PETRI NET APPROACH TO DESIGN AND DEVELOPMENT OF MODERN COMPUTER SYSTEMS

A THESIS

submitted in fulfilment of the requirements
for the award of the degree
of
DOCTOR OF PHILOSOPHY
in
ELECTRONICS & COMMUNICATION ENGINEERING

By

ALI ATHAR KHAN

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING
UNIVERSITY OF ROORKEE
ROORKEE-247672 (INDIA)

September, 1981

Candidate's Declaration

I hereby certify that the work which is being presented in the thesis entitled PETRI NET APPROACH TO DESIGN AND DEVELOPMENT OF MODERN COMPUTER SYSTEMS in fulfilment of the requirement for the award of the Degree of Doctor of Philosophy, submitted in the Department of Electronics & Communication Engineering of the University is an authentic record of my own work carried out during a period from August 1978 to July 1981 under the supervision of Dr. Harpreet Singh, Professor Electronics & Communication Engineering, University of Roorkee.

The matter embodied in the thesis has not been submitted by me for the award of any other degree.

University of Roorkee, Roorkee
Certified that the attached Thesis/
Dissertation has been accepted for the
award of Degree of Doctor of
Philosophy / Master of Engineering
..... vide notification
No. Ex/..162.A/65 (Degree) dated 15.12.82
Assistant Registrar (Exam.)

Allhar
(Ali Athar Khan)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Harpreet Singh
(Harpreet Singh)
Professor of Electronics &
Communication Engineering
University of Roorkee
Roorkee (India)

Date: Sept 12, 1981

ABSTRACT

Petri nets (PN) have aroused considerable interest during the recent years as a model, primarily to represent and study concurrent systems. This thesis deals with the PN approach to design and development of modern computer systems. In particular several aspects of the development of PN theory, its applications to some problems of computer systems and to the problem of optimization in microprogrammed computers have been proposed.

Much of the available work on PN is scattered over various reports, dissertations and journals. The review chapter brings together the existing literature in a coherent manner so as to aid a reader in subsequent chapters.

The design and development of a system demand that a knowledge about the model be known. For this reason, the reachability tree technique and state equation for Petri nets have been proposed by earlier researchers. However, there are many unsolved problems in PN theory. For example, analysis of large PN is generally cumbersome or even impracticable. It is possible to build complex nets with desired properties from smaller nets, the analysis of which can easily be managed. This involves interconnection of nets. It is shown in this thesis how smaller nets could be connected in cascade or in parallel to preserve same properties. A state equation approach has been exploited for this purpose.

Another well-known problem concerning the analysis of PN is the lack of information of firing sequences and existence of spurious solutions of corresponding state equations. This problem is studied and an algorithm is proposed to find minimal legal firing sequences to transform a given marking into another given marking.

It is a well established fact that PN can not model, as such, systems in which interruption or priorities are involved. Many extensions have earlier been proposed but they are either too specific or provide inadequate analysis technique. In this thesis such limitations in modeling capabilities of PN are overcome by proposing an inventor transition in which the token at output place is the complement of the token at input place. All the logic operations have been modeled by PN with additional invertor transitions. To analyse these, a generalized state equation is developed. It is also shown that state equation of PN proposed earlier by Murata is a special case of generalized state equation.

The state equations of Petri net appear to be very powerful. Many problems of computer science, for example, the enumeration of simple paths between two modes of graph, the terminal reliability of a computer network and program complexity evaluation are formulated in the framework of state equations and solutions for them proposed.

Of significant importance in the design of microprogrammed computers are microprogram optimizations. They are called for to reduce the cost of the system and to increase the efficiency.

In this thesis, only bit optimization in control memory and data path optimization are taken up because of their practical utility. In the bit optimization all the maximal compatible classes of microcommands are generated using the PN state equation. From these, minimal bit solutions are obtained. The possibility of further reduction in bits, is also looked into by employing bit steering through extended PN concept.

Two approaches are proposed in this thesis to solve the problem of data path optimization. In the first approach the concept of invariance in PN is employed and solutions which include all the minimal cost solutions, are obtained. In the second approach the problem is reformulated in PN domain. The places of PN are then merged according to defined rules and minimal cost solution obtained.

Finally, the results are summarized and some suggestions alongwith critical discussions for further work are given.

ACKNOWLEDGMENT

The author wishes to express his sincere gratitude to his advisor, Prof. Harpreet Singh for his excellent guidance and encouragement without which it would have been rather impossible to carry out this research.

Special acknowledgment is due to Dr. N.K. Nanda for his keen interest and fruitful technical discussions.

Thanks are also due to Aligarh Muslim University, Aligarh for sponsoring the author under Quality Improvement Programme to persue this research and the Government of India , Ministry of Education for financial support.

The author is very much grateful to his parents, wife and children who have been a constant source of inspitation and encouragement.

Finally, the author is thankful to his friends and colleagues for extending their help directly or indirectly, during this work.

Thanks are also due to Mr. Darshan Lal Jaggi for his efficient typing of the thesis.

TABLE OF CONTENTS

LIST OF SYMBOLS

A	Incidence matrix
B	Number of bits
B_f	Fundamental circuit matrix
C_i	Control field i
E	Number of edges
I	An invariant of a PN
M_o	Initial marking of a PN
M_n	Marking after n th Firing in a PN
M(p)	Number of tokens in a place p
Mc	Number of microinstructions
Nc	Number of microcommands
P	Set of Places
|P|	Number of places
PN	Petri Net
T	Set of transitions
|T|	Number of transitions
V	Number of vertices
W	A weighted vector
W_l	Number of irrendundant words in a ROM
a_{ij}	Element of incidence matrix A
m_i	i th microinstruction
p	A place
${}^{o}p$	Input transitions to place p
p^{o}	Output transitions to place p
${}^{o}p^{o}$	Set of transitions connected to place p

t	A transition
${}^{o}t$	Set of places connected to transition t
o_t	Input places to transition t
t^{o}	Output places of transition t
W	Weight of token in place p
α	Forward incidence function
β	Backward incidence function
$\cup$	Union of sets
$\cap$	Intersection of sets
$\forall$	For all
w	A very large quantity

CHAPTER I

INTRODUCTION AND STATEMENT OF THE PROBLEM

1.1 INTRODUCTION

Modern computer systems comprise of multiple communicating components each of which may itself be a system. Although the interactions between them are well defined, yet they are very complex and the concepts relating to asynchrony and concurrency need close examination. The direct consequence of this is both logical and topological distribution of data, processing and control, which makes representation and performance evaluation results more and more difficult to obtain. Further, the difficulty of representation and analysis of combination of hardware and software systems has increased with the level of sophistication. Hence in order to design a secure and analysable system, the methodology must be able to depict in a formal way the system specification and must help designer to prove their correctness. In search of a formal model to do so, one finds that the tools used for modeling sequential systems are completely inadequate. For instance, block diagrams do represent the interconnections that may exist between them, but give no information about where or when these interconnections are used. A computer system can also be described by comprehensive set of logical diagrams, pertaining to the hardware, and complete listing of the code and microcode necessary to provide system operation. But obviously,

it is too much. The conventional flow diagram becomes excessively clumsy to represent concurrent systems when tried on multi-programming and multiprocessing. Another disadvantage with flow diagram is the difficulty to represent combination of hardware and operating system, as one looses sight of parallel actions and their potential interactions. Finite state machines could be used for representation of such systems but will lead to unmanageably large single states. These difficulties have led to an extensive research to find a suitable model for modern computer systems. Among various models proposed [122], Petri nets and its subclasses [5], [97], [120], [121] have emerged as convenient and powerful tool. They can serve as an intermediate tool between program statement (ckt. diagram) - too complex to analyse and block diagrams- too simple to predict behaviour of a system. This is the prime justification for studying the application of Petri nets to design and development undertaken in this thesis. Another significant achievement is that the knowledge about many models used for manipulating the systems in parallel environment is directly obtained from the study of PN. This is mainly because several models such as computation graphs, flow graph schemata, UCLA graph, vector addition system, vector replacement system, etc. are either included into or are equivalent to PN [122].

1.2 ADVANTAGES OF PETRI NETS

Petri nets find their basis in a few simple rules yet they are very powerful and possess many advantages in modeling a system.

Some of them are:

i. Petri nets often make easier to understand overall system which they represent because of their graphical and precise nature of presentation.

ii. Petri nets are equally suited for representation of hardware and software systems.

iii. Petri nets possess inherent concurrency and parallelism.

iv. Asynchronous nature of Petri nets makes them suitable for representing real systems. In real life, events take variable amount of time. Petri net model reflects this variability by not depending upon a notation of time but contains all the necessary information to define the possible sequences of events of a modeled system.

v. Petri net execution is nondeterministic. The choice as to which transition fires is made randomly i.e. non-deterministically. This feature makes Petri nets to represent real time situation where several events are occuring concurrently and the order of occurences of events is not unique.

vi. Petri nets can be used as heirachical model. This is because they can be used at all levels including networks, register-transfer, functional, and gate etc. Interpretation can be varied to suit all the particular requirements. An entire net can be replaced by a single place or transition for modeling at more abstract level or places and transitions may be replaced by subnets to provide more detailed modeling.

vii. The behaviour of any system can be analysed using Petri net theory. It can, thus, make use of high speed computers as computational tool for modeling and analysing larger and more complex systems than ever before.

viii. Petri nets can be synthesized using both bottom-up and top-down approaches. Methodical design of systems with known or easily verifiable behaviour can be done [6].

ix. Petri nets are a compromise between Modeling Power (ability to correct and faithful representation of the modeled system) and Decision Power (ability to analyse and determine properties of the modeled systems). Generally, in a model, decision and modeling powers are conflicting. For example, finite-state model possesses very high decision as almost all questions about the model are answered, but has very low modeling power. On the other hand, Turing machines have good modeling power but poor decision power.

1.3 LIMITATIONS OF PETRI NETS

It has been found that Petri nets are too simple and limited to easily model real systems. In PN representation of the CDC 6400 operating system [105], Noe has experienced many shortcomings. Normally, a Petri net transition fires when all of its inputs satisfy AND logic. On firing a transition, each output place gets one token. The transitions with such simple firing rules were inadequate to represent the above. As an example, let us consider the execution of a program ISI in operating

system of CDC 6400. This program advances a job from staging queue to input queue if the following is satisfied:

> "Job in staging queue AND only one of the conditions (i) 'No tape required-job advances', and (ii) 'tape-job queued until tape available' is satisfied".

Obviously this cannot be represented by AND logic only. Noe, introduced an Exclusive-OR transition to represent this. Figure 1.1 shows the Petri net representation of the ISI program.

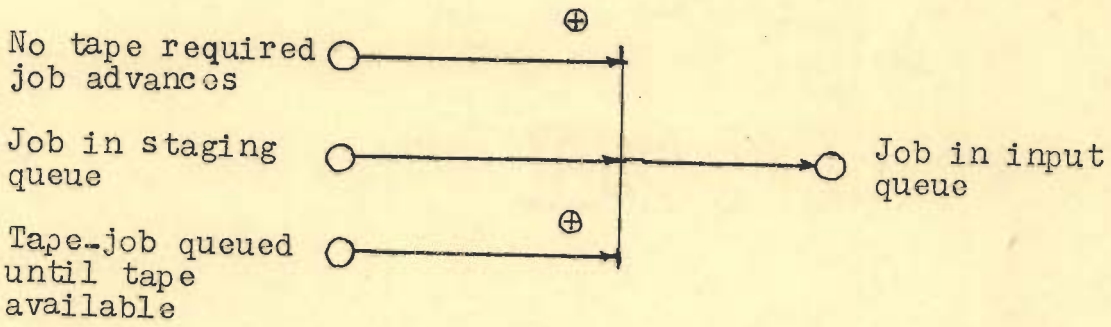


Fig. 1.1 PN Representation of Advancing a Job in Staging Queue of CDC 6400 Operating Systems.

Such deviations from conventional transitions made it impossible to make use of analysis techniques developed for PN.

Another point which was observed is that the tokens represented only conditions but did not carry any attribute. No data structure was associated with the location of nets. This seriously hampered the transfer of data in the system. Further, no time duration was associated with transition firing. This created another obstacle to deal with quantitative measures such as throughput and turn-around time.

Baer [15] faced similar difficulties in modeling a compiler with Petri nets. He extended the PN model by the addition of OR logic, switches and token absorbers.

Patil [115] has created a synchronization problem called Cigarette Smokers Problem. He has shown that this synchronization problem cannot be represented by P and V operations on Semaphores described by Dijkstra [38] or by Petri nets. Following Patil's work, it was shown formally by Kosaraju [81] that Petri nets cannot represent systems where Priority constraints has also to be satisfied. He described a coordination problem of two producer/consumer with shared channel with one producer/consumer having a priority over other. This problem has been shown to fall outside the domain of Petri nets. A similar limitation had been earlier discovered [70]. The difficulties in PN modeling of some relatively reasonable systems have been further demonstrated [3], [7].

It was felt that the limitation on Petri net modeling is due to failure of a transition to react to absence of tokens rather than only to their presence. This inability to test for a zero marking in a place, is known as Zero Testing [70]. Petri nets cannot test an unbounded place for zero. If the place is bounded, zero can be tested. For a bounded place p with bounded k, a complement place p' can be created such that $M(p) + M(p') = k$ for all reachable markings. This allows to zero test M(p) by testing if M(p') is k.

1.4 EXTENSIONS OF PETRI NETS

To overcome some of the problems encountered in the Petri net modeling of CDC 6400 operating system, Evaluation Nets (E-nets) have been proposed [107], [110], [111]. E-nets are extended, interpreted model for parallel computation for performance evaluation and simulation. These represent an approach towards timing information to a Petri net. However, the five primitives proposed in E-nets proved to be too restrictive and it was difficult to model structures with more than two inputs and outputs for transitions. Some larger structures were developed [109] but were still inflexible for general use. A modification on E-nets resulted in Pro-nets [106]. The name Pro- was suggested because of the use of nets for Processors or processes. Pro-nets were allowed with multiple arcs.

Many other extensions to Petri nets like addition of inhibitor arcs [81], constraints [115], exclusive -OR transition [105], Switches [15] etc. have been suggested. Invariably all of them introduced in one way or the other, the reaction of transitions on absence of tokens as well. But they were introduced, basically to solve particular problems rather than with intention to increase the modeling capacity of Petri nets. An extensive study on the completeness of different models with extended Petri nets have been made by Agerwala et al. [1], [2], [7]-[9]. It has been observed that adding zero testing allows a Petri net to simulate a Turing machine. Thus a PN with zero testing produces a modeling scheme which can model

any system. However, many analysis questions of Petri nets become undecidable, since they are undecidable for Turing machines. This may perhaps be one of the reasons that analysis technique is not, generally, available for generalized extension of Petri nets.

