.33 million additions per second.

A more interesting example is the timed Petri Net model of a three-stage pipelined floating point adder shown in Figure 6.1.2.

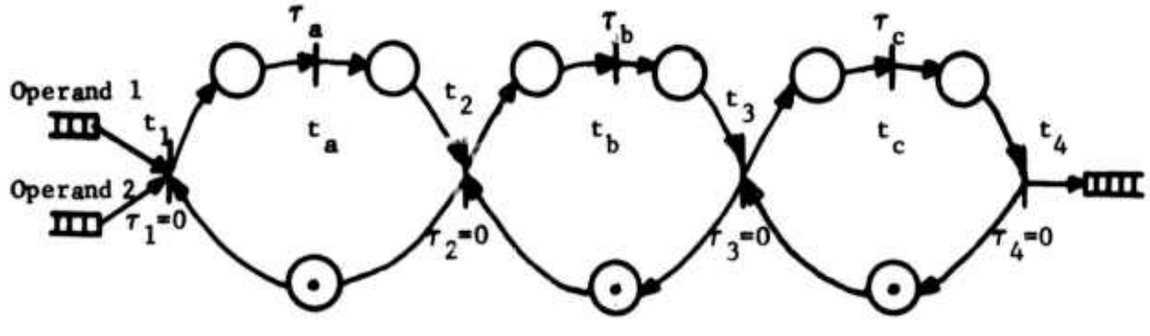


Figure 6.1.2

This pipelined adder has only one hardware unit per stage, so that there is overlapped operation of the stages without any parallelism in each stage. The maximum throughput rate of the system is seen to be

$$\min [\ 1/\tau_a\ ,\ 1/\tau_b,\ 1/\tau_c\ ].$$

Let us examine the above expression. Suppose $1/\tau_a$ is the smallest of the three quantities in parenthesis. Then stage A has the slowest hardware unit, and the natural computation rate of this stage determines the throughput rate of the system. This means that having hardware units in stages B and C that are faster than the hardware unit of stage A is wasteful, since their added speed does not result in any extra system throughput. We will say that a deterministic system is balanced if the natural computation rate of all system parts is equal. In the context of the pipeline adder in Figure 6.1.2. this means that $\tau_a = \tau_b = \tau_c$.

A more complex pipelined adder would be one in which there are multiple hardware units in each stage. Such a pipelined adder is shown in Figure 6.1.3.

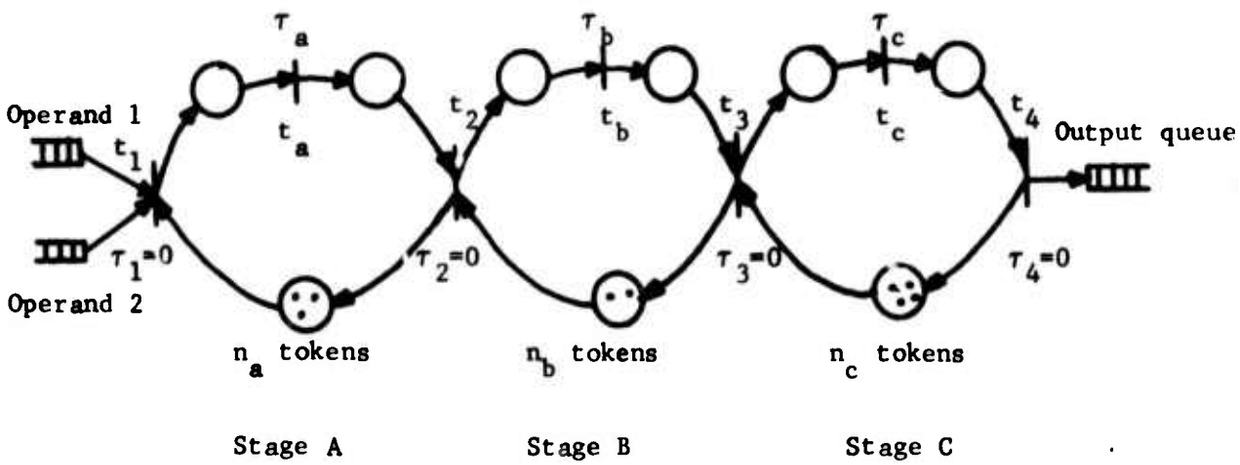


Figure 6.1.3

Stages A,B and C have n_a, n_b and n_c hardware functional units, respectively. In this case, the throughput of the system is given by

$$\min [\, n_a/\tau_a,\ n_b/\tau_b,\ n_c/\tau_c \,].$$

For the system to be balanced, we see that $n_a/\tau_a = n_b/\tau_b = n_c/\tau_c$. This implies the following:

If a stage consists of slow hardware functional units, a proportionately larger number of units should be present in the stage to ensure that the system is balanced. Absence of system balance implies that there are parts of the system with excess capacity that cannot be used. Our results can thus be used to test for system balance in deterministic systems, and they serve to formalize the intuitive notions of balance that a hardware designer would, no doubt, use in the design of pipelined systems.

PERT Networks and Project Scheduling

PERT charts are used in Project scheduling to determine the shortest time that it takes for an ensemble of concurrent activities to complete, given the precedence relationships between the activities and the time duration of each activity. We examined the relationship between PERT charts and timed Event Graphs in Section 5.6. We now see that by using timed event graphs, we can model aspects of project scheduling and assembly lines that are not within the power of PERT charts. The two main advantages of timed event graphs over PERT charts is their ability to:

(a) model systems that act recurrently.

(b) model physical resource units explicitly.

Suppose we consider the project represented by the PERT chart in Figure 6.1.4. The project consists of activities "a" through "g" with the precedence constraints expressed by the PERT chart. We begin by drawing the equivalent acyclic event graph for this PERT chart , using the method given

in Section 5.6. This acyclic event graph is shown in Figure 6.1.5.

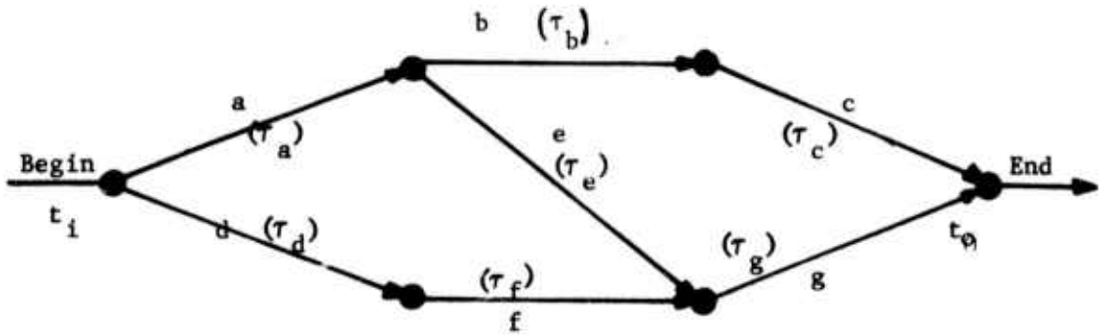


Figure 6.1.4 A PERT Chart.

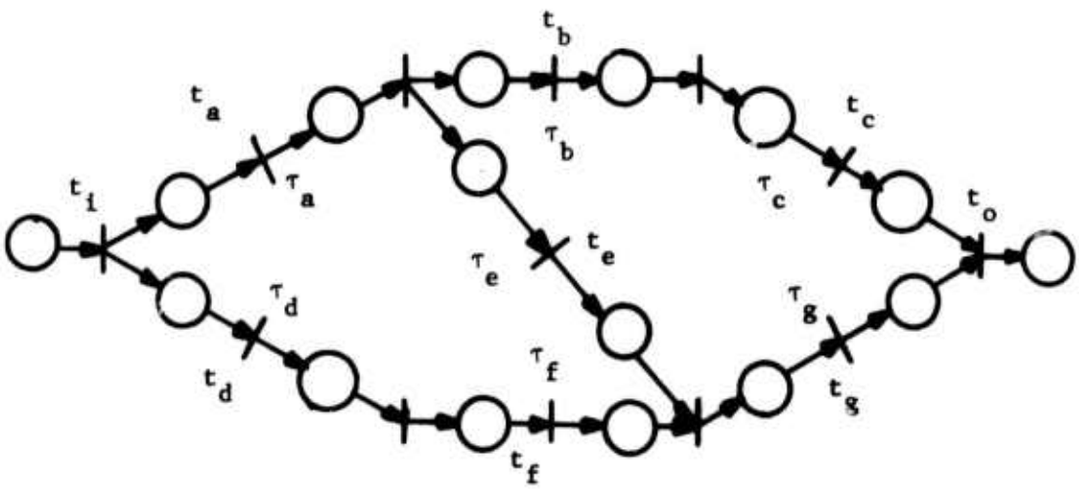


Figure 6.1.5 Equivalent Event Graph.

Now suppose we want the PERT chart in Figure 6.1.4 to represent a manufacturing process, where t_i represents the start of the process, and t_o the end. We assume that each activity in the system requires the use of a unit of resource, like a lathe, a milling machine, etc. We wish to express

the fact that there are only a finite number of units of each resource type. Also, some resource units may be very expensive and may have to be shared among several activities (e.g. a high precision lathe). How are we to express such system constraints using a PERT chart? The answer is that there is no way of doing this without augmenting the structure of PERT charts. Let us see what added descriptive power can be had by using timed event graphs. Around each of the transitions t_a through t_g, we draw a loop. Each loop is marked with a number of tokens equal to the number of physical processing units available for the corresponding activity. The resulting event graph is shown in Figure 6.1.6.

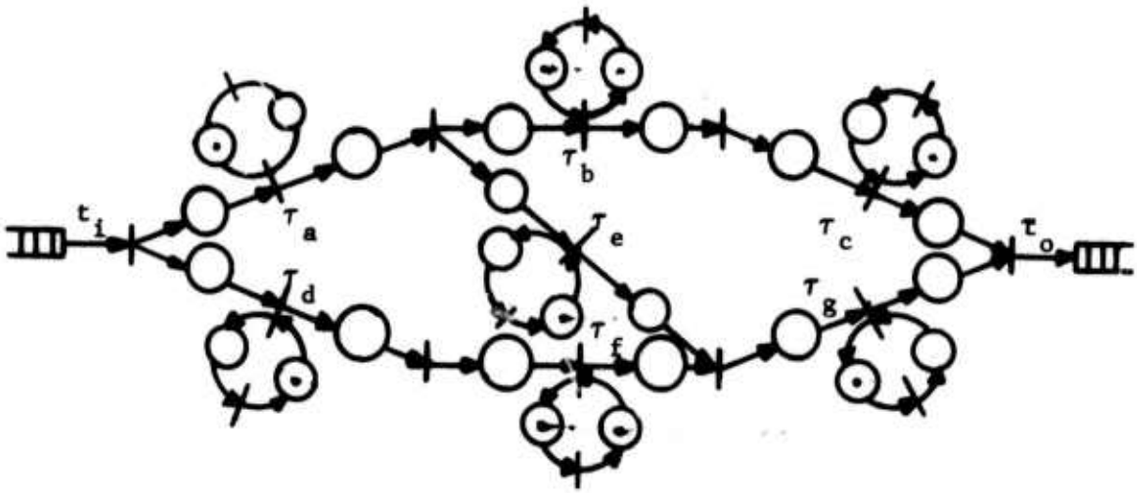


Figure 6.1.6 Event graph which models limited resources available for each activity in PERT chart.