In Chapter IV an invertor transition is defined which describes a NOT operation. This facilitates the representation of all logic operations with Petri nets alongwith invertor transitions. To analyse, such a Petri net, a generalized state equation is proposed. This overcomes many shortcomings in Noe's model [105] and makes the analysis of the systems possible.

1.5 STATEMENT OF THE PROBLEM

This thesis attributes itself to the problem of Petri net approach to the design and development of modern computer systems. Specifically, the problems considered in this thesis can be stated as follows:

i. To evolve efficient and reliable techniques for solving some of the unsolved and outstanding problems in PN theory with a view to develop a basis for PN approach to several challenging aspects of modern computer design.

ii. To utilize the modeling of PN (particularly, state equation representation), with a view to provide automated tools, for formulating and solving various problems of computer hardware and software.

iii. To exploit the use of Petri nets in design and development of microprogrammed computers.

Though some aspects of the above mentioned problems have been studied by a few investigators, however, not much results have been obtained. For example, as regards the first problem a decomposition technique [137] for analysing large Petri nets does exist, but it is quite involved. Furthermore, only the necessary condition for reachability [97] has been obtained. So far as the second problem is concerned, surprisingly no work of similar nature has appeared in the literature. For the design of microprogrammed computers, only the conventional methods based upon classical approaches are available.

1.6 ORGANIZATION OF THE THESIS

Petri nets are becoming increasingly popular for the representation and analysis of system in general and computer systems in particular. Before it can be applied effectively some bottlenecks which appear in the theory need be removed. An attempt has been made in this thesis, to first identify those, and then to propose techniques for removing them.

Many problems in computer hardware and software can be represented by PN and can be solved by similar set of equations. Such investigation is another type of work reported in this thesis.

Invariably every modern day computer utilizes the concept of microprogramming. This has called for the need of optimization in order to have a compromise between flexibility and cost. Petri nets appear to be a natural representation for such computer systems and can provide optimization techniques in a more

systematic and easy-to-beimplemented way. This is another aspect which has been taken up here. It may be noted that almost no work except from one author [136] apart from the candidate's work on the optimization consideration of Petri net is available.

For the sake of better understanding the following arrangement has been adopted in the organization of this thesis.

The review of the existing literature related to Petri net approach in computer systems has been included in the second chapter. This chapter also contains basic concepts, important properties and applications of Petri nets scattered over many journals, reports and dissertations. Comments regarding the analysis problems of Petri nets have also been given as and when required.

Chapter III discusses the basic concepts of microprogramming, the different optimizations required therein, and the outline of the existing techniques for them. The justification for the importance of optimizations for control bit memory and interconnections of modules alongwith the comments regarding the computational complexity of the available techniques has also been made.

Chapter IV addresses to the solutions of some important problems in PN theory. First of all an investigation for the interconnection and decomposition properties of Petri nets has been carried out and useful results obtained. A new and computationally better technique for decomposition is also proposed. Another important aspect involved in the theory is that of reachability because many properties of PN are directly dependent

upon it. Therefore, it is imperative that an efficient algorithm for solving reachability problem be devised. In this direction a technique has been formulated through the determination of minimal legal firing sequences which transforms one marking into another. Furthermore, to increase modeling capabilities of a PN, an invertor transition has been proposed. This takes care of negation operation often encountered in systems. The analysis technique of PNs having such transitions has also been presented in this chapter. This technique not only allows the representation of all the logic operations by Petri net, but makes possible the analysis of many computer systems incorporating those operations. Many examples are taken up for explanation purposes.

Chapter V identifies some of the classes of state equation for Petri nets, which can solve many problems in computer hardware and software by the same technique. The proposed classes take care of the problems of enumeration of simple paths between two nodes of a graph, terminal reliability of a computer network, program complexity evaluation, and determination of maximal compatible classes (MCCs) of microcommands in control memory bit optimization. A novel technique for the solution of state equation of different classes is proposed and its superiority established by using examples considered by earlier researchers.

In Chapter VI, the application of PN to the design and development of microprogrammed computers is explored. Particularly, the optimizations involved therein have been studied. This is entirely a new approach. The control memory bit optimization is obtained first by enumerating all MCCs via PN

application and then putting the corresponding microcommands in different blocks so as to yield optimal solutions. From these solutions the number of bits are further reduced by exploiting the concept of bit steering [10] through extended Petri net. Advantages of the proposed technique over the existing methods have been highlighted. An example considered by many investigators, has been reconsidered to show the utility of the proposed technique. Another problem i.e. optimization of data path has been solved by first representing the data transfers among modules of the system by a Petri net and then by defining and applying the concept of merging-in of places. The proposed method is simple and requires less computational efforts. A comparison of this is made with the existing methods by way of a numerical examples considered earlier [85].

A summary of the work done has been given in Chapter VII. A brief outline for further work has been included in this chapter.

CHAPTER II

CRITICAL REVIEW AND GENERAL CONSIDERATIONS OF PETRI NET

2.1 INTRODUCTION

Petri nets have emerged in the last decade into a very powerful and suitable model to represent, analyse and synthesise a very large and interesting class of systems exhibiting concurrency. This is because Petri nets can provide with a minimal amount of effort, a simple, natural and easy-to-understand representation. Research on Petri nets have focussed on the representation of computer hardware [5], [34], [62], [100], [108], [120], [121], computer software [5]-[7], [15], [16], [20], [28], [83], [105], [120], [121], [136], speed independent circuits [95], [112], [113], [118], production schemata [60], communication protocols [90]-[92], [100], [126], [127], asynchronous arrays [68], [117], [131], performation evaluation of computer systems [109], [111], formal language theory [19], [29], [55], [119], legal systems [89], mathematical knowledge [42], [49], propositional calculus [44], [142] etc. Analysis and synthesis of Petri nets and their subclasses with a motivation to provide some properties of the modeled system, is another area which has received a larger attention from researchers [6], [7], [11], [21], [26], [27], [43], [48], [49], [60], [67], [68], [83], [97]-[103], [136], [147], [148]. Research has also been carried out to increase the modeling capability of Petri nets [1], [2], [7],

[8], [9], [15], [81], [105], [115]. There is an ever-increasing interest in Petri nets as is evidenced by the abundance of recent work reported in most of the major conferences in the area of computers and information processing. A critical survey of the work done in this field is embodied in this chapter.

2.2 HISTORICAL REVIEW

The theory of Petri nets was originated in C.A. Petri's dissertation 'Kummunication mit Automaten' [123] in Germany. In his thesis, starting with the concept that many events of communication such as asynchronous and concurrent operations can be represented by purely combinatorial - topological means, Petri proposed a new model of information flow in systems. In 1965, the ideas of Petri received the attention of a group of researchers led by A. Holt at Applied Data Research, Inc., U.S.A. The group working on the Information System Theory Project [61] was concerned with finding a proper descriptive means for modeling, evaluating, and implementing systems. Here, it was shown how Petri nets could be applied to concurrent systems and the concept of Petri nets was refined and developed to such a state that it is applicable to many areas. In fact the actual definition of Petri nets, as used today, is due to Holt et al.[60].

The research on Petri net, at about same time, was also carried out under Project MAC at MIT, U.S.A. A large contribution was made particularly by the Computation Structure Group under the direction of Prof. J.B. Dennis. Examples of the use of Petri

nets for the description of control mechanisms of complex computers were given with a goal to develop automatic mechanisms for implementing the Petri net as a digital system [34]. Several Ph.D. theses, M.S. dissertations, numerous reports and papers on Petri net were produced [18], [19], [34], [35], [41], [49], [51]-[53], [56], [113]-[116], [118], [130]. The Computation Structure Group also organised the Project MAC Conference on Concurrent Systems and Parallel Computation in 1970 at Woods Hole [34] and the Conference on Petri Nets and Related Methods in 1975 at M.I.T.

The outcome of investigations at Applied Data Research and MIT, and the two conferences triggered the research activity in nature and the applications of Petri net. However, there was a similar independent research going on in Europe, particularly at the Institute fur Informationssystemforschung of the Gesellschaft fur Mathematik und Detneverarbeitung in Bonn. The institute is now involved in finding a more general and abstract theory.

Much of the work on Petri nets is in the form of theses, dissertations, reports and memos. These are neither widely circulated nor readily available. The first readily available work was by Baer [14]. However, this was mainly a survey of some of the theory developed for parallel computation. Several models including Petri nets were presented. But, there was hardly any paper in a journal of international repute, which could communicate, in a coherent manner, the work done in the field of Petri nets. In 1977, Peterson published an excellent paper [120] which is both a survey and tutorial on Petri nets. Another excellent

paper by Murata [98] came up the same year. This is a tutorial in nature but aims mainly to introduce Petri nets to those who are working in circuits and systems. These two were followed by another paper [5] which brings together a large body of work on useful applications of Petri nets.

It appears that interest in Petri nets is ever increasing. A workshop was held in Paris in 1977. Petri is still continuing his work and has extended the concepts in a form of general system theory called General Net Theory [124]. An advanced course on General Net Theory was held in Hamburg in 1979. A special interest group on Petri nets has also been formed in Germany. Research in and application of Petri nets has become widely popular.

Having given a brief historical review; the concept of Petri nets, their properties required for studying the systems modeled and recent work in the analysis of Petri nets are discussed further in the following sections.

2.3 WHAT ARE PETRI NETS?

Petri nets are an abstract and formal model of information flow. The concepts, properties and techniques of Petri nets are outcome of research by a number of people working at different times in different places with different backgrounds and motivations. This has resulted in many class of Petri nets and many concepts defined in different ways [27], [56], [60], [61], [113], [123]. However, there seems to be no substantial difference between

between them. Each of them is a restriction, in one way or the other, on general Petri nets. Here the most general and widely used concepts are presented.

The representation of Petri nets is done both graphically and mathematically. While the graphical representation of Petri net structure is useful in illustrating the concepts, mathematical representation is required for analysis. Both these representations are given side by side.

A Petri Net Graph is a Directed Bipartite graph with two types of Nodes called Places and Transitions. A Circle represents a place and a Bar represents a transition. Directed Arcs connect the transitions and places. Only single arc between a transition and a place is allowed in Ordinary Petri Nets or hereafter called Petri nets. A place p is called an Input (Output) place of a transition t_i if there exists a directed arc from (to) place j to (from) transition t_i. Fig. 2.1 is an example that represents a Petri net graph.

These concepts are defined mathematically as follows:

Definition 2.1 : A Petri net is a quadruple $N = (P, T, \alpha, \beta)$ where:

- P is a set of places, $P \neq \emptyset$
- T is a set of transitions, $T \neq \emptyset$, $P \bigcap T = \emptyset$
- α forward incidence function
- β backward incidence function
- α,β are binary relations with $\alpha, \beta \subseteq P \times T$, field $(\alpha,\beta) = P \bigcup T$

Definition 2.2 : Let $N = (P, T, \alpha, \beta)$ be a PN. We call ${}^{o}t$ (t^{o}) set of input (output) places of t and by analogy, ${}^{o}p$ (p^{o}) set of input (output) transitions of p where:

For $t \in T$, ${}^{o}t = \{p \in P | \alpha(p,t) \neq 0\}$ and $t^{o} = \{p \in P | \beta(p,t) \neq 0\}$

For $p \in P$, ${}^{o}p = \{t \in T | \beta(p,t) \neq 0\}$ and $p^{o} = \{t \in T | \alpha(p,t) \neq 0\}$

The set of places (transitions) connected to a transition t (place p) is denoted by ${}^{o}t^{o}$ (${}^{o}p^{o}$).

These notations are extended to subsets of T and P, for example, if $P_1 \subset P$ then, ${}^{o}P_1 = \bigcup_{p_k \in P_1} {}^{o}p_k$

The structure of Petri nets can also be described by their Incidence Matrix [83], [97].

Definition 2.3 : Let $N = (P, T, \alpha, \beta)$ be a PN, the incidence matrix A of the PN is defined as

$$A = [a_{ij}]_{|P| \times |T|} \text{ with}$$

$$a_{ij} = \begin{cases} 1, & \text{if i th transition has an outgoing arc to place j} \\ -1, & \text{if i th transition has an incoming arc from place j} \\ 0, & \text{otherwise} \end{cases}$$

The Petri net can also be defined as $N = (A, —)$.