In addition to the transition representing an activity, each loop consists of a second transition, which we will interpret as the act of allocating a resource unit to the activity. Since we wish to model a recurrently acting production facility, we complete the loop between t_o and t_i and add a large number of tokens to place p as shown in Figure 6.1.7. One of the self-loops

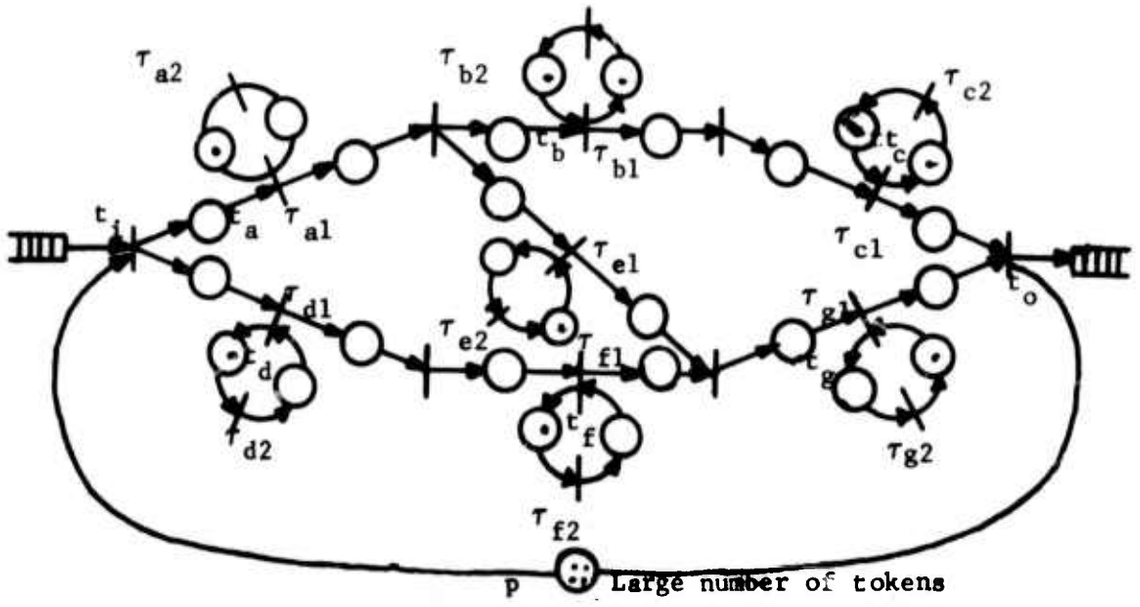


Figure 6.1.7

(i.e., loops around each of the transitions $t_a \ldots t_g$ now becomes the critical circuit. The system can be balanced by assuming that the natural computation rate of each of these circuits is the same, i.e.

$$n_a/(\tau_{a1}+\tau_{a2}) = n_b/(\tau_{b1}+\tau_{b2}) = n_g/(\tau_{g1}+\tau_{g2})$$

The n's denote the number of tokens on the corresponding circuits. If all the n's are infinite, then the processing capacity of the production facility becomes infinite. This is the situation represented by Figure 6.1.5. Since any production facility in real life has only finite resources available to it, these resources can be explicitly represented as we did in Figure 6.1.6.

A word now about adding an infinite number of tokens to the loop formed by joining t_o to t_i through place p. Each token represents the possibility

of a set of subassemblies being input to the production facility for processing. Our assumption that we can add an infinite number of tokens to place p is equivalent to saying that there is an unbounded amount of buffer capacity between the activities $t_a \ldots t_g$. In practice the amount of buffer space (in the form of storage bays) between work stations in a production process is bounded, and this would place a limit on the number of tokens we can put on place p. In that case, we would have to consider the natural computation rate of all circuits in Figure 6.1.7 including circuits like

$t_i t_a t_b t_c t_o t_i$, $t_i t_d t_f t_g t_o t_i$, etc.

Let us give an example of how the buffer capacity of the system may affect the maximum throughput rate of the system. We do this by considering some actual values for system parameters.

Let $\tau_{a2} = \ldots = \tau_{g2} = 0.$

τ_{a1}	= 3	n_a	= 2.
τ_{b1}	= 2	n_b	= 1.
τ_{c1}	= 4	n_c	= 2.
τ_{d1}	= 6	n_d	= 3.
τ_{e1}	= 5	n_e	= 2.
τ_{f1}	= 8	n_f	= 3.
τ_{g1}	= 3	n_g	= 2.

Case (i) We assume that M(p) (i.e., the number of tokens on place p) = 6.

The throughput rate of the system = min[2/3 1/2, 2/4, 3/6, 2/5, 3/8, 2/3, 6/9, 6/11, 6/17] = 6/17.

We see that circuit $t_i t_d t_f t_g t_o t_i$ limits the throughput rate of the system.

Case (ii) Now let M(p) = 100 (i.e.,"very large").

The throughput rate of the system becomes min [2/3, 1/2, 2/4, 3/6, 2/5, 3/8, 2/3, 100/9, 100/11, 100/17] = 3/8.

The loop around transition t_f now becomes the bottleneck in the system.

The production facility we have considered so far was arranged in such a way that each activity has its own set of resource units available to it. We now consider the issue of resource sharing.

Resource Sharing

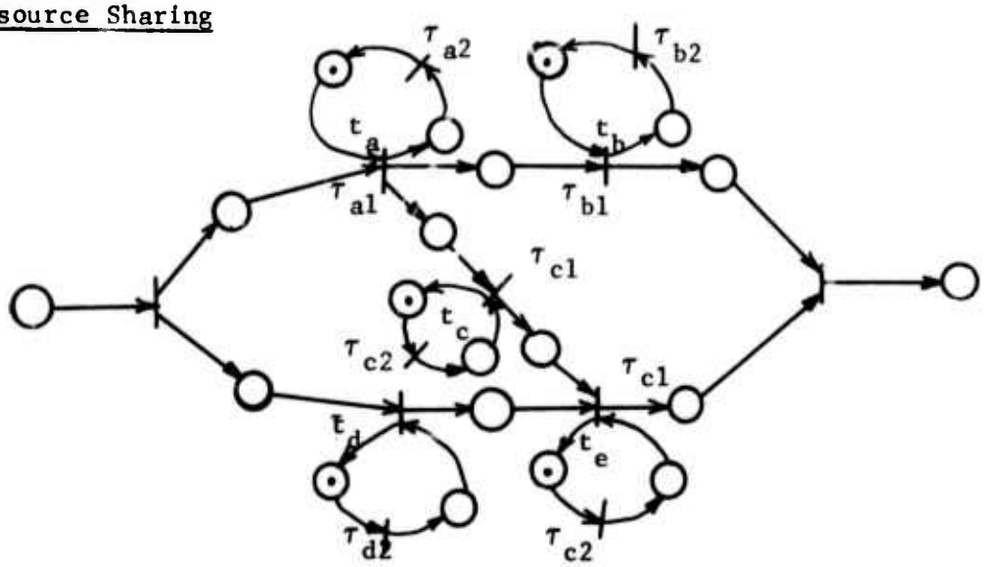


Figure 6.1.8

Consider the production process shown in Figure 6.1.8. Once again, letting $n_a, \ldots n_e$ denote the number of tokens on the loops around transitions $t_a \ldots t_e$ respectively, we assume the following values for the parameters of the system:

$\tau_{a2} = \tau_{b2} = \ldots = \tau_{e2} = 0.$

$\tau_{a1} = 6 \qquad n_a =, 1.$

$\tau_{b1} = 2 \qquad n_b = 1.$

$\tau_{c1} = 5 \qquad n_c = 1.$

$\tau_{d1} = 10 \qquad n_d = 2.$

$\tau_{e1} = 4.5 \qquad n_e = 1.$

We make the assumption that the same type of resource is used by both activities t_a and t_b. In Figure 6.1.8 , each activity has its own resource unit. In this configuration, the system throughput is

$$\min [\ 1/6,\ 1/2,\ 1/5,\ 2/10,\ \ 1/4.5\ ] \ = \ 1/6.$$

We see that activity t_a is the bottleneck in the system. Since activities t_a and t_b use the same type of resource, it is possible to pool their resource units together . Now whenever resources are pooled together, some resource allocation strategy must be adopted to ensure fair resource allocation to the contending resource users. Since event graphs can model only deterministic systems, we will use a simple strategy in which each resource unit is allocated alternately to the two activities. The resulting system is shown in Figure 6.1.9.

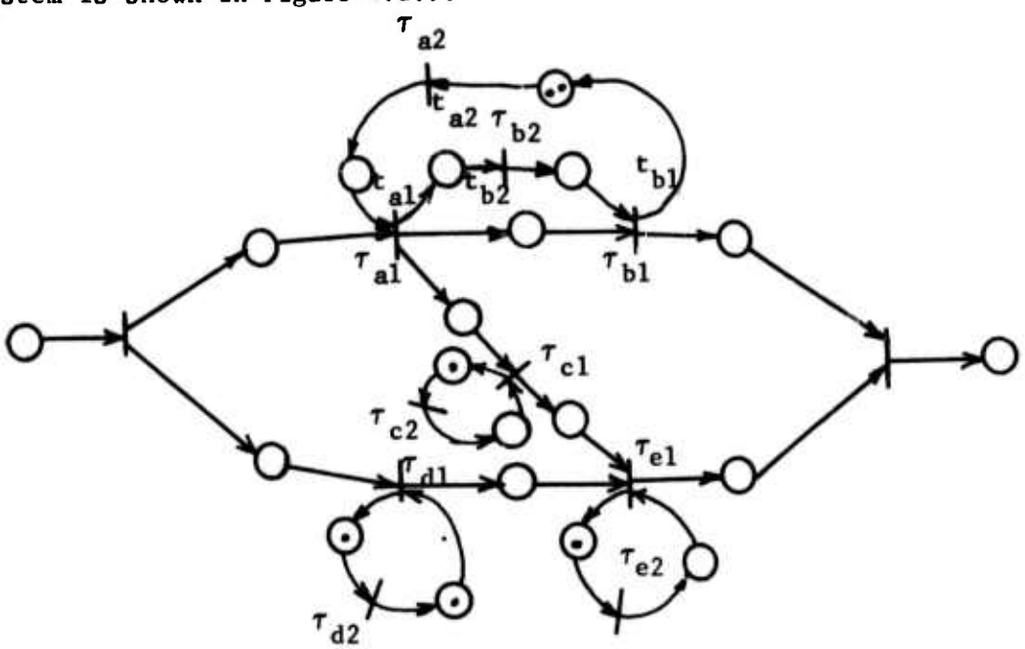


Figure 6.1.9

We assume that all parameters of the system are the same as before; this time the throughput of the system is

$$\min [\ \ 2/8\ \ 1/5,\ 2/10,\ 1/4.5\ ] \qquad = \quad 1/5.$$

Thus, the activity t_a is no longer the bottleneck in the system. It should be noted that other resource allocation strategies are possible, but they result in non-deterministic nets.

So far, the system models we have considered have all been timed event graphs. We now give an example of a system which is modelled by an LSP Petri net.

Figure 6.1.10 is a timed LSP Petri net model of the assembly line that alternately outputs two types of automobiles, which we discussed in Section 1.2.

The maximum computation rates of transitions t_5 and t_6 are of interest because they tell us the maximum rate at which automobiles are output by this assembly line. These computation rates can be found from the timed steady state equivalent net of Figure 6.1.10(b).

We now suppose that there is parallelism within some of the assembly stages, but that automobiles are still manufactured alternately. This can be modelled by adding tokens to places in the net of Figure 6.1.10(a) other than p_8 and p_9, and we get an LBP Petri net of the type shown in Figure 6.1.10(c).

The reader will recall from Section 3.4 that in order to draw the steady state equivalent net of an LBP net that is also LSP, tokens are removed until a life, safe marking results and then drawing the steady state equivalent net for it. The marking is constructed by the technique given in Section 3.4. We get the steady state equivalent net of Figure 6.1.10(d). Once again, the maximum computation rate of transitions k_5 and k_6 can be found.

The system models we have considered so far have all been determinis-

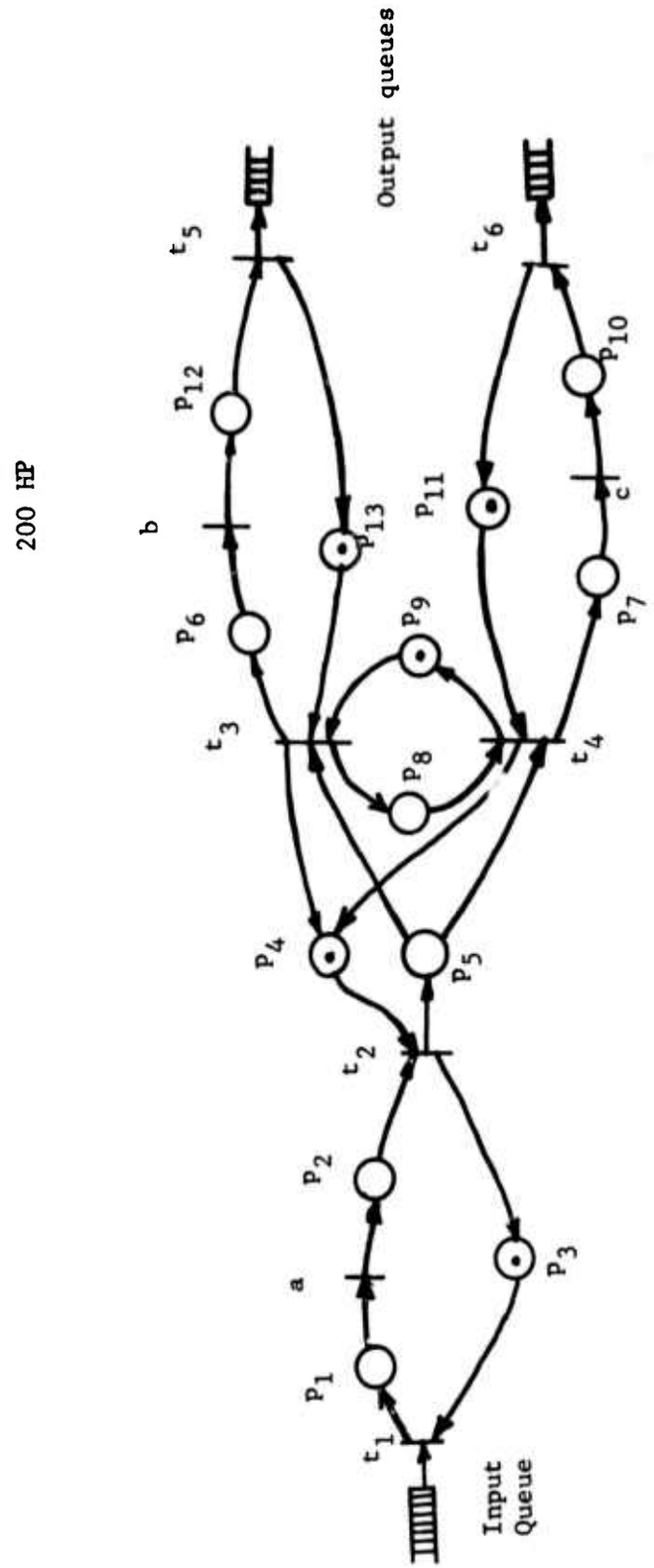


Figure 6.1.10(a)

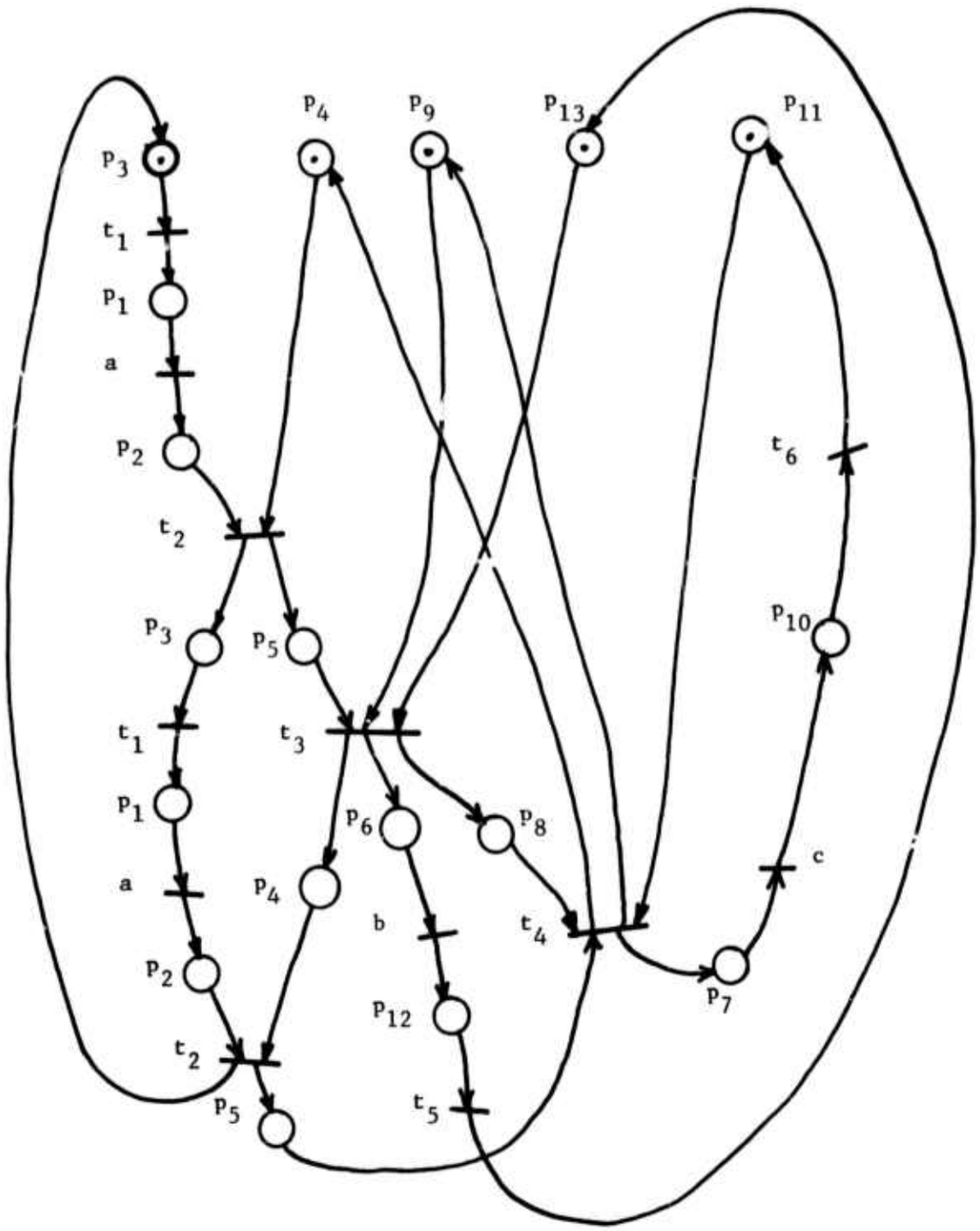


Figure 6.1.10(b) Steady State Equivalent Net

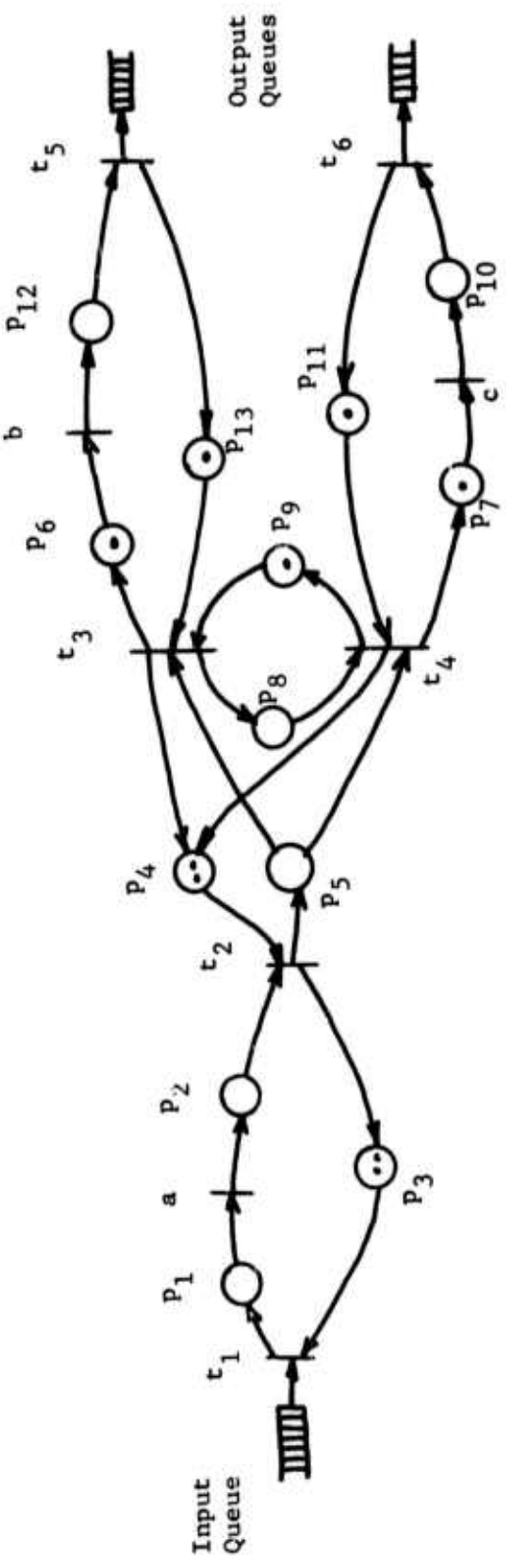


Figure 6.1.10(c)