The graphs as of Fig.2.1 will represent only static (time-independent) behaviour of the Petri net. In order to simulate the dynamic behaviour, each place in a Petri net is Marked (assigned) with a non-negative number of Tokens. The tokens are represented by Dots in circles representing places. The token distribution in

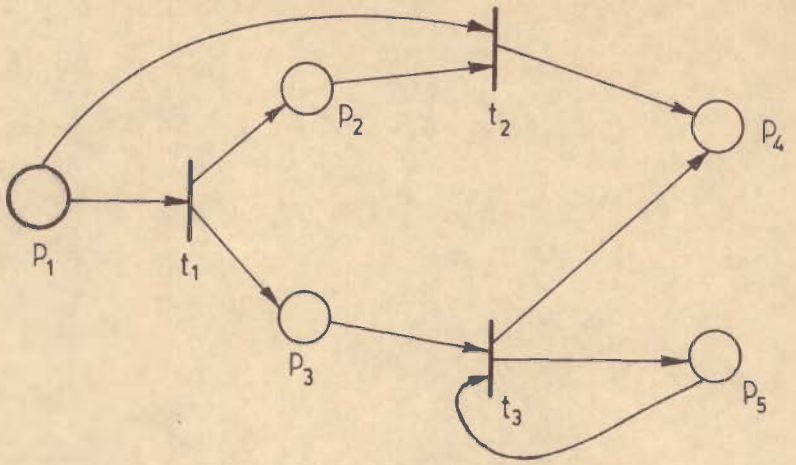


FIG. 2.1 _ A PN GRAPH

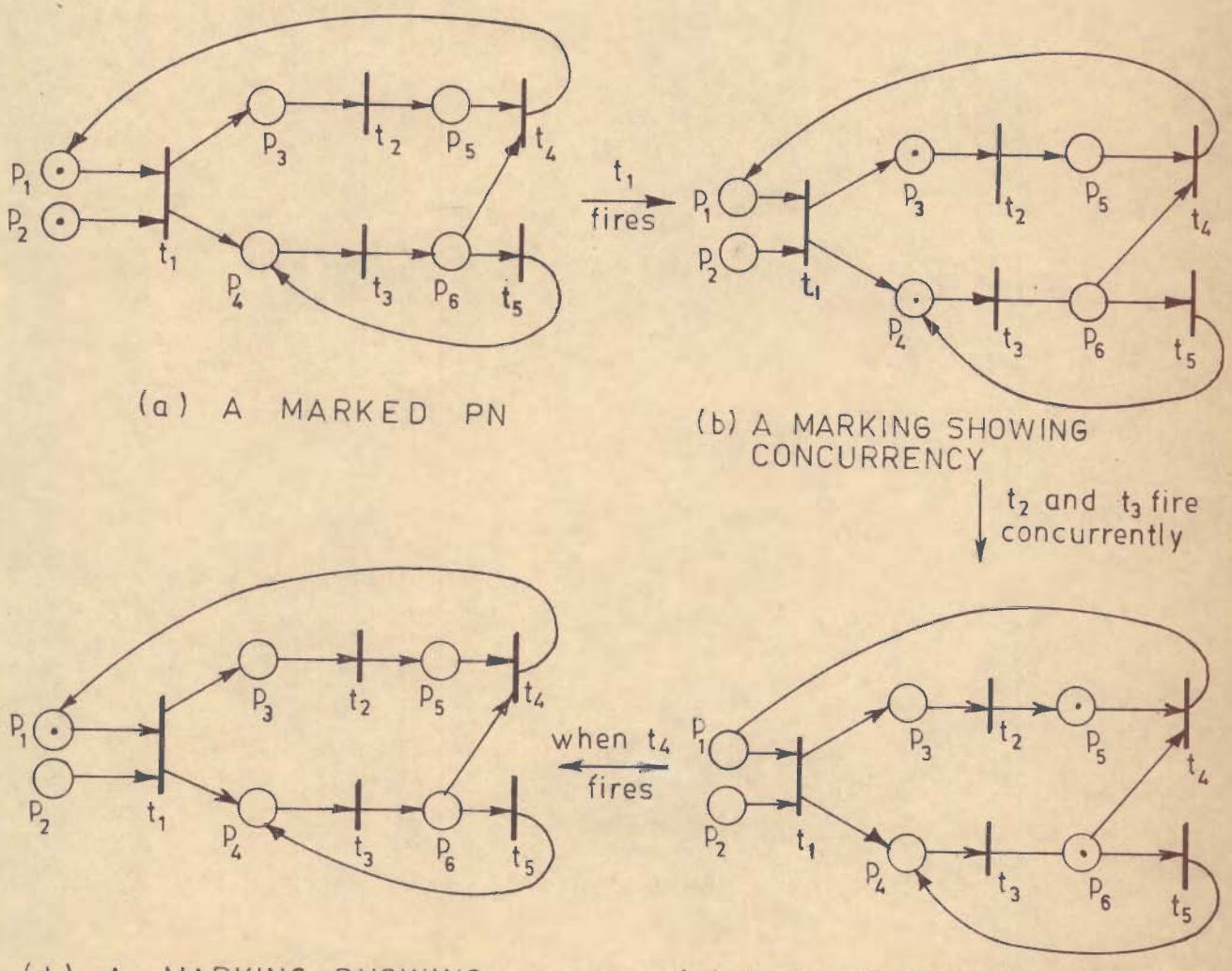


FIG. 2.2 _ PN SHOWING MARKING, CONCURRENCY, CONFLICT, DEADLOCK AND REACHABILITY

a Petri net is called Marking or State Space or PN.

Definition 2.4 : A marking M of a PN, N = (P, T, α, β), is a mapping of P into IN (the set of natural integers : 0,1,2,...): $P \xrightarrow{M} IN$. The number of tokens in a place p is denoted by M(p). A marking M can also be represented as a |P| x 1 column vector of non-negative integers, the j th entry of which denotes the number of tokens in place j.

The dynamic behaviour of a PN is obtained by the position and Movement of tokens i.e. change in marking. Marking of a PN changes as a result of Firing of a transition. Not all transitions can fire. Only those transitions which are Enabled can fire. However, an enabled transition will fire only if it is asked to do so. The firing of an enabled transition is called a Legal Firing. The firing rules of a transition are as follows:

1. A transition is 'enabled' or 'firable' if each of its input places contains at least one token.
2. An enabled transition will fire on application of a control signal.
3. On firing an enabled transition, one token from each of its input places will be removed and one token will be added to each of its output places.

Definition 2.5 : A PN, N = (P, T, α, β) with a marking M is a Marked Petri Net C = (P, T, α, β, M) or C = (A,M).

Definition 2.6 : A transition t is enabled to fire if

$$\forall\, p \in {}^{o}t : M(p) > 0$$

Firing of t in M yields a new marking M' where :

$$M'(p) = \begin{cases} M(p) - 1 & \text{if } p \in {}^{o}t,\ p \notin t^{o} \\ M(p) + 1 & \text{if } p \in t^{o},\ p \notin {}^{o}t \\ M(p) & \text{otherwise} \end{cases}$$

In Fig.2.2a, as an example, only transition t_1 is enabled to fire. When t_1 fires, the marking changes. The new marking is shown in Fig.2.2b. The transitions t_2 and t_3 now become firable. Since there is no common place input to both t_2 and t_3, the two enabled transitions do not affect one another in any way. Thus t_2 and t_3 can fire Concurrently. After the firing of t_2 and t_3 is complete, the places p_5 and p_6 will have one token each (Fig.2.2c) enabling the transitions t_4 and t_5. However, firing of one will Disable the other. This is called a Conflict. In such a case the decision as to which transition fires is arbitrary. This ability to represent both concurrency and conflict makes Petri nets a very powerful modeling device.

The marking of Fig.2.2b is obtained from the marking of Fig.2.2a by firing only one transition t_1. The marking of Fig.2.2b is called Immediately Reachable from the marking of Fig.2.2a. The marking of Fig.2.2d is obtained by firing the transitions in sequence $t_1 t_2 t_3 t_4$ or $t_1 t_3 t_2 t_4$. The marking of Fig. 2.2d is said to be Reachable from the marking of Fig.2.2a. Thus,

Definition 2.7 : A marking M' is immediately reachable from M if the firing of some t in M yields M'.

Definition 2.8 : A marking M' is reachable from M if there exists a legal firing sequence which transforms M to M'.

Definition 2.9 : The Reachability Set $R_N(M)$ of a marked PN, $N = (P, T, \alpha, \beta, M)$ is the set of all markings reachable from M. Whenever there is no ambiguity R(M) will be used instead of $R_N(M)$.

Definition 2.10 : The Reachability Problem is defined as: Given M' , is $M' \varepsilon R(M)$?

Other concepts of Petri nets can also be explained with the help of Fig.2.2. It is observed here that the token in any place and in any marking is atmost one. Such a PN is called Safe or 1-bounded

Definition 2.11 : A place in PN, $N = (P, T, \alpha, \beta, M)$ is k-bounded if and only if there exists a fixed k such that $\forall M' \varepsilon R(M)$: $M'(p) \leq k$. A place is safe if it is 1-bounded. A marked PN is bounded if each place is k-bounded for some k. A marked PN is safe if each place is safe.

For a Petri net which is to model a real hardware device, one of the important properties is boundedness and in special case safeness. If a place is safe, then the number of tokens in the place is either 0 or 1. Thus the place can be implemented by a single flip-flop. In an ordinary PN, if a place p is not safe, then p can be forced to be safe by supplementing it by another place p' [121] in the following manner:

If $p \varepsilon {}^{o}t$ and $p \notin t^{o}$, then add p' to t^{o}

If $p \varepsilon t^{o}$ and $p \notin {}^{o}t$, then add p' to ${}^{o}t$

However, if a place is not safe but bounded, it can be implemented by a counter. Thus a bounded PN could be realized in hardware while a PN with unbounded place can not in general be implemented.

Another important property of a PN is the representation of Deadlock which has been the subject of a number of studies in Computer Science [59]. A deadlock in a PN is a transition (or a set of transitions) which can not fire. Consider Fig.2.2d. None of the transitions can fire and PN is said to be deadlocked. A transition is Live if it is not deadlocked.

Definition 2.12 : A transition t in a marked PN, $N = (P, T, \alpha, \beta, M)$ is live if for each $M' \varepsilon R(M)$ there exists a marking reachable from M' in which t can be fired. A marked PN is live if each transition is live.

Definition 2.13 : A marked PN, $N = (P, T, \alpha, \beta, M)$ is free from deadlock if for $\forall M' \varepsilon R(M)$, some $t \varepsilon T$ can fire in M'.

So far the discussion has been limited to PNs as an abstract model. When it represents a real system, a meaning or Interpretation is assigned to various entities - namely, transitions, places and tokens. Thus a transition in a PN may represent an event, instruction or a program. A place can represent a condition or a type of resource etc. whereas tokens will represent holding of a condition or number of resources etc.

An important property in Petri nets is Conservation of tokens. If toknes are used to represent resources, then the tokens must be conserved because the resources can neither be created

nor destroyed. One way to do this is to maintain total number of tokens in the net, constant for every marking.

Definition 2.14 : A marked Petri net, $N = (P, T, \alpha, \beta, M)$ is Strictly Conservative if $\forall M' \in R(M)$: $\sum_{p \in P} M'(p) = \sum_{p \in P} M(p)$

This implies that each transition in a conservative net must have equal number of input and output places i.e. $|{}^{o}t| = |t^{o}|$. If it were not so, firing transition t will change the number of tokens in the Petri net. More generally, weights can be defined for each place as long as weighted sum is constant [84]. This still allows the conservation of resources because there is no one-to-one mapping between takens and resources. Some tokens may represent program counters etc., and a token may represent several resources. This token is later used to create one token for one resource by firing a transition. Hence a generalized conservation is defined.

Definition 2.15 : A marked Petri net, $N = (P, T, \alpha, \beta, M)$ is conservative with respect to a weighting vector $W = [w_p]_{|P|x1}$ with $w_p \geq 0$, if

$$\forall M' \in R(M): \sum_{p \in P} w_p . M'(p) = \sum_{p \in P} w_p . M(p)$$

2.4 MODELING WITH PETRI NETS

Petri net has found its way in modeling a variety of systems. Holt, et al. [60] have shown that Petri nets can model two aspects of the systems, Events and Conditions. In their view, a token may be thought of as representing the presence of some condition associated with its place. The firing of a transition is

thought of as corresponding to the occurence of an event which may take place if all the necessary conditions are satisfied. The occurence of an event will cause some of previous conditions to cease holding, and causing other conditions to begin to hold. This is represented by a new marking. Many systems can be covered into events and conditions and, thus, can be represented by a PN [121].

2.4.1 MODELING OF HARDWARE

One of the important features of Petri nets is their ability to model computer hardware at different levels. At the lowest level i.e. simple memory devices and gates, computer systems can be described by state machines and hence by Petri net [121]. Though the PN discription is a bit complicated compared to state machine description, it has certain advantages in combination of machines. The combination in state machine is complex and requires a composite state with components of many submachines - a Cross-Product machine. For Petri nets, the composition is simply the Cascade connection (i.e. the overlapping of the output places of one net with input places of another and so on), or Parallel Connection (i.e. duplicating the input tokens which represent input symbols). It appears such interconnections of Petri nets have not been studied so far and have been taken up in Chapter IV.

An example for modeling computer hardware at the level of registers as fundamental components of the system is the representation of n-stage pipelined operation through Petri nets [5], [100], [120], [121]. It is interesting to note that two

successive stages could be modeled in more detail by a PN, if one is interested to ensure that each pair of successive processors communicates through 'ready' and 'acknowledgement' signals.

Another approach to build very fast large computer systems, is to provide multiple functional units to perform computations on multiple registers with maximum possible parallelism. Computers such as the CDC 6600 [143] and the IBM 360/91 [12] are based on this concept. These can be modeled by Petri nets though they will require very complicated and large nets [121]. Dennis [34] has modeled through Petri nets a functional unit computer which resembles CDC 6600. Petri nets have also been used to model the interconnection of hardware modules [62], a modular micro-programmable computer [108] and I/O devices of a mini-computer [150].

2.4.2 MODELING OF SOFTWARE

The efforts in modeling computer software have resulted in different concepts and techniques in analysis, specification and description of programs. A program has two aspects - computation and control. Petri net can represent in a straightforward manner the control aspects (i.e. the sequencing of instructions and flow of information and computation) but not the actual information values. The flowchart representation of a sequential program can easily be converted into a Petri net. The transitions are associated with the actions of the program, i.e. the computations and decisions. A token residing in a place means that the program counter is positioned ready to execute the next instruction.

Parallelism in a program has also been represented by PN [5] but can be exploited usefully only when the Component Processes Coordinate. Such a coordination requires sharing of information. Many synchronization problems arising in coordinating processes, for example mutual exclusion problem [37], producer/consumer problem [38], the dining philosphers problem [38], and the readers/writers problem [28] have been proposed. Petri nets can clearly and explicitly represent these [5]-[7], [28], [83], [120], [121]. Also available is the analysis for system verification [5]-[7], [83].

Petri nets have been applied to compiler modeling to determine whether existing compilation algorithms are suitable for parallel processing [15], [16]. The Fortran programs for CDC 6600 computers have been converted into PN showing precedence constraints between operations [136]. This net is then merged with a PN representing the CPU. Timing information is associated with transitions, and an exhaustive search is used to determine the sequence of operations to minimize execution time. The SOLO operating system and the Scope operating system of CDC 6400 computers have also been modeled by Petri nets [20], [105]. However, the latter has used some extensions such as addition of Exclusive-OR to Petri Nets.

2.4.3 SPEED-INDEPENDENT CIRCUITS

In a speed independent circuit the presence of arbitrary delays in elements and connections have no effect upon circuit operations. Synthesis of music through a processor [95] is one of

many examples. Petri net has a potential for describing such circuits [118]. Places, tokens and transitions could represent wires, signals and actions. In addition to modeling, implementation of Petri nets has also been studied [95], [112], [113], [118], and shown to be inherently fail-secure [112]. However, there are many unsolved problems, like fault detection and isolation. This is mainly because these problems require the use of timing information.