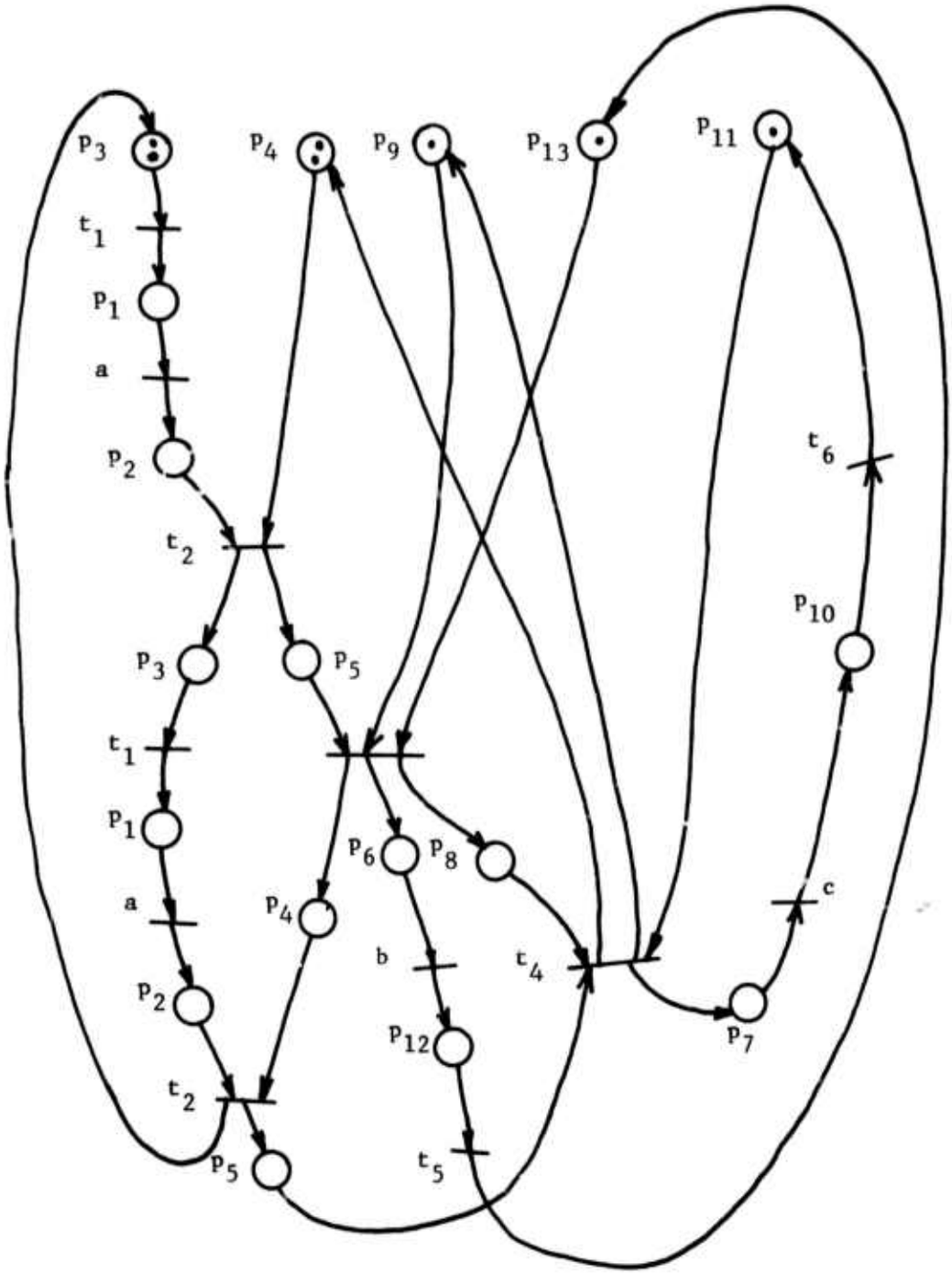
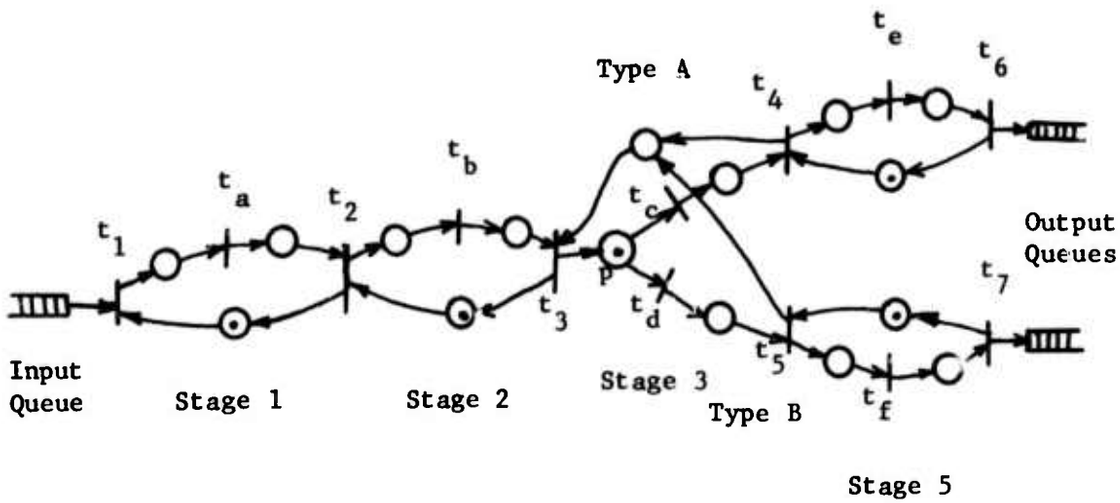


Figure 6.1.10(d). Steady State Equivalent Net of the LBP Petri Net of Figure 6.1.10(c)

tic Petri nets. We now turn our attention to the modelling of non-deterministic systems.

The simplest example of a non-deterministic system that we wish to model is the pipelined processor with decisions of the type discussed in Section 1.2. A timed SMD Petri net model for this is given in Figure 6.1.11.



$$\tau_1 = \tau_2 = \tau_3 = \tau_4 = \tau_5 = \tau_6 = \tau_7 = 0$$

Figure 6.1.11

This net has five state machine components, as the reader can easily verify (indicated as Stage 1,...,Stage 5). Two types of instructions can be processed, Type A and Type B. Let us suppose that the relative frequency of these two instruction types is in the ratio of 2:3 (this can be found by statistical analysis of program traces). This leads to

the following minimal integer consistent current assignment:

$$\varphi_1 = \varphi_a = \varphi_2 = \varphi_b = \varphi_3 = 5$$
$$\varphi_c = \varphi_4 = \varphi_e = \varphi_b = 2$$
$$\varphi_d = \varphi_5 = \varphi_f = \varphi_7 = 3.$$

The computation rate ρ_1 of transition t_1 gives the maximum rate at which instructions are absorbed from the input queue. Similarly, the computation rates ρ_6 and ρ_7 give the maximum rates at which results are placed in the output queues. To calculate these quantities, we first find ρ'. From Theorem 5.2.1,

$$\rho' = \min [\psi_1, \psi_2, \psi_3, \psi_4, \psi_5]$$

where each of the terms represents the fundamental computation rate of the corresponding stage in the system. System balance requires that

$$\psi_1 = \psi_2 = \psi_3 = \psi_4 = \psi_5.$$

Let τ_i be the firing time of transition t_i where i = a,...,f. Then, we have

$$\rho' = \min\left[\frac{1}{\varphi_a \tau_a}, \text{ etc....}\right]$$

$$= \min \frac{1}{5\tau_a}, \frac{1}{5\tau_b}, \frac{1}{2\tau_c + 3\tau_d}, \frac{1}{2\tau_a}, \frac{1}{3\tau_f}$$

Suppose we choose $\tau_a = \tau_b = \tau_c = \tau_d = \tau_e = \tau_f = 1$ μsec., then $\rho' = 1/5$ and the system is not balanced.

A balanced and more economical system results when $\tau_e = 2.5$ μ sec., $\tau_f = 1.67$ μ sec., the other parameters being the same. Under this new firing time assignment,

$\rho' = \frac{1}{5}$ $= 2 \times 10^5$ instructions/sec.

therefore,

$\rho_1 = 5 \times \rho'$ $= 10^6$ instructions/sec.

$\rho_6 = 2 \times 2 \times 10^5$ $= 4 \times 10^5$ instructions/sec.

$\rho_7 = 3 \times 2 \times 10^5$ $= 6 \times 10^5$ instructions/sec.

The reader can construct further examples based on processors in which there are multiple hardware units within each stage, and can work out a method for an optimal design based on certain objective and cost functions. We shall not attempt to do this here.

The SMD Petri nets we have exhibited so far have mainly been models of pipelined processors or assembly processes. Let us now look at models for interacting cyclic processes in computer systems. We begin with a model for two processes that interact with each other through mailboxes. Figure 6.1.12 gives a schematic or a flowchart model for such a pair of processes. One of the processes is deterministic (or decisionless) and the other one has two decisions in it. The processes communicate by passing messages to each other through mailboxes, and the reader can convince himself that they can operate concurrently without deadlocking. This can be verified formally when the Petri net model for this system shown in Figure 6.1.13 is examined. The net is seen to be SMA, and in Figure 6.1.14 we indicate its state machine components. Since the marking shown puts at least one (in this case exactly one) token on each state machine, the marking is clearly live. This, as the reader will recall from Chapters 2 and 3, means that the system of processes has no redundant operators, and furthermore, that the processes can operate without being deadlocked.

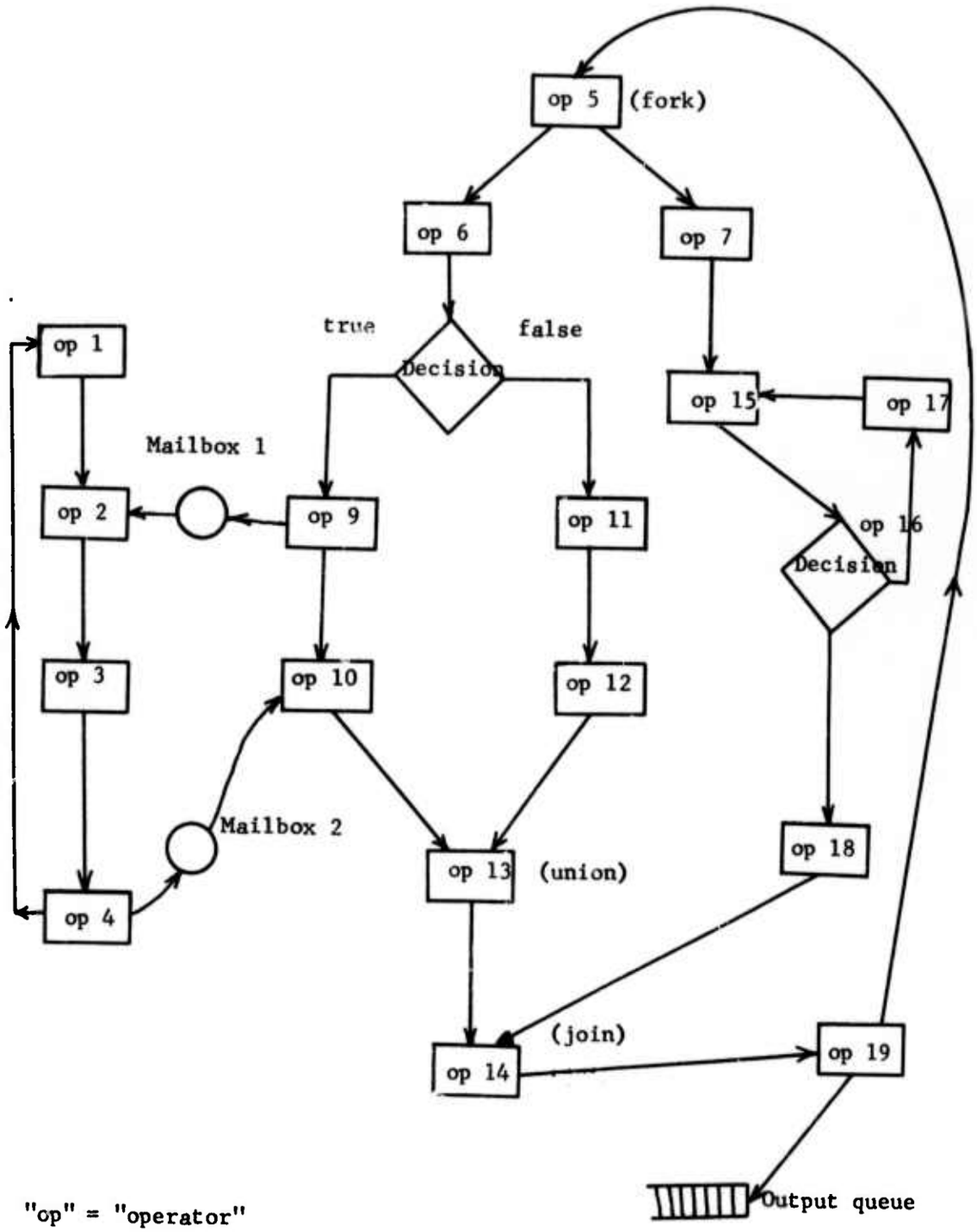

op 5
(fork)
op 6
op 7
true
false
op 1
Decision
op 15
op 17
Mailbox 1
op 2
op 9
op 11
op 16
Decision
op 3
op 10
op 12
Mailbox 2
op 18
op 13
(union)
op 4
(join)
op 14
op 19
Output queue
"op" = "operator"