2.5 SUBCLASSES OF PETRI NETS

Generalized Petri nets and for that matter Petri nets, are too powerful to analyse. Many researchers have defined, by restrictions on the structure of Petri nets, many subclasses with intentions mainly to improve their analysing capability. These subclasses do model several systems in different environments but, obviously, their modeling power is limited. The important subclasses are discussed as follows:

2.5.1 MARKED GRAPH

Marked graph is a subclass of Petri nets in which each place has exactly one outgoing and one incoming arc. These arcs are combined into one to represent the place. Further, a vertex represents a transition. The signals or data passing through each arc come from a predetermined source (the initial node), and are sent to predetermined destination (the terminal node). Thus, marked graphs can represent concurrency but not conflict. This limits its modeling power. However, computer systems such as

communication protocols in distributed computing [91], [128], n-stage pipelined operations used in high performance computer systems, parallel activities between central processing and disc or I/O jobs [111], and the changing operations in GRAY-1 computer [128] are but a few examples that can be modeled by marked graph [100]. Marked graph, on the other hand, has been shown to have very high decision power [26], [27], [43], [60], [64], [67], [98], [99], [101]-[103]. There are algorithms available for liveness, safeness and for solving reachability problem.

2.5.2 STATE MACHINES

State machines [60], [120] are restricted Petri nets so that each transition has exactly one input and one output place. These are in the class of finite-state machines, hence are very powerful as far as decision problems are concerned. The modeling power is, however, limited.

2.5.3 FREE-CHOICE PETRI NETS

A free-choice Petri net is one wherein every place p is either the only input place of a transition or there is atmost one transition which has p as one of the input places. This means that either the token will remain in that place until its unique output transition fires or if there are multiple outputs for the place, then there is a free-choice of firing a transition. Hence, either all of these conflicting transitions are simultaneously enabled, or none of them are. It has been shown [26], [49] that liveness and safeness are decidable for free choice Petri nets. Although these nets are very helpful in modeling systems similar

to that of assembly-line, no work is available regarding other properties like reachability, equivalence, containment, and languages , etc.

2.5.4 PURE OR RESTRICTED PETRI NETS

A pure Petri net is one in which no place is both input and output of the same transition. It has good modeling power and has been shown to represent [83] a Semaphore by Dijkstra [38], bounded buffer problem [48] and five dining philosphers problem [38]. As Pure Petri nets are equivalent to PN and each can be transformed into another as far as reachability is concerned, these have the same decision power as that of Petri net. Furthermore, structural properties have been studied [83], [137] to decide many problems like liveness, broundedness in terms of invariance and consistency of pure nets.

2.5.5 SIMPLE PETRI NETS

In simple Petri nets [49] each transition has atmost one input place which is shared with another transition and so also serve to restrict the manner in which conflict can occur. No investigations have been made about the properties of this sub - class of Petri nets.

A simple chart showing some of the subclasses of Petri nets with allowed and not allowed configuration is given in Fig.2.3. It can be easily found by inspection as to why the configuration on the right hand of Fig.2.3 are not allowed.

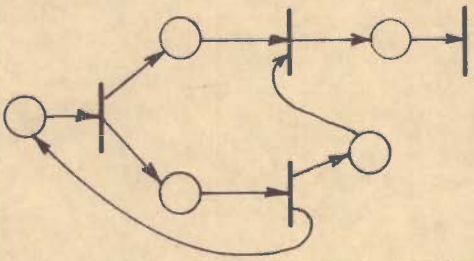
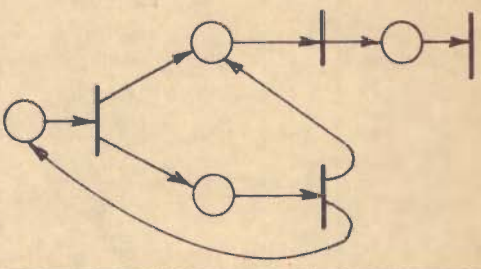
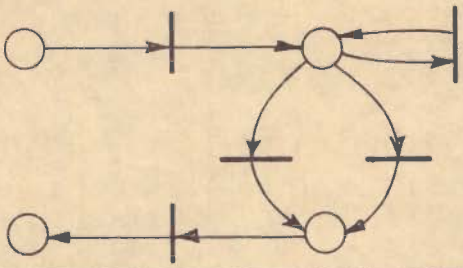
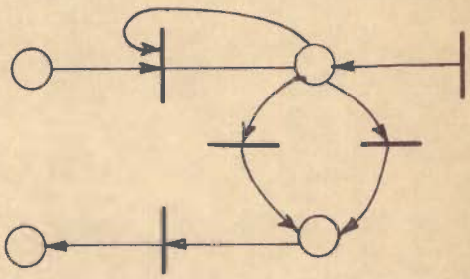
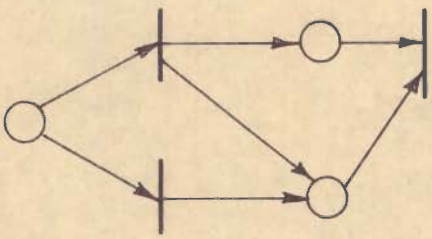
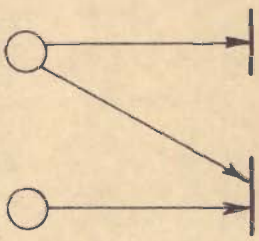
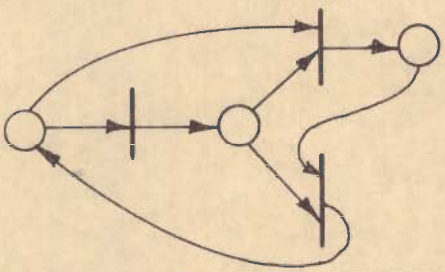
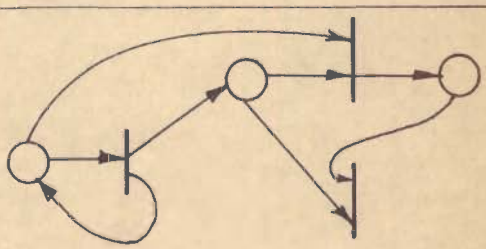
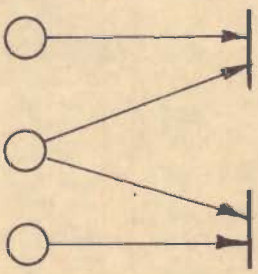
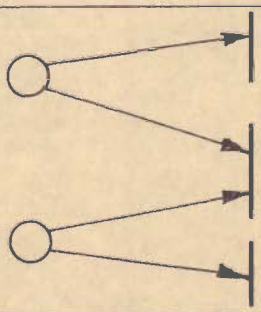
SUBCLASS	ALLOWED	NOT ALLOWED
MARKED GRAPH		
STATE MACHINES		
FREE CHOICE PETRINETS		
PURE PETRINETS		
SIMPLE PETRINETS		

FIG. 2.3 _ ALLOWED AND NOT ALLOWED STRUCTURAL CONFIGURATION OF VARIOUS SUBCLASSES OF PETRINETS

2.6 ANALYSIS OF PETRI NETS

To study the systems through Petri nets there are two approaches available in literature. One approach is aimed at deriving properties of Petri nets, and the properties of the system modeled. In the other approach [96] the design process is carried out directly in terms of Petri nets and the resultant PN is implemented straightway [40], [113], [116]. Both these approaches require that the knowledge about the Petri net itself be available. This has led to an extensive research in the theory of Petri nets [6], [7], [21], [27], [49], [60], [82]-[84], [97], [100], [102], [120], [137], [147], [148].

The objective of the analysis of PN is to determine certain properties. Some of these such as reachability, safeness, boundedness, conservation etc. have been discussed in Section 2.3. There are other important properties as well and those will be introduced as and when needed. Obviously the analysis technique must be such that it is easily implemented on computer to allow automatic analysis of modeled systems. With this view, two major available analysis techniques are discussed as follows:

2.6.1 THE REACHABILITY TREE

The reachability tree of a Petri net is a tree the nodes and arcs of which represent the reachable marking and the possible changes in state resulting from the firings of transitions [69], [70], respectively. As the reachability tree is finite [121], it is possible to have a finite representation of infinite

reachability set often encountered in Petri nets. If it is found that a transition adds a token at a place every time it fires, then the number of tokens in that place is represented by w which is 'too large' such that $w \pm x = w$, $x < w$ for any integer x. With these concepts, the reachability tree of PN, $N = (P, T, \alpha, \beta, M_o)$ where M_o is initial marking is constructed as follows:

Let the initial marking be the root node and tag it 'new'
WHILE new markings exist DO

Select a new marking M.

If M is identical to another node in the tree which is not new, then tag M to be old and stop processing M.

If no transition is enabled in M, tag M to be 'terminal'.

For every transition t enabled in M

(1) Obtain the marking M' which results from joinint t in M

(2) If there exists a path from the root to M containing a marking M" such that $M' > M"$, then replace M'(p) by w when $M'(p) > M"(p)$.

(3) Introduce M as a new node, draw an arc from M to M' labeled t, and tag M' to be 'new'.

As an example, the reachability tree of a marked Petri net (Fig. 2.4) is shown in Fig.2.5.

The reachability tree can be used as a useful analysis tool to determine some of the properties of Petri nets and thus can solve several problems. The following properties are decided using the reachability tree.

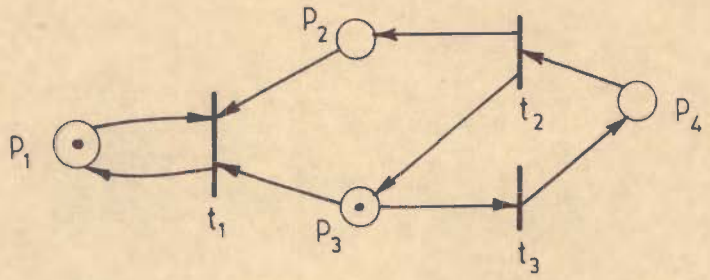


FIG. 2.4 _ A MARKED PN

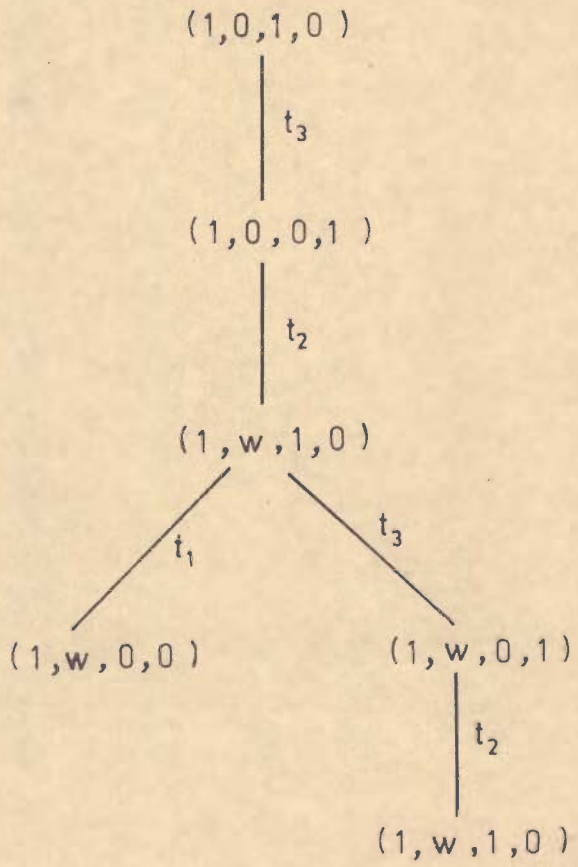


FIG. 2.5 _ THE REACHABILITY TREE OF PN IN FIG. 2.4

1. Safeness and Boundedness:

If a PN is k bounded, then by definition no more than k tokens are present in any of the places. Each place can have any number of tokens given by an element of the set $\{0,1,2,\ldots\ldots,k\}$ i.e. each place can have (k+1) possible tokens. Hence the number of possible reachable markings equal $(k+1)^n$ which is a finite reachable state-space. In order to determine the bound k on a particular place, the reachability tree is first generated. If w appears in the reachability tree, the net is unbounded because w is 'too large' and there exists a sequence of transition firings which can be repeated arbitrarily many times to increase the number of tokens to an arbitrarily unbounded number. If w does not appear, then the reachability tree is scanned for the largest value of the components of the markings corresponding to that place. If bound for all place is 1, then the net is safe.

2. Conservation:

If a PN is strictly conservative, then the number of tokens in each marking remains constant. Say, this number is k. Since there are finite number of ways to partition k tokens among n places, we have a finite reachability set. Thus strict conservation of tokens can be tested by computing sum of tokens in each marking. If sums are same then the net is strictly conservative. However, if w appears in the reachability tree, then the Petri net is not strictly conservative because w, though 'too large', is different for any two markings.

The generalized conservation is generally given with respect to a defined weighting vector or undefined weighting vector. In the first case the weights of each place is known. If w appears in reachability tree for a place, say p and the weight of p is nonzero positive, then the weighted sum of the tokens for two markings will be different in their w component. Thus the net will not be conservative. On the other hand if the weights of all the places for which w appears are zeros, then the net is conservative. For the case when there is no defined weighting vector, a Petri net is conservative if it is conservative with respect to some weithting vector W, with $w_p > 0$.

3. Coverability:

An important problem in Petri nets is coverability of markings which is useful in determining the occurence of variation in mutual exclusion in a system and in testing transitions for liveness and deadlock. This is defined as follows:

Definition 2.16 : Given a PN, $N = (P, T, \alpha, \beta, M)$ and a marking $M' \varepsilon R(M)$. The existence of another marking $M'' \varepsilon R(M)$ such that $M'' \geq M$ is called a coverability problem.

This problem can be solved by inspecting reachability tree [54], [69] and searching for a node x with marking $M_x \geq M'$. If no node is found, the marking M' is not covered by any reachabe marking. If such a node is found, this gives a reachable marking which covers M'. Karp and Miller [69] have proposed an algorithm to determine the minimal number of transition firings to cover a given marking.

2.6.1.1 LIMITATIONS OF REACHABILITY TREE

The reachability tree can not solve the reachability or liveness problem. Also this can not determine which firing sequences are possible. To solve these problems it is required to know the exact number of tokens on places in different markings. This information is lost in the symbol w. However, in some particular cases reachability or liveness may be solved. For example, a Petri net whose reachability tree has terminal node (one with no successors), is not live. Similarly a marking M' may appear in the reachability tree then M' is reachable from M. Also, if a marking is not covered by some node, then it is not reachable.

2.6.2 STATE EQUATION OF PETRI NETS

Another approach to the analysis of Petri nets is based on state equation [97] which is quite different from that of normal dynamic systems in that the behaviour of Petri nets is essentially characterized by the control vector of non-negative integers. Even then the state variable technique of system theory can be useful in studying Petri nets. Without loss of generality (as equivalence of generalized Petri nets and single-arc nets exists [50], Petri net is considered.

Since the marking changes as a result of firing, instead of M (definition 2.4) a |P|x1 column vector of nonnegative integers M_k is defined. The j th entry of M_k denotes the number of tokens on place j immediately after k th firing. Specifically M_o denotes the initial state. Out of many enabled transitions,

which transition fires is defined by a control vector V_k as a $|T|x1$ column vector containing exactly one nonzero entry 1 in the i th position if i th transition is fired at k th firing. The concurrent firing of more than one transitions is allowed and can be expressed as the sum of corresponding control vectors.

From the definition of firing, it is found that the state M_k resulting from another state M_{k-1} by k th firing V_k can be given in terms of following Murata's state equation:

$$M_k = M_{k-1} + A^T V_k, \quad k = 1,2,.... \tag{2.1}$$

where A^T is transpose of transition-to-place incidence matrix (definition 2.3). It is noted that the i th row of A represents the token changes in $|P|$ places when i th transition fires once. For marked graphs, A reduces to the incidence matrix of a diagraph.

Since M_k is a vector of nonnegative integers, V_k must satisfy

$$M_{k-1} + A^T V_k \geq 0 \quad \text{for each } k \tag{2.2}$$

Let there exists a firing sequence $\{V_1, V_2,....,V_n\}$ that transforms an initial marking M_o to M_n of a Petri net. Then the solution is given by addition of the n equations of (2.1) for $k = 1,2,....,n$.

$$M_n = M_o + A^T[\sum_{k=1}^{n} V_k]$$

which can be written as

$$A^T \Sigma = \Delta M \tag{2.3}$$

where $\Delta M = M_n - M_o$ and $\Sigma \overset{\Delta}{=} \sum_{k=1}^{n} V_k$ is a $|T|x1$ column vector of

nonnegative integers called by Firing Count Vector. The i th entry of Σ represents the number of times the transition i would fire in a firing sequence leading from M_o to M_n.

Based upon the above concept, the following have been obtained:

1. A necessary condition [97] that a Petri net is completely reachable i.e. any initial marking can reach any other marking is

$$\text{Rank } A = |P| \tag{2.4}$$

2. Given two markings M_o and M_n for a connected marked graph G, there exists a non-negative integral solution Σ for

$$A^T \Sigma = \Delta M$$

if and only if

$$B_f \Delta M = 0$$

where $\Delta M = M_n - M_o$ and B_f is a fundamental circuit matrix [102].

In most of the practical applications of Petri nets, the condition (2.4) is rarely satisfied. The rank of A is generally less than the number of places in a Petri net. In such cases, the PN is uncontrollable or not completely reachable. Let the rank of A be r. Then, A can always be partitioned in the following form:

$$A = \begin{array}{c} \overset{|P|-r \qquad r}{\longleftrightarrow \quad \longleftrightarrow} \\ \left[\begin{array}{cc} A_{11} & A_{12} \\ A_{21} & A_{22} \end{array}\right] \begin{array}{l} \updownarrow r \\ \updownarrow |T|-r \end{array} \end{array} \tag{2.5}$$

We can always find ($|P|-r$) x $|P|$ matrix B_f:

$$B_f = [I \vdots A_{11}^T (A_{12}^T)^{-1}] \tag{2.6}$$

where I is the identity matrix of order ($|P|-r$), such that

$$A B_f^T = 0$$

It may be noted that B_f corresponds to fundamental circuit matrix in case of marked graph.

Thus, eq. (2.2) is consistent if and only if

$$B_f \Delta M = 0 \tag{2.7}$$

If there exists a firing sequence which transforms M_o to M_n, then the corresponding firing count vector Σ must exist and (2.7) must hold. Therefore,

3. A condition (2.7) is necessary for the existence of a firing sequence which transforms an initial marking M_o to another marking M_n in a Petri net [97].

4. In a Petri net, a state N_o cannot reach another state M_n, if their difference is a linear combination of the rows of B_f [97] i.e., if

$$\Delta M = M_n - M_o = B_f^T V_n$$

where V_n is a nonzero ($|P|-r$) x 1 column vector.

Comments** : To test if a given initial state can not reach another given state, one has to calculate B_f. For this, first A is determined for PN. Then largest nonsingular submatrix A_{12} of A is obtained in the form as in (2.5). To find B_f, $(A_{11}^T)^{-1}$ is determined. This is quite involved process. Thus calculation of B_f requires a lot of computation. It has been shown in the Chapter IV that the nonreachability condition can be obtained in terms of A and ΔM. As B_f is not calculated, the testing of nonreachability becomes very simple.●

The state equation approach deals basically the reachability problem. It has great promise and can solve many problems of Petri nets. It must also be noted that a similar matrix approach has been taken in independent research in Europe [21], [29], [43], [83], [137]. The liveness problem and indirectly, the reachability problem which is reducible to liveness [56] has been tackled in [21], [43], [81]. While the reference [29] uses the concept of formal languages as main vehicle, some structural properties of Petri nets have been found in [137]. Some concepts not covered in state equation and proposed by these researches in matrix approach and by others, for example [5], [7], [100], etc. are given below. As the inclusion of these concepts does not change the nature of state equation, the combined approach will be called as the state equation approach.

** Denotes specific comments by the author.

● End of specific comments.

Of special interest for determining properties of Petri nets is the concept of Invariance. It is defined as under:

Definition 2.17 : A Petri net, $N = (P, T, \alpha, \beta) = (A, ——)$ is said to be invariant if there exists a $|P| \times 1$ vector I with all its components positive such that

$$AI = 0 \tag{2.8}$$

Using eq.(2.3) and (2.8), the following is obtained

$$(\Delta M)^T I = V_K^T (AI) = 0$$

or

$$M_{K-1}^T I = M_K^T I \tag{2.9}$$

Due to the invariant property expressed in (2.9), I is called an invariant of Petri net. However, a widely used concept [83] is that of a Simple Invariant.

Definition 2.18 : A simple Invariant is a set of places, I, such that $\sum_{p \in I} M(p)$ is a constant for each reachable marking M, and I does not have any proper subsets that are simple invariants.

From the definition, it is evident that the simple invariants are disjoint sets. From the set of simple invariants, some properties about the dynamic behaviour of the Petri net can be deduced [7], [83]. For example, the following properties are obtained:

1. Boundedness and safeness:

If each place is in some simple invariant and the Petri net has initially a bounded marking, the net is bounded. Same is true for safe nets. This is, obviously, so because the number of tokens

in a simple invariant is constant and all the places are covered by simple invariants.

2. Conservativeness:

If the set of places can be partitioned with disjoint subsets each of which is a simple invariant, the net is conservative and the total number of tokens in the net remains constant.

3. Mutual Exclusion:

If an input or output place of a transition t is contained in a simple invariant I, t is said to be a transition of I. If two transitions correspond to same simple invariant and the initial marking is such that the sum of tokens in the places of the invariant is 1, then the transitions are mutually exclusive and cannot fire simultaneously.

4. Liveness:

Under certain presuppositions, the simple invariants may be interpreted as Complete System of Circuits. When analysing liveness, so-called Variants are of interest, and these may be interpreted as incomplete systems of circuits [83].

In order to reduce the complexity in deciding the properties of Petri nets, the notions of boundedness and liveness have been defined so as to be independent of a given marking [137] but dependent only upon the structure of PN.

Definition 2.19 : A PN is Structurally Bounded if it is bounded for every marking and is Structurally Live if there exists a marking for which it is live.

The structural boundedness and liveness have been studied [137] with a view that it is generally easier to decide if a Petri net is bounded (respectively not live) for every marking than it is bounded (respectively not live) for a given marking. A concept which has been used apart from the net invariance is Consistency.

Definition 2.20 : A PN, N = (A,—) is Consistent if there exists a |T|x1 vector X with all its components positive such that

$$A^T X = 0 \qquad (2.10)$$

From the study of incidence matrix of pure Petri nets, the following properties have been obtained [137].

i. If a Petri net is bounded and live then it is consistent. The converse is also true.

ii. If a Petri net is invariant, it is bounded.

iii. If a PN is bounded and consistent, then it is invariant and for every marked PN (A,M) the reachability set R(M) is constituted of pairwise incomparable vectors. This permits to calculate an upper bound of the cardinality of its marking classes. For example, if (A,M) is safe [49], then $R(M) \leq \binom{n}{k}$ where $k = \frac{n}{2}$ if n is even and $k = \frac{n-1}{2}$ if n is odd.

iv. (a) If a PN is consistent and invariant, it is bounded and live.

(b) If a PN is invariant and nonconsistent then it is not live.

(c) If a PN is noninvariant and nonconsistent, then it is neither bounded nor live.

Definition 2.21 : Given a PN, $N = (P, T, \alpha, \beta)$, a Subnet of N is a PN, $N_1 = (P_1, T_1, \alpha_1, \beta_1)$ such that $P_1 \subseteq P$, $T_1 \subseteq T$ and α_1, β_1 are the restrictions of α, β on $P_1 X T_1$. The Union of two subnets $N_1 = (P_1, T_1, \alpha_1, \beta_1)$ and $N_2 = (P_2, T_2, \alpha_2, \beta_2)$ is a subnet $N_3 = (P_3, T_3, \alpha_3, \beta_3)$ with $P_3 = P_1 \bigcup P_2$ and $T_3 = T_1 \bigcup T_2$. Being given a PN, N and a set S of subnets N it is said that N is Covered by S or that S is a Decomposition of N if the union of elements of S is equal to N.

Definition 2.22 : Let $N = (P, T, \alpha, \beta) = (A, -)$ a PN. For $X_o \in \mathbb{N}^m$, the Support of X_o denoted by $S(X_o)$ is the subnet of N, $S(X_o) = (P_1, T_1, \alpha_1, \beta_1)$ with $T_1 = \{t_j \in T \mid X_{oj} \neq 0\}$ and $P_1 = {}^oT_1 \bigcup T_1^o$. Also for $I_o \in \mathbb{N}^m$, the Support of I_o, $S(I_o)$ is subnet of N, $S(I_o) = (P_2, T_2, \alpha_2, \beta_2)$ with $P_2 = \{p_j \in P \mid i_{oj} \neq 0\}$ and $T_2 = {}^oP_2 \bigcup P_2^o$.

v. Every consistent (invariant) PN, $N = (A, -)$ is decomposable into a set of elementary consistent (invariant) components.

Let X_o, $A^T X_o = 0$ such that $S(X_o) = N$. If N is not an elementary consistent component then there exists a consistent component N_1 contained in N and X_1, $A^T X_1 = 0$, such that $S(X_1) = N_1$. If $\lambda = \min_{x_{ij} \neq 0} \left\{ \frac{x_{oj}}{x_{ij}} \right\}$ and $X_2' = X_o - \lambda X_1$, then $X_2' \geq 0$ and it is possible to find a vector $X_2 = \mu X_2'$, $X_2 \geqslant 0$, having integer

components. Furthermore, $S(X_o) = S(X_1) \bigcup S(X_2)$. This method can be applied iteratively in order to decompose a consistent or invariant PN with elementary consistent or invariant components.

Comments: To decompose a consistent and invariant PN, first X_o is to be obtained from $A^T X_o = 0$ and then iterative method as discussed above is used. This has two drawbacks: (i) it is quite cumbersome to calculate integer-valued solutions of systems of linear equations, and (ii) it is not known 'apriori' how many elementary components can be obtained. These limitations have been removed and a technique has been proposed in Chapter IV, to find 'apriori' the number of elementary components and to decompose the net into elementary nets.●

The proofs of the above properties are scattered over [82]-[84], [137] and are not given here as they are not necessary for maintaining the readability of this thesis.

It has been seen here that state equation approach is very helpful in deciding many properties of Petri nets, but there are limitations of this approach as well.

2.6.2.1 LIMITATIONS OF STATE EQUATION APPROACH

i. The self-loops (i.e. transitions which have both inputs and outputs from the same place) cannot be represented in matrix A. This is a loss of information about the structure of Petri nets.

ii. A serious problem is that although a solution to eq.(2.3) is necessary for reachability, it is not sufficient. Thus,

the existence of spurious solutions (i.e. the solutions which do not correspond to possible transition sequence) is necessary to be detected.

iii. Another problem is the lack of sequencing information in the firing vector. Consider equation (2.3) which is as follows:

$$A^T \Sigma = \Delta M$$

Here the solution Σ merely gives which transition should fire and how many times in order that a marking M_o is reachable to another marking M_n but does not say in which sequence the transition must fire. The problem becomes even more complicated when the rank of A is not ($|T|-1$) which is, generally, the case. Let r be the rank of A, then eq.(2.3) is a set of ($|T|-r$) independent equations in $|T|$ unknowns. This will, obviously, lead to a set of solutions rather than a unique one. Hence not one but many sequences may exist. State equation approach does not provide any technique to find such firing sequences of transition.

iv. The Non-negative integer solution of (2.3) and for that matter, the determination of invariants and consistency, is quite involved process for larger Petri net. The invariants have been systematically obtained by following certain rules during the synthesis of Petri nets [2]. However, there is no alternative for the solution of eq.(2.3).

An attempt has been made to find the firing sequence of

PNs in Chapter IV and thus the limitations (ii) and (iii) have been removed to a fairly great extent.

2.7 CONCLUSION

The recent literature available on the use of Petri nets in design, analysis and synthesis of systems is an evidence of growing interest in this field. Many different aspects of various systems can be studied through PN. This is, because, it can easily exhibit parallelism and represent systems at different levels of abstraction. These concepts and capabilities of PN have been reviewed in this chapter to form a background for the investigation taken up in subsequent chapters.

A very limited research on the optimization consideration through Petri nets is available [63], [136]. If a Petri net exhibits a certain behaviour, as indicated by transition firing sequences and its reachability set, the question is : can a Petri net be optimized (changed) without affecting its behaviour ? This may involve deleting dead transitions or dead places or perhaps the redefinition of some transition and place. However, one has to be careful in defining the problem. This is, because, if equivalences are defined as equal reachability sets, then number of places cannot change and on the other hand if equality of sets of transition firing sequences is required, then transitions can not change.

In Chapter VI such considerations are dealt with reference to the important optimization problems in the design of microprogrammed computers.