Figure 6.1.12

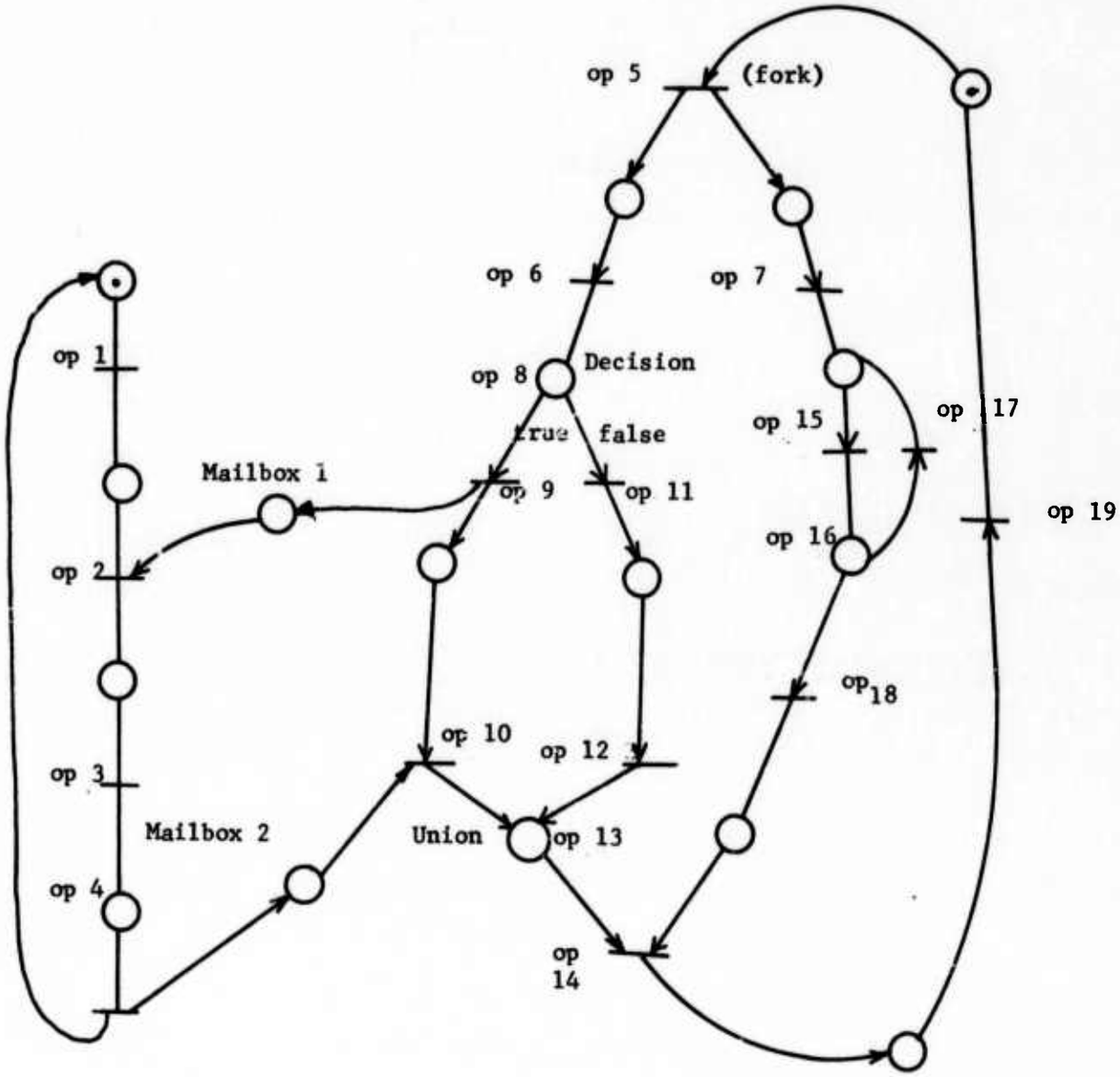
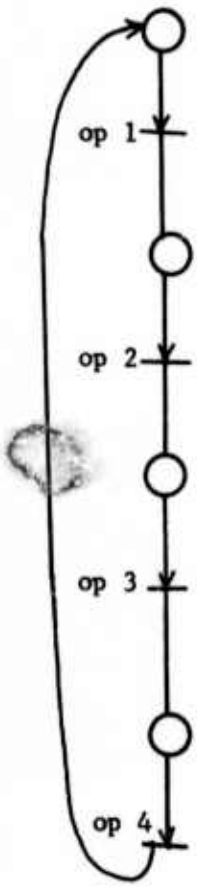
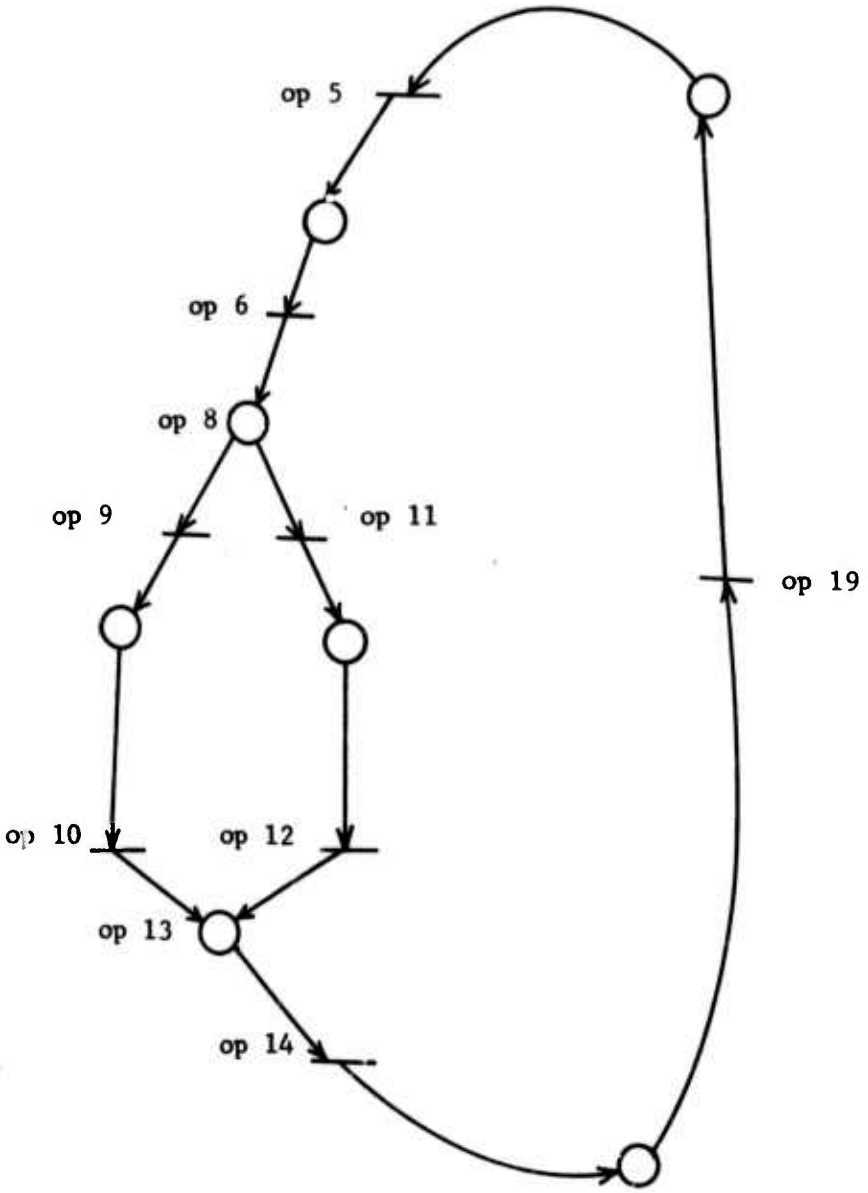

op 5
(fork)
op 6
op 7
op 1
op 8
Decision
op 15
op 17
true
false
Mailbox 1
op 9
op 11
op 19
op 16
op 2
op 18
op 10
op 12
op 3
Mailbox 2
Union
op 13
op 4
op
14

Figure 6.1.13.



S_1

Figure 6.1.14(a)



S_2

Figure 6.1.14(b)