CHAPTER III

CRITICAL REVIEW AND GENERAL CONSIDERATION OF MICROPROGRAM OPTIMIZATION

3.1 INTRODUCTION

Recent decades have seen with growing interest the development of microprogrammed computers. This is because microprogramming provides a systematic way of designing the control unit, increases the flexibility of a computer and makes it possible to execute directly the programs written in machine languages of a different computer - a process called emulation.

Microprogramming was conceived by Wilkes [151]-[153] as a technique for the implementation of control function in digital computers. Since then it is finding wide applications in modern day computers. One development has been the use of microprogramming in the economical implementation of control circuits of very larger computers like Illiac IV and Control Data STAR 100. Direct interpretation of high-level or intermediate languages and microprogramming of operating system functions are other possibilities. Additionally the development of dynamically user programmable computer has been made possible because of advances in hardware technology. This allows the replacement of operating system and language translators with inexpensive hardware, whose functions can be modified readily by microprogramming. In all these applications one way to increase efficiency is via

microprogram optimization.

3.2 BASIC CONCEPTS OF MICROPROGRAMMING

The concepts and terms involved in defining microprogramming are well established but are given here to avoid ambiguity and for the sake of completeness and continuity. A computing machine, exclusive of control, consists largely of registeres and combinational execution resources (adders, shifters etc.). The former comprises the local store and the latter, the functional units. The hardware interconnections between the functional units and storage resources are called Data Paths. The cycle time of a computer is the time required to change the information in a set of register via such data paths. A control logic is required to exercise an overall control of the various resources of a computer. Conventionally the control information has been permanently built into system by means of combinatorial and sequential logic network in an adhoc manner. This type of control is highly complex, because a need for the slightest modification to the instruction set could call for a major change in the entire control structure.

In microprogrammed computers Microinstructions control the operation of various resources and are defined as the basic machine functions sanctioned by the manufacturers. These are stored as words in a high-speed nondestructive read only control store generally called Read Only Memory (ROM) which is usually (but not always) separate from main memory. Through the ROM address and data registers, the computer hardware provides the

facility for systematically executing sequences of micro instructions.

A Microoperation is a unit of microprogram activity which performs a particular function (e.g. opening or closing gate, shift, increment, etc.). The subcommand to execute a microoperation is called Microcommand. A microinstruction is a specification of microoperations to take place in one cycle. A sequence of such microinstructions constitutes a machine instruction.

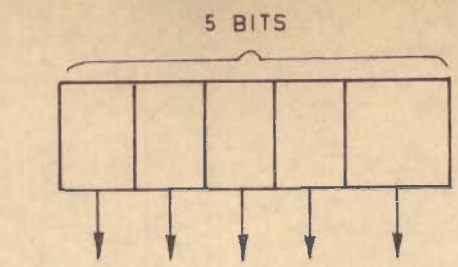
Two aspects of microinstructions i.e. design and implementation, need examination with reference to hardware characteristics. Design of microinstruction determines the information required to control hardware resources and its arrangement in a microinstruction word. Of primary interest is the number of resources controlled by each microinstruction. For this reason, microinstructions are commonly classified as vertical or horizontal. Vertical microinstruction refers to one type of operation and are characterized by short formats, limited ability to express parallel microoperations and considerable encoding of control information. Horizontal microinstructions, on the other hand,control many resources which operate in parallel. These have the attributes of long formats, ability to express a high degree of parallelism and little encoding of the control information.

The degree of encoding in a microinstruction word affects its length, and is often a parameter for the vertical and horizontal characteristics. In simplest design no encoding of bits

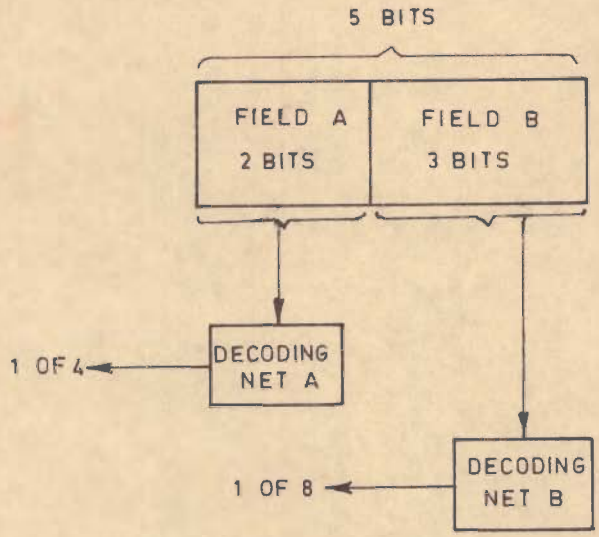
in microinstruction word is done. Each bit controls one resource or operation. This is shown in Fig.3.1(a). In single level encoding, one bit controls many resources and depending upon the parallelism, microcommands are grouped in different control fields (Fig. 3.1(b)). Another type of encoding which plays an important role in the bit optimization is two level encoding or bit steering [10]. Here, as shown in Fig.3.1(c), some bits are shared by different fields. The uncoded format has the advantage that the control signals may be derived directly from the microinstruction. When encoded fields are used, each control field must be connected to a decoder from which the control signals are derived.

The microinstruction implementation is important for execution time. In serial implementation, fetching of next microinstruction does not begin till the control terminates. This results in simplicity of realization because the hardware need not control execution and fetch simultaneously, and no problem arises in execution conditional branch instruction. On the other hand in the parallel implementation, fetch of next microinstruction is to be executed in parallel with the execution of microinstruction. Hence, there is saving in time. A combined serial-parallel implementation is generally adopted to have the simplicity of realization and some saving in time.

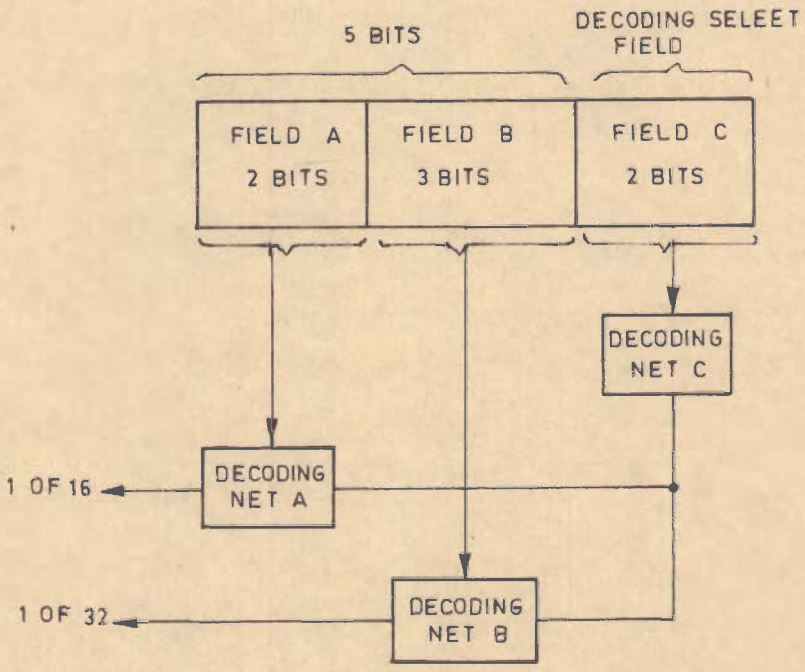
A control store is characterized by its word and bit dimensions. The Word Dimension Wd of the control store is the number of words of control storage required for a certain application. The Bit Dimension B represents the number of bits



(a) NO ENCODING



(b) SINGLE LEVEL ENCODING



(c) TWO LEVEL ENCODING OR BIT STEERING

FIG. 3.1 FIELD CONTROL FORMATS

per word of control store. Microprogram Optimization is performed to reduce/minimize the size of control memory (Wd x B) and/or reduce the execution time. The next paragraphs are devoted to strategies and techniques involved in such optimizations.

3.3 STRATEGIES OF OPTIMIZATION

The various strategies of microprogram optimization are classified [4] in four broad categories, namely, Word dimension reduction, bit dimension reduction, state reduction and heuristic reduction.

Word optimization [13], [25], [33], [36], [65], [78], [88], [129], [139], [141], [144], [146], [154] is performed in microinstruction generation phase or compilation. Here the deletion of non-essential microoperations, the identification of parallel microoperations and resource allocations are considered. However, even after spending substantial time in such optimization, there is no guarantee of significant savings in microcode space and execution time.

In bit optimization [17], [31], [32], [47], [57], [66], [99], [135], for given ROM specification or the instruction set specification, the microinstruction length is reduced by partitioning the microcommands into appropriate groups and encoding them.

In state optimization [45], [46] the microprogrammed computer is analysed in terms of two interacting finite state machines namely a control part and a functional part. State

reduction techniques [22], [81] can be applied to reduce the control part. However, this optimization is impractical for any modern computer with innumerable states.

Heuristic optimization methods [93], [94], [140] employ adhoc techniques to minimize the control part of the composite automata. This, however, does not guarantee a minimal control part because of its adhocism.

Of the four categories of optimization, bit optimization has practical advantages particularly for the design of spacial purpose microcomputers dedicated to single applications. It is this aspect that has been dealt in detail in this thesis.

3.3.1 BIT OPTIMIZATION

To minimize bit dimension, there are two type of approaches. One type of approach starts with the given ROM specification and the other with the given instruction set description.

Schwartz [135] has described a model for ROM as a rectangular array of binary storage elements consisting of Mc microinstructions of B bits each. Each microinstruction specifies one or more subcommands to perform elementary operations. Table 3.1 is a hypothetical ROM described by Schwartz.

Microinstruction	Microoperations
1	a,b,c,d,e,f
2	c,g,h,i
3	a,b,h,i,j
4	d,h,k
5	f,h

TABLE - 3.1

All the microcommands contained in a microinstruction are performed in parallel while microinstructions are executed sequentially. Since the sequencing of ROM words is not of any concern for the minimization, address fields are not considered. The problem of bit optimization for the given ROM specification, thus, addresses to the encoding of Nc microcommands in Mc instructions of the control memory. There are two ways of encoding the microcommands:

1. Every microcommand is encoded by one bit each. The advantage is maximum flexibility. Since no combinational circuit is required at the output of the ROM the contents of ROM can be arbitrarily changed. However, this method is inefficient because of the large number of bits needed due to usually large microoperations.
2. All the microcommands are encoded by a minimum $\lceil \log Nc \rceil$ * bits. In this case a complicated decoding network is a must. All advantages of microprogramming are lost because even a slight modification is difficult to attain.

In bit optimization, a compromise between the maximum flexibility and bit minimality is made. A minimum number of encoding bits is obtained such that the parallelism in micro commands is not lost. Following are given the different bit optimization techniques.

---

* Throughout this thesis, logarithm is to base 2 and $\lceil X \rceil$ is smallest integer greater than or equal to X.

3.3.1.1 SCHWARTZ's ALGORITHM

Schwartz [135] took a midway position and proposed partitioning of the microinstruction format into disjoint groups, the only constraint being that no two microoperations from the same microinstruction could be assigned to the same group. A solution is, thus, a collection of sets $S = \{S_1, S_2, \ldots, S_p\}$, where

Condition 1 : Every n_j ($j = 1,2,\ldots,Nc$) is contained in a (unique) set S_n.

Condition 2 : No pair n_{j1}, n_{j2} ($J1 = 1,2,\ldots,Nc$; $j2 = 1,2,\ldots,Nc$; $J1 \neq J2$) belonging to the same microinstruction m_i can be in the same set S_k.

Condition 3 : The number of sets S_p is minimal

3.3.1.2 GRASSELLI AND MONTANARI's ALGORITHM

Grasselli and Montanari [47] pointed out that a minimum group solution does not imply a minimum B solution. The minimum group solution of Schwartz for the ROM in Table 3.1 is (a), (b,g), (c,j,k), (d,i), (e,h), (f). The number of groups is 6 and B = 10. In the solution (a), (b), (c), (d,g,j), (e), (f,i,k), (h), suggested by Grasselli and Montanari, although number of groups is 7, the number of bits required to encode is reduced to 9. Thus, to obtain minimal B, the problem was reformulated by changing condition 3 to:

Condition 3' : The quantity

$$B = \sum_{k=1}^{p} \log \lceil (|S_{ik}|+1) \rceil$$

is minimal, where $|S_{ik}|$ denotes the cardinality of S_{ik}, and 1 is added to the cardinality of each set to include the no-microoperation-N_{op}.

Two microcommands n_{j1} and n_{j2} are defined to be Compatible if condition 2 is satisfied. A Compatible Class C_i of micro-operations is a class whose members are pair-wise compatible. A Maximal Compatible Class (MCC) is one to which no microoperation can be added without violating pairwise compatibility. Then a minimal solution is a set of compatibility classes : $(C_{i1}, C_{i2}, \ldots, C_{ih})$ such that $B = \sum_{k=1}^{h} \lceil \log(|C_{ik}| + 1) \rceil$ and $\bigcup_{k=1}^{h} C_{ik}$ is set of microoperation in ROM. Grasselli and Montanari show that the only classes that need be considered for a minimal solution are Prime Compatible Classes as defined below:

1. C_i is nonmaximal and $|C_i| = 2^h - 1$ $(h = 1,2,\ldots)$; and

2. C_i is maximal and $|C_i| \neq 2^k$ $(k = 1,2,\ldots..)$

The minimal cover is then obtained by solving a covering table of the prime implicant type.

3.3.1.3 LINEAR PROGRAMMING METHOD

Jayasri and Basu [66] applied linear programming technique to solve this problem by minimizing the following cost function:

$$C = \sum_{i} \lceil \log(i+1) \rceil a_i$$

where a_i is the number of mutually exclusive classes containing i microcommands. The value of i ranges upto a maximum, which is the number of microcommands in the largest MCC. The algorithm

also takes advantage of some special features of cover table.

3.3.1.4 CM COVER TABLE METHOD

Das et al.[31] start with the same basic formulation as Grasselli and Montanari. However, they start directly with the maximal compatible classes whose number is usually small. The basic procedure is as follows : From a CM cover table with rows and columns representing MCCs and microoperations respectively, the irredundant solutions are obtained. For each of which a solution CM table similar to CM cover table is constructed. This indicates which are locally essential MCCs and which micro-commands are to be covered by them. The different covers for remaining microcommands are obtained through the construction of a reduced solution table. Going through all such possible coverings, the minimum solution is obtained.

3.3.1.5 BRANCH AND BOUND METHOD

Recently branch and bound method has been applied by Baer and Koyama [17] to solve the problem of bit optimization. They modeled the problem in graphical terms. Starting from root node with as many singleton sets as the number of microcommands in largest microinstruction, a partial solution represented by a node is continually extended until a solution satisfying a cost constraint is reached. Their algorithm can best be described in terms of pseudo Pascal notation as follows:

Step 1 : [Initialize] B $\leftarrow$ Nc, I $\leftarrow$ 0

Step 2 : [Obtain initial solution, then back up and proceed a new, if necessary]

While $I \geq 0$ do

If $(LB_I \geq B)$ or if all possible placements have already been attempted

Then $I \leftarrow I-1$

(i.e. back up since no better solution can be obtained for this subtree).

Else repeat

$I \leftarrow I+1$; Select new microoperations to be e encoded;

Place in same S_j ; Compute LB_I

Until $(I = Nc-m)$ or $(LB_I \geq B)$

If $I = Nc-m$ then $B \leftarrow \sum_{j=1}^{p} \lceil \log|S_j|+1 \rceil$

$B \leftarrow \min(B, B)$; $I \leftarrow Nc-m-1$

3.3.1.6 MONTANGERO's ALGORITHM

As suggested by Grasselli and Montanari, a better and/or flexible solution for bit optimization could be obtained from the specification of the instruction sets rather than from ROM. The problem is then to encode the instruction set by finding amongst all possible ROM descriptions, the appropriate ROM having minimum value of B x Wd. This is the approach adapted by Montangero [96]. He describes the instruction set by a forest F of directed acyclic graph. A graph G_i represents an instruction i. The nodes of G_i represent the microoperations and an arc represents a temporal precedence relation between the micro-operation. A level assignment to a graph G is a partition of the

nodes of G into different levels such that if there is an arc (m,n), then the level of m is less than the level of n. A level assignment to each graph of a forest F constitute a level assignment of F. All possible level assignments of F can be obtained. The scheme of Gresselli and Montanari can be applied to each assignment to select the one with minimal B x Wd. But Montangero prefers a heuristic procedure to avoid an exhaustive search by considering only a subset F consisting of prime graphs. The algorithm first finds actual costs with minimum estimates. Then all assignments whose optimistic estimate is not lower than the actual cost, are eliminated. It is repeated until all assignments have been considered.

Comments : It is observed that the methods referenced yields optimal solution after tedious and time expensive procedures. Schwartz's algorithm is basically one of the exhaustive evaluation and does not necessarily result in minimal bit solution. The technique of Gresselli and Montanari is an improvement over Schwartz's algorithm. But it suffers from the drawback that the cover table has usually very large number of rows and the covering is cyclic in nature. To find a minimal solution is, therefore, a difficult task.

The linear programming method requires judicious selection of mutually exclusive groups of microcommands in order to have less processes of trial and error. Although this provides a good starting point, the computations involved are large.

The essential approach in Das et al. is to solve a number of small cover tables rather than a big one. It is, however, not apparent if the overall effort is less than that required in Grasselli and Montanari.

The technique of Baer and Koyama appears more effective because it leads quickly to near optimal solution. However, it is time expensive if all minimal solutions are needed.

One of the major problems in Montangero [96] is that no method is given to detect prime graphs for a general case. Further, a solution is provided if every microcoperation occurs atmost once in any graph of original forest. But this drawback can be removed by renaming microoperations appearing more than once so that they are considered to be different in beginning. This results in additional computational efforts.

It is not surprising in the light of recent work [132] that all these methods require computations which are exponential in nature. Here, Roberston has shown that the problem of bit optimization is a NP-complete and hence "one should not attempt to solve for general case but concentrate on heuristics for reasonable subcases". o

Barring Schwartz's technique, almost all the methods require the generation of MCCs. These generations are usually done with the help of compatibility chart and graph the construction of

which prevents its adaptability on computers. A method employing state equation of Petri nets to generate all MCC's is given in Chapter VI. A method of bit optimization which requires fewer enumerations for a 'reasonable subcases' is also presented.

An important aspect of bit optimization is a good 'engineering reduction' [4]. If ROMs are available in 4 bit configurations, then the efforts required to reduce the bits from 12 to 9 is a sheer wastage and the bit reduction is not a 'good reduction'. It is because the number of chips needed will be 3 for 12 as well as for 9 bits. If however, one more bit is reduced, the number of chips needed will be 2 and there will be considerable saving in the control memory and hence cost. To do this the concept of two level encoding or bit steerability [10] discussed in Section 3.2 is used. Bit steerability reduces the number of optimized bits.

We can inspect control fields with multiple bits, whether a part of the field can be shared by two or more such control fields. If it is possible, the further reduction in bit dimensions can be made. The shared bits are called steering bits and the remaining bits of control fields are steered fields. If shared field consists of more than one bit, then a decoder (Fig.3.1c) is needed. It is obvious that the maximum number of bits in steering field is one less the number of bits in the smallest amongst original fields considered for steering.

3.3.1.7 BIT STEERING IN BIT REDUCTION

The only work available in literature on the bit reduction through bit steering is by Mathialagan and Biswas [86]. Given the ROM specification and all the optimal bit solutions, their algorithm for detection of bit steering in two fields and encoding them involves the following steps.

1. Setting up a concurrency matrix. The columns and rows of concurrency matrix correspond to microcommands (including (Nop) in the two fields, respectively. The entry 'X' at the intersection of a row i and column j is present only when microcommands i and j are present together in every microinstruction.

2. Grouping of entries in rows and columns if there is an entry at the intersection of rows and columns.

3. Testing the following conditions:

   (a) all entries in the concurrency matrix are accommodated in 2^q or less groups when q is number of steering bits and $1 \leq q \leq [\min(n_1, n_2)-1]$, where n_1 and n_2 are the number of bits needed to encode the two fields respectively.

   (b) each group includes not more than $2^{(n_1-1)}$ and $2^{(n_2-1)}$ microcommands.

   When these two conditions are satisfied only then bit steering is possible.

4. Encoding the steerable sets by assigning the same steering code to microcommands of a group, considering groups one

after another and then assigning unused code combination to microcommands not covered.

The detection of n set steerability is done on the same lines. But as the construction of n dimensional concurrency matrix is not possible, the following conditions must be satisfied for the sets of microcommands C_i ($i = 1,2,\ldots,n$) for $n \geq 3$ to be steerable.

1. The sets must be pairwise steerable
2. The disjoint sets with distinct steering code of each set C_i are the same when the set C_i is steered with each of the remaining (n-1) sets.

Comments : Although the algorithm for bit optimization through bit steerability is very elegant, it suffers from two major drawbacks. Firstly, the formation of groups in concurrency matrix is done by the intersection of row and columns sets such that the group contains maximum number of entries. This type of formation of a group is not easily adapted on computers. Secondly, the method becomes too cumbersome for detecting steerability among 3 or more control fields. This requires the testing of pairwise steerability. When n control fields are considered for steerability, the number of concurrency matrix needed is $\frac{n(n-1)}{2}$. Therefore, total enumerations required is $\frac{n(n-1)}{2}$ times the enumeration in one concurrency matrix. But as pairwise steerability is a necessary and not sufficient condition,

even after such large enumerations it requires a part of encoding procedure and detection of disjoint sets of all the fields appearing in different concurrency matrices. The disjoint subset of a field C_i with distinct steering code are, then, to be tested for no conflict. These are quite involved processes. To overcome these difficulties, an extended Petri net approach is presented in Chapter VI.

3.4 DATA PATH OPTIMIZATION

The number of control signals required for the control store design is partially decided by the data path assignments, as the control signals are applied to the resources through the respective interfaces. Therefore, minimal control signals will be needed only when data path assignment is made with minimal interfaces — an optimal data path assignment. Thus, optimal data path assignment is a very important aspect and a prerequisite for obtaining minimal control store through bit optimization techniques.

With the advent of integrated circuits, the digital system design is now required to solve the problem of selecting the appropriate modules and their interconnections. These modules constitute a relatively fixed fraction of the total chip area, while interconnections occupy a dominant share. Further, the number of bits in control memory is decided by the Parallelism in microoperations which is essential to increase the execution

speed of a microprogrammed processor (Section 3.3). These two factors are, therefore, very important as far as cost of the system is concerned. This calls for the data path optimization and control bit optimization of a system involving micro-programming.

3.4.1 INTERCONNECTION BUSES

The modules, as a rule, are connected to a bus (a set of interconnectors) through same interfacing circuitry. The output of a module is placed on a bus by a driver circuit and the input of a module receives data through a receiver circuit. There are two ways of interconnecting the modules. When modules are connected through dedicated bus, all possible data transfers among the modules are allowed. However, in general, all possible data paths between modules are not needed. Therefore, to save wiring it is a common practice to 'hang' several modules on a group of wire called shared bus or for simplicity, bus. Obviously, this will require lesser number of interfaces compared to dedicated bus. If each transfer of data takes place at distinctively different times, then the input and output ports of all the modules can be connected to a single bus. But parallelism in data transfers is an essential requirement to increase the speed of operation. Hence, a single bus cannot be used for more than one data transfer concurrently except when the source is the same in all cases. Also, there cannot be concurrent data transfers to the same sink from different sources unless the sink has more than one input port. Such concurrent transfers, cannot take

place through the same bus. Thus, a compromise should be struck between the costly dedicated bus scheme with faster speed of operation and the cheaper single bus scheme with slow operation speed. This is obtained by interconnecting modules to buses that allow concurrent data transfers with minimal interfaces called a data path optimization problem. Available in literature are only two algorithms for solving this problem. These are as follows:

3.4.2 DYNAMIC PROGRAMMING APPROACH

Torng and Wilhelm have solved the data path optimization problem using dynamic programming approach [145]. Following are the steps needed in their technique:

1. Setting up the transfer matrix to represent data transfers, determining the column and row sets (i.e. the set of connection variables appearing in rows and columns respectively) and constraint sets (i.e. the set of data transfer with different sources which cannot take place through the same bus).
2. Constructing the connection graph by connecting any two nodes by an edge if they appear in the same row or column set.
3. Generating the solution table starting from a minimum group of row/column set (i.e. the set containing fewest number of connection variables).
4. Extracting the minimum cost solutions, and
5. Repeating steps 3, and 4 with increased number of buses

to examine the possibility of further minimization.

Torng and Wilhelm claimed that the number of enumerations required for the generation of solution table is additive and, thus

$$NG_{en} = N_B^{|G_1|} + N_B^{|G_2|} + \ldots\ldots + N_B^{|G_k|}$$

where,

N_B = the number of buses

$|G_i|$ = the number of connection variables contained in a minimum group G_i

and, $N_B^{|G_i|}$ = the maximum number of enumerations for each minimum group G_i.

During the solution table generation, the cost is calculated accumulatively. The enumerations, thus, become multiplicative between two consecutive minimum groups. If G_1 is the first minimum group and G_k is the last minimum group, then the enumerations required to generate solution table is not one given above but as

$$N_{Gen} = N_B^{|G_1|} + N_B^{|G_1|} \times N_B^{|G_2|} + N_B^{|G_2|} \times N_B^{|G_3|} + \ldots\ldots$$

$$\ldots\ldots + N_B^{|G_{k-1}|} \times N_B^{|G_k|}$$

The worst case (when all the entries in G_k have the same cost) enumerations for the extraction of minimal cost solutions and for all minimal solutions, are given by

$$N_{Ext} = N_B^{|G_1|} \times N_B^{|G_2|} \ldots\ldots \times N_B^{|G_k|}$$

Of course, many of them will be abandoned before completion due to constraint or contradiction with the initialized cost.

Further, all the constraints are not considered in the process of solution table generation, an obtained minimum cost may or may not be feasible. In case the minimum cost is not feasible, none of the solution of the particular enumerations N_{Ext} gives the initialized minimum cost. The assigned cost is, then, incremented by 1 and the entire enumeration is carried out again. Thus, the algorithm becomes much more laborious. Some of these defects were removed in the following approach.

3.4.3 SWITCHING THEORETIC APPROACH

Mathialagam and Biswas [85] recast the data path optimization problem into a minimal covering problem following a switching theoretic approach. They solved the problem in the following steps:

1. Setting up transfer matrix with rows and columns as the intercomnecting modules, and entries by variables representing data transfers.
2. Obtaining a reduced transfer matrix, if possible, by first deleting isolated transfer variables and then removing any concurrency identifier (an integer attributed to simultaneous data transfers) appearing only once.

3. Determining -

(a) Incompatible simultaneous transfer set (ISTs) - the set of all concurrent transfers with different sources at the same time.

(b) Bus compatibles (BCs) - the set of data transfers with the same source or sink.

(c) Concurrent compatibles (CCs) - the largest subset of BCs containing only concurrent data transfer variables.

(d) Nonconcurrent compatibles (NCs) - largest subsets of BCs without concurrent data transfer variables,

and (e) NB_{min}.

4. Obtaining minimal covers, if possible, with as few CC's as NB_{min}.

5. Determining maximal concurrent compatible sets (MCCs) i.e. the set of maximum number of concurrent data transfer variables which can take place on a bus, and finding minimal cover with as few MCC's as N_{Bmin}.

6. Extracting the solutions from minimal covers by including NC's and isolated elements.

and

7. Minimizing the cost of the solutions by increasing the number of buses, if possible

Comments : Mathialgan and Biswas have shown that their algorithm is better than that of Torng and Wilhelm in the sense that lesser enumerations are required. This is because first feasibility is considered and then minimal

cost solution is obtained. This also provides all the solutions simultaneously. The comments regarding its enumeration are as follows:

The steps 1 and 2 are easily executed. The determination of minimal covers will require $\binom{N_c}{N_{B_{min}}}$ number of enumerations in step 3 and $\binom{N_M}{N_{Bmin}}$ number of enumerations in step 4 when N_C and N_B are the number of CC's and MCC's respectively.

The determination of MCC's (step 4) is done first by constructing a compatability chart which requires $\frac{n(n-1)}{2}$ comparisons where n is the number of concurrent transfers. Then compatability graph is constructed to find complete polygon corresponding to each of (n-1) concurrent transfers and from these compatible classes not included in other compatable classes are obtained as MCC's. This requires $\frac{n_c(n_c-1)}{2}$ number of enumerations where n_c is the number of compatable classes. Generally n_c is much larger than n. Apart from a large and heavily data dependent number of enumerations, the construction of compatability chart and graph is a must. This is not easily adaptable on digital computers.