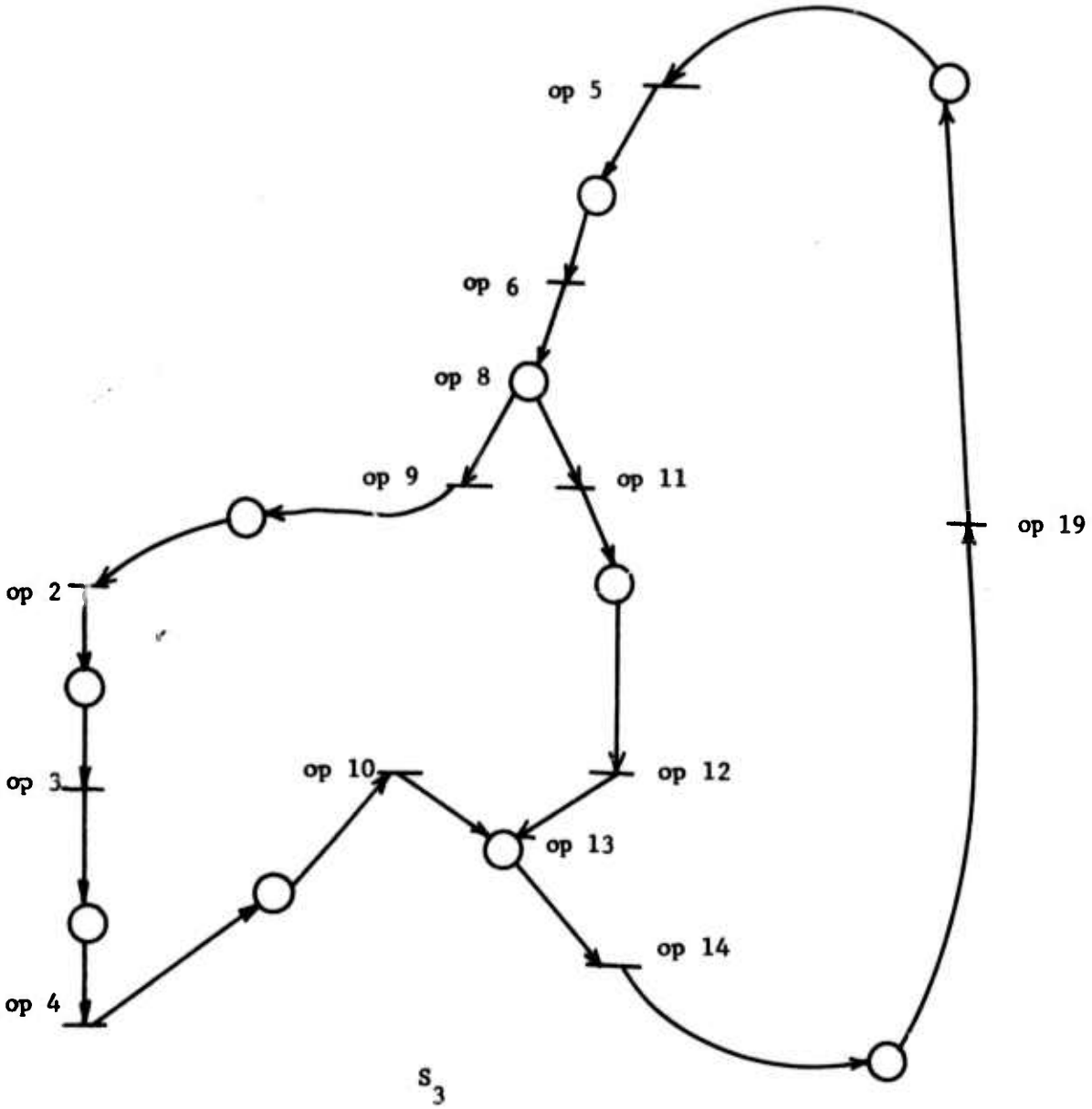


Figure 6.1.14(c)

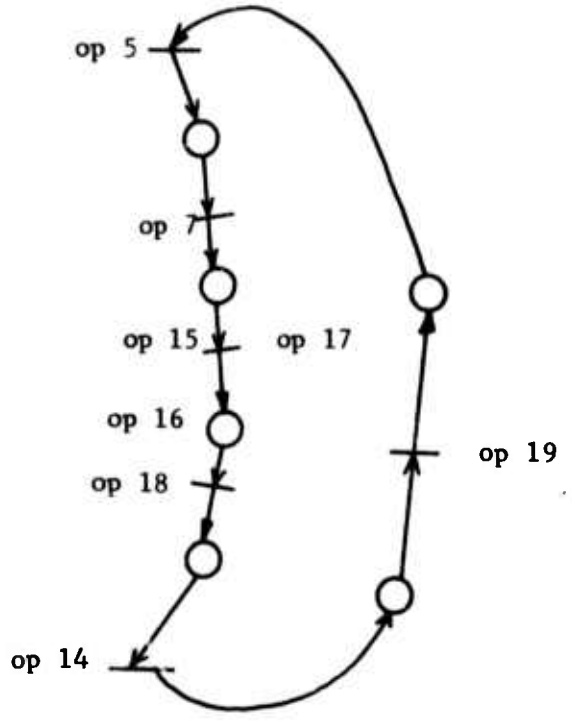


Figure 6.1.14(d)

Having thus established that the Petri net in Figure 6.1.13 has a live marking, we can now suppose that the transition labelled "op 19" represents an operation which outputs a result every time the transition fires. Our problem now is to find out the maximum rate at which op 19 outputs results, given the usual parameters like the firing times of the transitions and a minimal integer consistent current assignment. The places marked "op 8" and "op 16" represent decisions, each decision having two outcomes. The relative frequency of the outcomes of each decision can be found by statistical means. Let the probabilities of op 11 and op 9 be p_{11} and p_9 respectively, where $p_{11} + p_9 = 1$.

Similarly, let p_{18} and p_{17} be the probabilities of op 18 and op 17 respectively, such that $p_{17} + p_{18} = 1$. We can now obtain a minimal

integer consistent current assignment for the net, where each current is denoted as usual by φ, and

$$\frac{\varphi_{11}}{\varphi_9} = \frac{p_{11}}{p_9} \qquad [6.1]$$

$$\frac{\varphi_{18}}{\varphi_{17}} = \frac{p_{18}}{p_{17}} \qquad [6.2]$$

As an example, let $p_{11} = 1/3 \quad p_9 = 2/3$

$p_{18} = 3/4 \quad p_{17} = 1/4$

Then, the following is a consistent current assignment:

$\varphi_5 = 1 \quad \varphi_6 = 1 \quad \varphi_7 = 1 \quad \varphi_{14} = 1 \quad \varphi_{19} = 1$

$\varphi_9 = \varphi_{10} = \varphi_1 = \varphi_2 = \varphi_3 = \varphi_4 = \frac{2}{3}$.

$\varphi_{11} = \varphi_{12} = \frac{1}{3}$.

$\varphi_7 = 1 \quad \varphi_{15} = 1\frac{1}{4} \quad \varphi_{17} = \frac{1}{4} \quad \varphi_{18} = 1$

From this consistent current assignment, we can derive the following minimal integer consistent current assignment:

$\varphi_5 = 12 \quad \varphi_6 = 12 \quad \varphi_7 = 12 \quad \varphi_{14} = 12 \quad \varphi_{19} = 12$

$\varphi_9 = \varphi_{10} = \varphi_1 = \varphi_2 = \varphi_3 = \varphi_4 = 8$

$\varphi_{11} = \varphi_{12} = 4$

$\varphi_7 = 12 \quad \varphi_{15} = 15 \quad \varphi_{17} = 3 \quad \varphi_{18} = 12$

The maximum computation rate $\rho_{19} = \rho' \times \varphi_{19} = 12\rho'$, where ρ' can be determined by applying Theorem 5.2.1.

The example we have worked out clearly shows the utility of Theorem 5.2.1. As we have pointed out in Section 5.2, an achievable bound can be found for ρ_{19} by drawing the timed c-equivalent nets for the Petri net in 6.1.13, and finding the c-equivalent net with the minimum value of fundamental computation rate. This method is not easy and Theorem 5.2.1 gives a far more tractable method, although the bound so computed may be overly optimistic, since the Petri net is not α - SMA.

The timed event graph of Figure 6.1.9 models a production facility in which a deterministic resource sharing strategy was used to share resource units between activities τ_{a1} and τ_{b1}. We pointed out that in order to model non-deterministic resource allocation strategies, we need SMD Petri nets. We show such a system in Figure 6.1.15.

A minimal integer consistent current assignment is one which assigns unit current to each transition. i.e.,

$$\varphi_{a2} = \varphi_{b2} = \varphi_{a1} = \varphi_{b1} = \cdots \varphi_{e1} = \varphi_{e2} = 1.$$

By decomposing the net into its state machine components, we see that only one state machine component has changed, namely the one containing the resource pool and the transitions t_{a1}, t_{a2}, t_{b1} and t_{b2}. The maximum fundamental computation rate of this state machine is $\frac{2}{\tau_{a1}+\tau_{a2}+\tau_{b1}+\tau_{b2}}$, which is the same as that of the simple circuit $t_{a2}\ t_{a1}\ t_{b2}\ t_{b1}$ in Figure 6.1.9. Thus, the non-deterministic strategy does not change the computation rate of the system. This can be seen to be true for deterministic production facilities, as they have a fixed minimal integer consistent

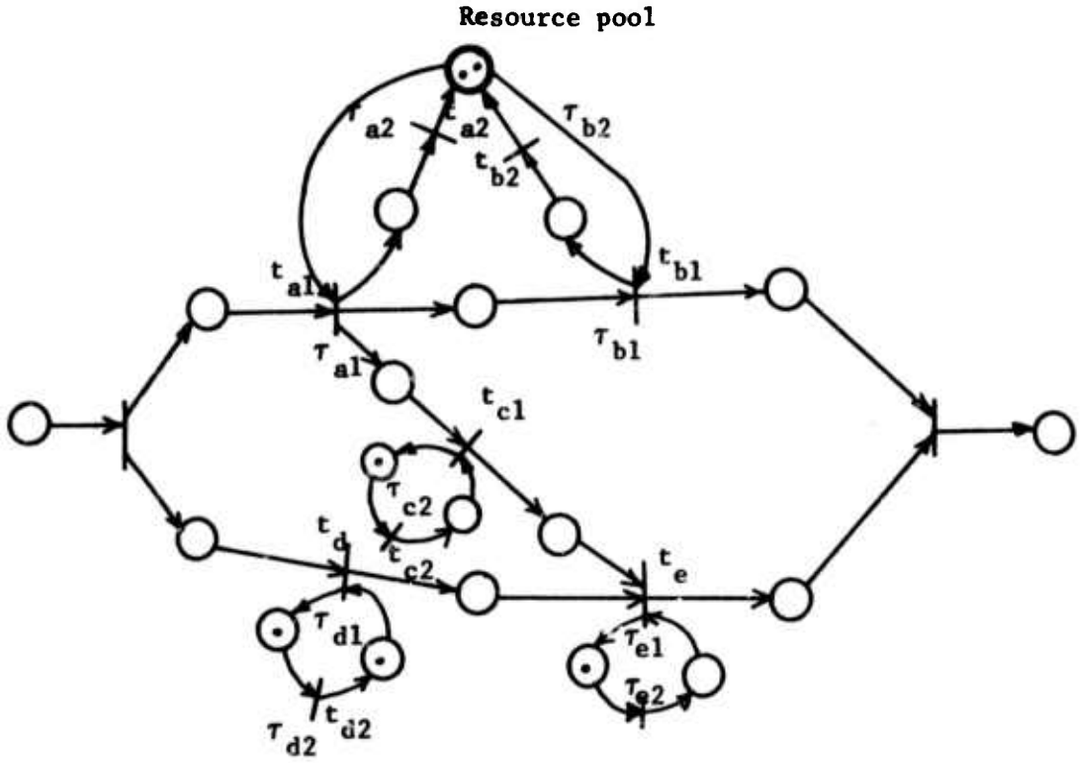


Figure 6.1.15 The Production Facility of 6.1.9 with a Non-Deterministic Resource Allocation Strategy.

current assignment. The reader is invited to construct further examples for himself.

We conclude this chapter with an example of an ensemble of sequential processes which contend for shared resource units in a resource pool. Figure 6.1.16(a) is an SMD Petri net model of a system of three sequential processes contending for two pooled resource units. The minimal integer consistent current assignment models the fact that the relative frequencies with which Processes 1,2 and 3 are allocated resource units

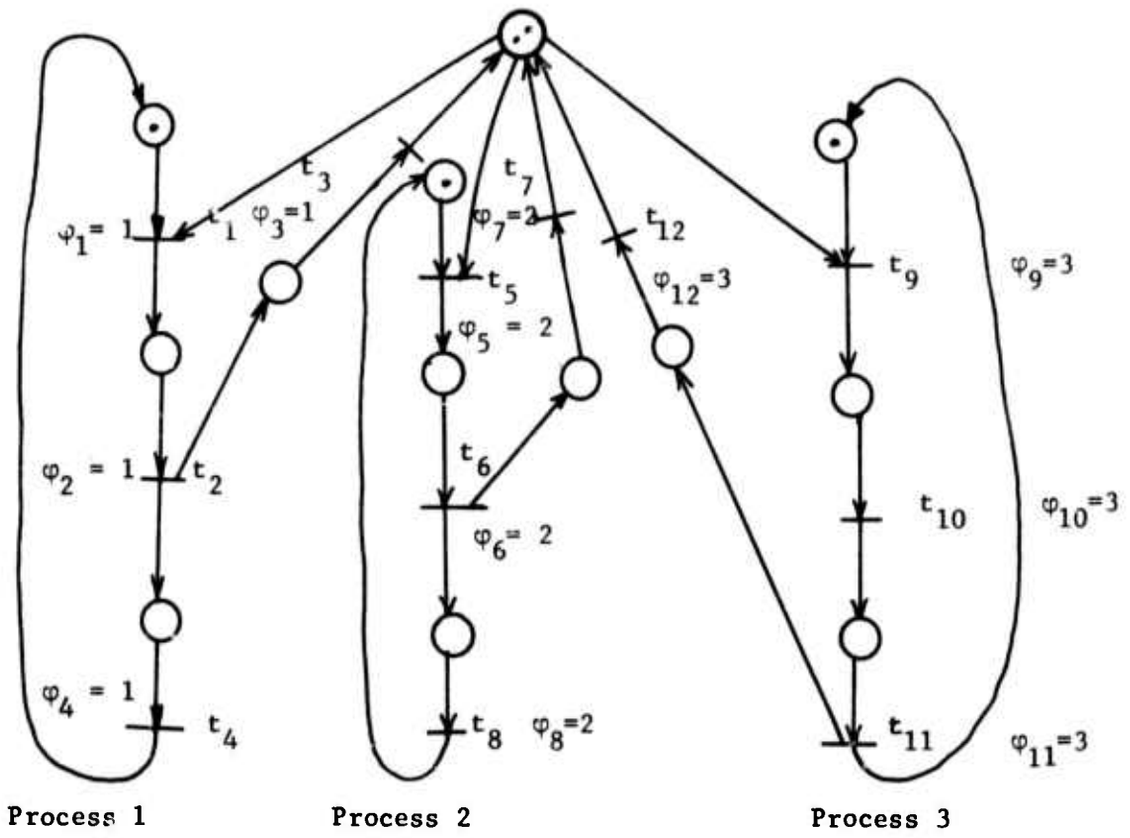


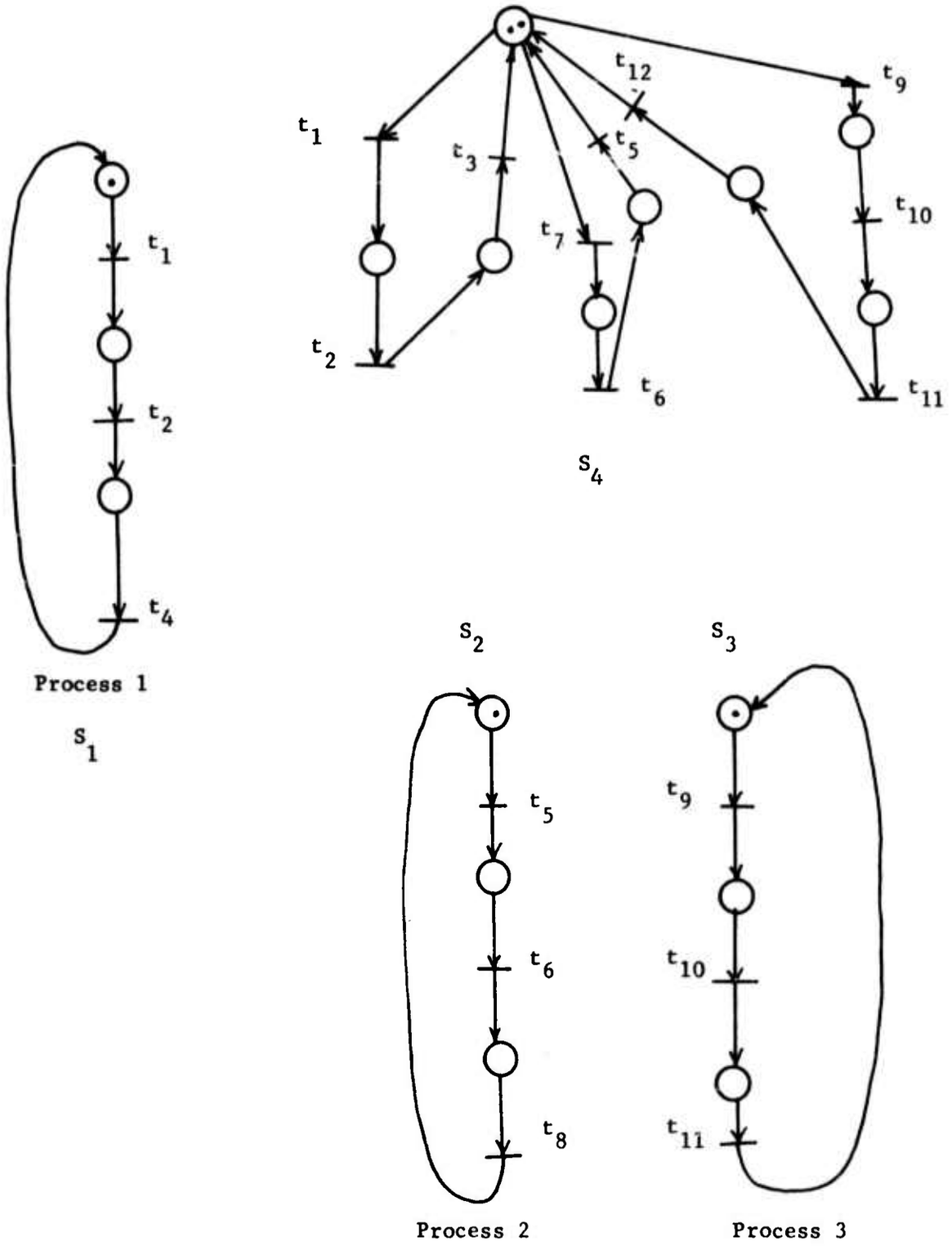
Figure 6.1.16(a)

are in the ratio of 1:2:3 (this is a coincidence). The net is SMA, and the four state machine components designated $S_1,\ldots,S_4$ are shown in Figure 6.1.16(b).

By applying Theorem 5.2.1, the reader can easily find the computation rate of any transitions of interest.

It is hoped that the material presented in this chapter has given the reader a good overview of the applications of the work presented in the earlier chapters. The reader should reinforce his understanding of this material by constructing further examples of his own.

Figure 6.1.16(b)



CHAPTER 7

This thesis has answered several questions but has opened up many others. Basically, we have laid to rest the issues we raised in Chapter 1. We have shown how to model asynchronous concurrent systems in an economical fashion, and, by augmenting our model with timing information, we have shown how to find bounds on the computation rate of a large class of systems.

Our approach in establishing the validity of the SMD Petri net model for asynchronous systems has been heavily dependent on example and intuition. This may seem rather unsatisfying to some of us, and we pose the following problem to our readers: can we come up with a set of axioms that specifies the structure and behavior of asynchronous systems, and then show that these axioms lead to SMD Petri nets?

SMD Petri nets have turned out to be a very rich class of structures for representing asynchronous concurrent systems. However, we do not have any necessary and sufficient structural conditions for an SMD Petri net to have a live marking, and we pose this as another problem to our readers.

The other major question that remains to be examined is to assume firing times to be random variables in order to model real-world systems more accurately than is possible with the model of Chapter 4. Good bounds are needed for the mean computation rate of such timed nets. Another issue that seems to merit some attention is to assume that input queues to an asynchronous concurrent processing systems are fed by items whose arrival rates are random (e.g., Poisson). This will lead to a statis-

tical fluctuation in the processing rate of the system, and the effect of buffers in smoothing out these fluctuations can be examined. A general performance analysis theory can then be worked out for asynchronous concurrent systems. We think that our thesis is a step in that direction.

APPENDIX I

In this appendix, we show that the rooted tree T(V) for any vector addition system V is finite, and the proof of Theorem 2.3.1 is given. The results and techniques in this appendix are taken from Karp and Miller [K2].

Finiteness of the Tree T(V)

To prove that T(V) is finite for any V requires two lemmas. The term subsequence used here does not refer necessarily to successive elements of a sequence. Thus 1, 3, 4, 15, 79,... is a subsequence of 1,2,3,...

Lemma I.1: Let $s_0, s_1, \ldots s_n, \ldots$ be an infinite sequence of elements from $(N \cup \{\omega\})^r$ for some positive integer r. Then there exists an infinite subsequence $s_{i_1}, s_{i_2}, \ldots, s_{i_n}, \ldots$ such that $s_{i_1} \leq s_{i_2} \leq \ldots \leq s_{i_n} \ldots$

Proof: In $s_0, s_1, \ldots s_n, \ldots$ there exists an infinite subsequence that is non-decreasing in the first element. In this sequence in turn, there exists infinite subsequence that is non-decreasing in the second element, etc.

Lemma I.2: (König Infinity Lemma [K3]. Let T be a rooted tree in which each vertex has only a finite number of successors and in which there is no infinite path directed away from the root node. Then T is finite.

Theorem I.1: For any vector addition system V, the tree T(V) is finite.

Proof: Assume otherwise and let $s_1, s_2, \ldots s_n, \ldots$ be a sequence of nodes in an infinite path directed away from the root node. By Lemma I.1 there must exist an infinite subsequence $s_{i_1}, s_{i_2}, \ldots s_{i_n}, \ldots$ of this sequence such that $\ell(s_{i_1}) \leq \ell(s_{i_2}) \ldots \leq \ell(s_{i_n}) \leq \ldots$ Since none of these nodes is an end, it can never happen that $\ell(s_{i_n}) = \ell(s_{i_{n+1}})$. If this were not true, then the path would be finite by condition 2(a) in the definition of T(V).

From condition 2(b) in the definition of T(V), $\ell(s_{i_{n+1}})$ must have at least one more element equal to ω than $\ell(s_{i_n})$ does. Since the number of elements is finite, we have a contradiction, and, therefore, no such infinite path can exist. From Lemma I.2 it must be the case that T(V) is finite.

Proof of Theorem 2.3.1: For any vector addition system V and any integer vector x of the same dimension

$(\exists y \in R(V)$ such that $x \leq y) \Leftrightarrow (\exists \beta \in T(V)$ such that $x \leq \ell(\beta))$.

Proof: We first show that the right hand side implies the left hand side. The idea of the proof is that, if β is a node in T(V), then there are vectors in R(V) which agree with $\ell(\beta)$ in its finite elements, and can be made arbitrarily large in the elements equal to ω by repetition of the sequence of vectors which led to the occurrence of ω. The details of the construction involve some calculation.

Suppose $x \leq \ell(\beta)$. Let the path from s to β have the successive nodes $\eta_0, \eta_1, \ldots \eta_k$, where $s = \eta_0$ and $\beta = \eta_k$. For $j = 1,2,\ldots k$, let v_j be the vector associated with the arc directed into η_j; i.e., $\eta_j = (\eta_{j-1})_{v_j}$. Assume without loss of generality that the first h components of $\ell(\beta)$ are equal to ω, and that the other components are less than ω. Assume further that, in the path from s to β, ω's are introduced in the order $1,2,\ldots h$. Then, for each i, $1 \leq i \leq h$ there exists a consecutive subsequence $t_i = v_{c(i)}, v_{c(i)+1}, \cdots v_{d(i)}$ such that the vector $u_i = v_{c(i)} + v_{c(i)+1}, \cdots + v_{d(i)}$ is positive in the i-th element and nonnegative in the i+1 st through nth elements. Note that t_i is the subsequence that "accounts for" the i-th ω.

Let $-\pi$ be a lower bound on all the (negative) elements of $u_1, \ldots, u_h$. Also let $\{n_1, \ldots, n_h\}$ be any set of non-negative integers satisfying:

$$n_1 \geq (x-s)_1 + \pi(h + L + n_2 + n_3 + \ldots + n_h)$$

$$n_2 \geq (x-s)_2 + \pi(h + n_3 + n_4 + \ldots + n_h)$$

. . . .

(1)

$$\eta_i \geq (x-s)_i + \pi(h + 2 - i + n_{i+1} + \ldots + n_h)$$

. . . .

$$\eta_h \geq (x-s)_h + 2\pi$$

Such a set must exist because of the triangular form of the inequalities.

Choose $s_1, s_2, \ldots, s_{h+1}$ such that for $1 \leq i \leq h$, $s_1 s_2 \ldots s_i$ is the prefix of $v_1 v_2 \ldots v_k$ up to the first occurrence of ω in the i-th element, and $s_1 s_2 \ldots s_{h+1} = v_1 v_2 \ldots v_k$. Then the sequence

$\delta = s_1 t_1^{n_1} s_2 t_2^{n_2} \ldots s_h t_h^{n_h} s_{h+1} = u_1 u_2 \ldots u_f$ has the following properties:

(a) $s + u_1 + u_2 \ldots u_f \geq x$

(b) each partial sum $s + u_1 + \ldots u_i$ is non-negative.

We omit the detailed derivation of (a) and (b) from the system of inequalities (1). To show that the left hand side implies the right hand side, suppose that the following are true: $s + u_1 + u_2 \ldots u_f \in R(V)$, $x \leq s + u_1 + \ldots u_f$, and $s + u_1 + \ldots + u_m \geq 0$, $m = 1,2,\ldots f$, where the $\{u_m\}$ are elements of W. Apply the following operation to the sequence s, $s + u_1$, $s + u_1 + u_2, \ldots s + u_1 + \ldots + u_f$ as many times as possible:

Find the first member of the sequence (call it u') such that, for some earlier member u'', $u' \leq u'$.

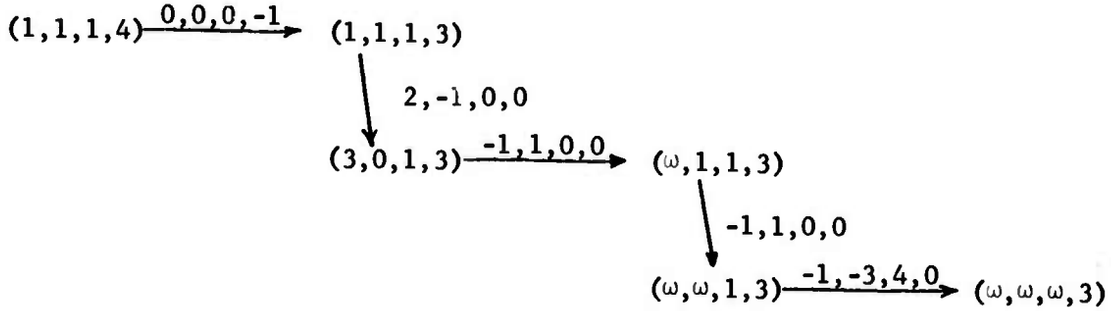
(a) If $u'' = u'$, then delete all members following u';

(b) otherwise, for each i such that $(u'')_i < (u')_i$, replace the i-th element of u' and of each vector beyond u' in the sequence by ω.

It should be clear that the sequence obtained at the conclusion of this process is the sequence of labels in some path directed from the root of T(V), and that the final label in this sequence is a vector greater than or equal to $s + u_1 + \ldots + u_f$. Hence, the left hand side implies the right hand side, and the proof is complete.

To illustrate the construction of the sequence δ given in the first part of the proof, the following example, due to Karp and Miller, is provided:

Suppose $s = (1,1,1,4)$ and $W = \{(0,0,0,-1), (2,-1,0,0), (-1,1,0,0), (-1,-3,4,0)\}$. Consider the following path in $T(V)$:



$s_1 = (0,0,0,-1), (2,-1,0,0), (-1,1,0,0)$ $\quad t_1 = (2,-1,0,0), (-1,1,0,0)$

$s_2 = (-1,1,0,0)$ $\quad t_2 = (-1,1,0,0)$

$s_3 = (-1,-3,4,0)$ $\quad t_3 = (-1,-3,4,0)$

Take $x = (22,16,9,3) \leq (\omega,\omega,\omega,3)$ and let $\pi = 3$. The system of inequalities (1) for this case is:

$$n_1 \geq 21 + 3(4 + n_2 + n_3)$$

$$n_2 \geq 15 + 3(3 + n_3)$$

$$n_3 \geq 8 + 3 \cdot 2$$

A solution is: $n_3 = 14$, $n_2 = 66$, $n_1 = 273$, giving the sequence $\delta = (0,0,0,-1), (2,-1,0,0), (-1,1,0,0), ((2,-1,0,0), (-1,1,0,0))^{273}$, $(-1,1,0,0), (-1,1,0,0)^{66}, (-1,-3,4,0), (-1,-3,4,0)^{14}$, which establishes that the point $(193,23,61,3) \geq x$ is in $R(V)$.

BIBLIOGRAPHY

B1 Baer, J.L., Bovet, D.P., and Estrin, G. Legality and other properties of graph models of computations. JACM, Vol. 17, No. 3, July 1970, pp. 543-554.

B2 Baer, J.L. Graph Models of Computations in Computer Systems. Ph.D. Thesis, Department of Engineering, Univeristy of California, Los Angeles, October 1968.

B3 Berge, C. and Ghouila Houri, A. Programming, Games and Transportation Networks. John Wiley and Sons, Inc., 1966.

C1 Commoner, F., Holt, A.W., Even, S. and Pnueli, A., Marked Directed Graphs. JCSS 5, 1971, pp. 511-523.

C2 Chen, T. C. Parallelism, Pipelining and Computer Efficiency. IBM RJ795 (No. 14666). January 4, 1971.

C3 Clark, C.E. The Greatest of a Finite Set of Random Variables. Operations Research 9, 145-162 (1961).

D1 Dennis, J.B. Modular Asynchronous control structures for a high performance processor. Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, ACM, New York 1970 pp. 5580.

D2 Dennis, J.B. and Patil, S.S. The Description and Realization of Digital Systems. IEEE Conference Record, COMPCON 72, September 1972. IEEE Computer Society, Northridge, California.

D3 Dennis, J.B. Programming Generality, Parallelism and Computer Architecture. IFIPS Conference Proceedings, 1968.

D4 Dijkstra, E.W. Cooperating Sequential Processes, Mathematics Dept., Technological University Eindhoven, Netherlands, September 1965.

D5 Dennis, J.B. and Patil, S.S. 6.232 Course Notes, Department of Electrical Engineering, MIT, Cambridge, Mass. 02139.

D6 Dennis, J.B. and Van Horn, E. Programming Semantics for Multiprogrammed Computations. MIT Project MAC, MAC-TR-23, December 1965.

D7 Dennis, J.B. Private Conversation.

E1 Elmaghrabhy, S.E. An algebra for the analysis of generalized activity networks. Management Science, Vol. 10, No. 3 (April 1964).

F1 Fulkerson, D.R. Expected Critical Path Lengths in PERT Networks. Research Memorandum RM-3075-PR, The Rand Corporation, Santa Monica, California, March 1962.

H1 Hack, M. Analysis of Production Schemata by Petri Nets. MIT Project MAC TR 94, February 1972.

H2 Holt, A.W. Events and Conditions. Applied Data Research, Inc., New York, 1970.

H3 Holt, R.C. On Deadlock in Computer Systems. Technical Report CSRG-6, University of Toronto, April 1971.

H4 Hennie, F.C. Finite State Models for Logical Machines, John Wiley and Sons, 1968.

H5 Hack. CSG Memo 78.

K1 Karp, R.M. and Miller, R.E. Properties of a Model for Parallel Computations -- determinacy, termination and queueing. SIAM Journal of Applied Mathematics, Vol. 14, No. 6, (November 1966) pp. 1390-1411.

K2 Karp, R.M. and Miller, R.E. Parallel Program Schemata. IBM Research Report RC-2053, April 10 1968.

K3 König, D. Theorie der Englichen und Unendlichen Graphen. Akademische Verlagsgesselschaft, Leipzig, 1936.

L1 Luconi, F.L. Asynchronous Computational Structures. Project MAC Technical Report MAC-TR-49, MIT, Cambridge, Mass., February 1968.

M1 Martin, D.F. and Estrin, G. Experiments on models of computations and systems. IEEE Transactions on Electronic Computers, Vol. 16, No. 1, February 1967.

M2 Martin, D.F. and Estrin, G. Models of computational systems -- cyclic to acyclic transformations. IEEE Transactions on Electronic Computers Vol. EC-16, No.1, February 1967.

M3 Martin, D.F. and Estrin, G. Models of Computations and Systems - evaluation of vertex probabilities in graph models of computation. JACM, Vol.14, No. 2 (April 1967) pp. 281-299.

M4 Mac Crimmon, Kenneth R. and Ryavec, Charles A. An Analytical Study of the PERT Assumptions. Research Memorandum RM-3408-PR, The Rand Corporation, Santa Monica, California, December 1962 (DDC Number AD-293-423).

P1 Patil, S.S. The Coordination of Asynchronous Events. Ph.D. Thesis, MIT Department of Electrical Engineering. Project MAC TR-72, MIT, Cambridge, Mass., June 1970.

R1 Ramchandani, C. On the Computation Rate of Asynchronous Computation Systems. Proceedings of the Seventh Annual Princeton Conference on Information Science and Systems, 1973.

S1 Slutz, D.R. The Flow Graph Schemata Model of Parallel Computation. Project MAC Technical Report MAC-TR-53, MIT, Cambridge, Mass., 1968.

B4 Baker, H.G. Private Conversation.

BIOGRAPHICAL NOTE

PII Redacted

Chander Ramchandani was born in West Pakistan,

PII Redacted

The partition of the country two months later led his family to migrate to India and settle down in New Delhi. He graduated with high honors from St. Columba's High School, New Delhi, receiving the General Certificate of Education, 'O' Level, awarded by Cambridge University, in May 1963. He studied Electrical Engineering and Electronics at the Indian Institute of Technology, New Delhi, graduating summa cum laude with the degree of B.Tech. in 1968.

In September 1968, he joined the Department of Electrical Engineering at MIT on a J.N. Tata Endowment Fellowship. Two months later, he joined Project MAC , where he has been a Research Assistant in the Fundamental Studies Group. During summers he has worked at Project MAC (1970) and IBM T.J. Watson Research Center (1971). His interests include computer systems, parallel computation and the modelling and analysis of computer systems and computer networks.

Mr. Ramchandani is a member of Sigma Xi and the Association for Computing Machinery.

PUBLICATIONS

(a) "On the Computation Rate of Asynchronous Computation Systems". Proceedings of the Seventh Annual Princeton Conference on Information Sciences and Systems, March 1973.

(b) "An Analytical Model for Memory Contention in Cache Memory Multiprocessor Systems". Proceedings of the Hawaii International Conference on System Sciences, January 1973.

PII Redacted