The extraction of the solution from minimal covers will require atmost $(\#m_c)$ $N_{Bmin}^{(\#NC)}$ number of enumerations where $\#m_c$ and $\#$ NC's are the number of minimal covers and number of sets of NC's respectively. The actual number of iterations will be lesser than this because of constraints in IST's and CC's. However, this

will still be large.

It has, thus, been observed that the major drawbacks in the algorithm are (i) the construction of compatibility chart and the graph which is not easily amenable on computers and (ii) the generation of all feasible solutions. Some of the feasible solutions will not at all contribute to minimal solution. This is because the minimal cover does not, necessarily, contain CC's but may cover a subset of them. In Chapter VI, these drawbacks are shown to be removed by reformulating the problem of data path optimization in the domain of Petri nets. This has been motivated by the natural representation of concurrency by a PN and its elegance.o

3.5 CONCLUSION

It is concluded that the control memory bit and data path optimizations are two important aspects of microprogrammed computers. The latter is a prerequisite of the former. The suggestions of Schwartz and the formulation of Grasselli and Montanari with modified compatibility relation are the building blocks for the development of techniques available in the control memory bit optimization. Recently Roberston came to a conclusion that emphasis should only be made on heuristics. The bit steering technique is found to be useful in further reduction in the number of bits. As far as data path optimization is concerned, the method suggested by Mathialagan and Biswas is better than

that of Torng and Wilhelm in that it is easier and requires lesser computations. However, this too suffers from some limitations. These two optimizations have been again taken up in Chapter VI with a view to propose easier and efficient algorithms for them, using Petri net.

CHAPTER IV

ON THE DEVELOPMENT OF PETRI NET THEORY

4.1 INTRODUCTION

With a view to provide for efficient design procedure and improved performance of systems, considerable work has been carried out in theoretical aspects of Petri nets. A reference about this has already been given in Section 2.6. Many unsolved outstanding problems in PN theory were also pointed out. This chapter investigates some of those problems and suggests solution for them.

One of the problems often encountered is unmanageability of large Petri nets. It is not the disadvantage of Petri nets as such because it reflects the complexity of system being modeled. However, it poses some difficulties in analysis. Small Petri nets can easily be analysed. With this view, a large net is decomposed into smaller nets and is studied through interconnection properties of such nets. This is taken up in Section 4.2. A subsidiary motivation of this study is because of the advantages of PN representation in combination of state machines. As pointed out in Section 2.4.1, while the combination of state machines is complex, the PN representation is simply the cascade or parallel connection.

Another problem is that of reachability. It is important because many properties of PN can be derived in terms of

reachability. Although it is decidable, the solution is difficult to obtain due to lack of a sufficient condition. In Section 4.3 a solution to this is proposed by finding a legal firing sequence.

Petri nets as such have limited modeling capability (Section 1.3). To improve this a concept of negation has been introduced. This makes the available analysis techniques inadequate. This problem is solved in Section 4.4. A new type of transition to perform NOT operation is defined. These transitions alongwith conventional transitions and places of PNs can model any system. For the analysis, a generalized state equation is proposed. The prime justification for the state equation approach is due to its elegance and well developed concepts.

4.2 INTERCONNECTION AND DECOMPOSITION OF PETRI NETS

It is a well-established fact that PN model allows a formal specification of many concurrent systems [5], [120]. It is not only a mathematical tool for the analysis of systems but can also be used to prove the 'correctness' of the concurrent process system through discrete state equation or linear algebra [83], [97]. Although the procedure is very elegent and powerful, it is quite involved to calculate integer-valued solutions of systems of linear equations for larger Petri nets particularly those with multiple-arc connections. Therefore, procedures for analysis and proving 'correctness' become , in general, cumbersome and even impractical for deciding many properties. However, it is possible to split PNs into smaller nets each of which can be analysed and correctness verified easily. From the

interconnection properties of such nets, the analysis and verification of larger nets can be made. It is this study of decomposition and interconnections of nets which has been taken up in this section. Two properties of PNs, namely, invariance and consistency have been considered. This is because invariance and consistency together can verify many properties like boundedness, liveness, deadlock etc. Their relationships have been discussed in detail in Section 2.6.2. Further, the study in this section considers pure PN (defined in 2.5.4 as a PN where no place is both input and output to a transition). Before the interconnection and decomposition of PNs are considered, a few definitions given below are necessary.

Definition 4.1 : The elements a_{ij} of the incidence matrix A of a multiple-arc PN is given by

$$a_{ij} = a^{+}_{ij} - a^{-}_{ij}$$

where $a^{+}_{ij}(a^{-}_{ij})$ is the number of output (input) arcs from (to) transition i to (from) place j. For single arc PNs, $a^{+}_{ij}(a^{-}_{ij})$ is 0 or 1.

Let there be two PNs, $N_1 = (P_1, T_1, \alpha_1, \beta_1) = (A_1, —)$ and $N_2 = (P_2, T_2, \alpha_2, \beta_2) = (A_2, —)$. Then,

Definition 4.2 : N_1 and N_2 are said to be in cascade through a set of places P_{12} (a set of transitions T_{12}) if

(a) $P_{12} = P_1 \bigcap P_2$ $(T_{12} = T_1 \bigcap T_2)$

(b) P_{12} (T_{12}) is set of places (transitions) input to one of N_1 and N_2 and output of the other,

where $P_1 = P_{11} \bigcup P_{12}$ and $P_2 = P_{22} \bigcup P_{12}$

(c) N is an invariant if and only if N_1 and N_2 are invariant.

Proof : Let $P_{11} \subset P_1$ and $P_{22} \subset P_2$ be set of places contained only in N_1 and N_2, respectively. Then A_1 and A_2 can be written as:

$$A_1 = T_1 \begin{bmatrix} P_{11} & P_{12} \\ A_{11} & A_{12} \end{bmatrix} \text{ and } A_2 = T_2 \begin{bmatrix} P_{12} & P_{22} \\ A_{21} & A_{22} \end{bmatrix}$$

(a) Suppose N_1 is consistent. Then (from definition 2.20) there exists a $|T_1| \times 1$ column vector X_1 with all positive integers such that

$$A_1^T X_1 = 0$$

or $$A_{11}^T X_1 = 0 \qquad (4.1)$$

and, $$A_{12}^T X_1 = 0 \qquad (4.2)$$

Equation (4.2) will satisfy if and only if A_{12} has both positive and negative elements. This means P_{12} will have some input places and some output places. This contradicts condition (b) of cascading through places (definition 4.2). Hence N_1 can not be consistent if it is cascadable. Similarly it can be shown for N_2.

(b) A is a direct result of definitions (4.1) and (4.2)

(c) Let, $I = [I_{11} \quad I_{12} \quad I_{22}]^T$ with all positive elements.

N is invariant (definition 2.17) if $AI = 0$

i.e., $$A_{11} I_{11} + A_{12} I_{12} = 0 \qquad (4.3)$$

$$\text{and, } A_{21}\ I_{12} + A_{22}\ I_{22} = 0 \qquad (4.4)$$

As (4.3) and (4.4) are the conditions of invariance of N_1 and N_2, respectively, hence the proposition 4.1(c).

Proposition 4.2 : Let N be the resulting net on cascading N_1 and N_2 through a set of transitions T_{12}. Then

(a) A necessary condition that N_1 and N_2 are cascaded is that both N_1 and N_2 are not invariant.

(b) The incidence matrix A of resulting net N is given by

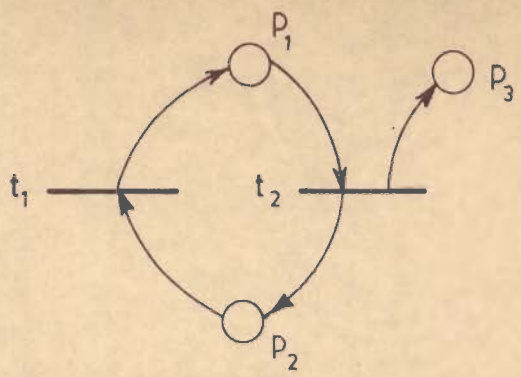
$$A = \begin{array}{c} \\ T_{11} \\ T_{12} \\ \\ T_{22} \end{array} \begin{array}{c} \begin{array}{cc} P_1 & P_2 \end{array} \\ \left[\begin{array}{c|c} & 0 \\ \hline A_1 & \\ \hline 0 & A_2 \end{array}\right] \end{array}$$

(c) N is consistent if and only if N_1 and N_2 are consistent.

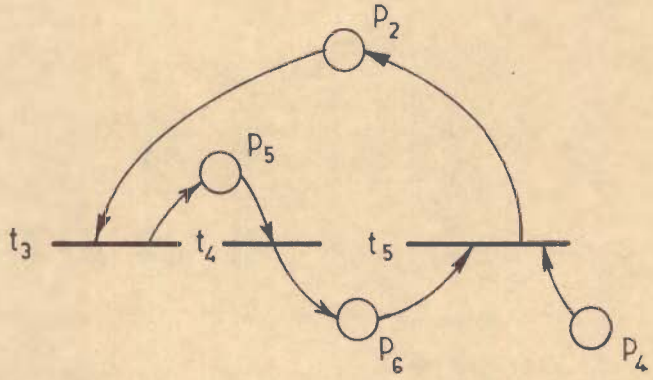
Remark : Proposition 4.1a (4.2a) suggests that even if N_1 and N_2 are not consistent (invariant) the resulting net can be made, in some cases, consistent (invariant) by merging a set of places (transitions). As an example, N_1 (Fig.4.1a) and N_2 (Fig.4.1b) are not consistent but a resulting net N (Fig.4.1c) obtained by merging-in of places p_3 and p_4 is consistent.

Proposition 4.3 : Let N be a resulting net when N_1 and N_2 are in parallel through a set of places P_c. Then

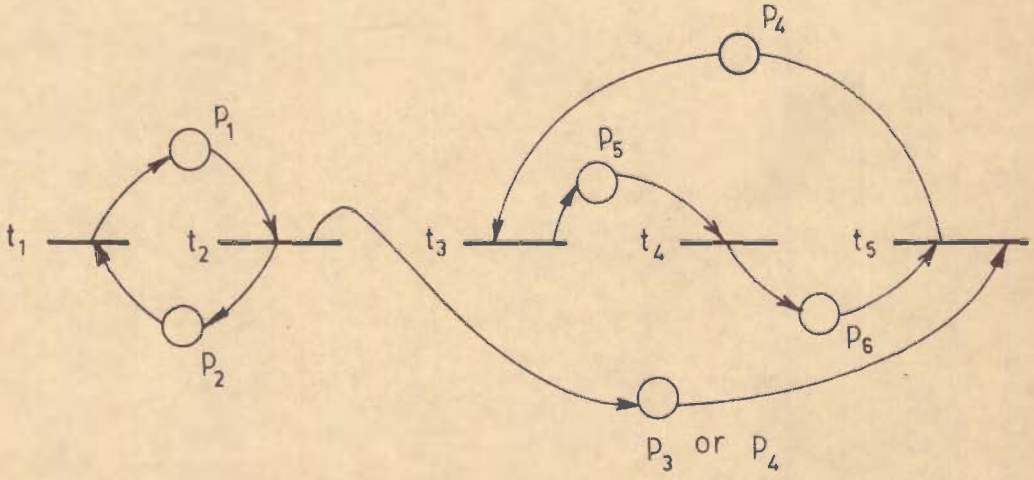
(a) A necessary condition that N_1 and N_2 are connected in



(a) A NONCONSISTENT PN, N_1



(b) A NONCONSISTENT PN, N_2



(c) A CONSISTENT PN OBTAINED BY MERGING-IN OF PLACES P_3 AND P_4 OF N_1 AND N_2, RESPECTIVELY

FIG. 4.1 _ FORMATION OF A CONSISTENT PN FROM NONCONSISTENT PNs

parallel is that both are not consistent.

(b) The resulting net N has

$$A = \begin{array}{c} \\ T_1 \\ T_2 \end{array} \begin{array}{c} \begin{array}{ccc} P_c & P_{11} & P_{22} \end{array} \\ \begin{bmatrix} A_{c1} & A_{11} & 0 \\ A_{c2} & 0 & A_{22} \end{bmatrix} \end{array}$$

where $A_1 = T_1 \begin{array}{c} \begin{array}{cc} P_c & P_{11} \end{array} \\ [\, A_{c_1} \quad A_{11} \,] \end{array}$ and $A_2 = T_2 \begin{array}{c} \begin{array}{cc} P_c & P_{22} \end{array} \\ [\, A_{c_2} \quad A_{22} \,] \end{array}$

(c) N is invariant if and only if N_1 and N_2 are invariant.

<u>Proposition 4.4</u> : When N_1 and N_2 are in parallel through a set of transition T_c, then

(a) The necessary condition that N_1 and N_2 are connected in parallel is that both are not invariant.

(b) The resulting net N has

$$A = \begin{array}{c} \\ T_{11} \\ T_c \\ T_{22} \end{array} \begin{array}{c} \begin{array}{cc} P_1 & P_2 \end{array} \\ \begin{bmatrix} A_{11} & 0 \\ A_{c1} & A_{c2} \\ 0 & A_{22} \end{bmatrix} \end{array}$$

where $A_1 = \begin{array}{c} \\ T_{11} \\ T_c \end{array} \begin{array}{c} \begin{array}{c} P_1 \end{array} \\ \begin{bmatrix} A_{11} \\ A_{c_1} \end{bmatrix} \end{array}$, and $A_2 = \begin{array}{c} \\ T_{22} \\ T_c \end{array} \begin{array}{c} \begin{array}{c} P_2 \end{array} \\ \begin{bmatrix} A_{22} \\ A_{c_2} \end{bmatrix} \end{array}$

(c) The resulting net N is consistent if and only if N_1 and N_2 are consistent.

The proofs of propositions 4.2, 4.3 and 4.4 are similar to that of proposition 4.1.

Let there be n subnets $N_1, N_2, \ldots, N_n$. Every pair N_i and N_j is connected through a set of transitions T_{ij} and a set of places P_{ij} to form a resultant N. Let in addition to conditions of definition 4.4 of connectivity the following also hold good

$$ {}^{o}P_{kk}^{o} \cap T_{ij} = P_{ij} \cap {}^{o}T_{kk}^{o} = \emptyset \quad \text{for } i,j \neq k $$

i.e. a set of transitions (places) common to N_i and N_j are not connected to set of places (transitions) contained only in a subnet N_k. Then $N_1, N_2, \ldots, N_n$ are <u>Decomposition</u> of N.

<u>Proposition 4.5</u> : The resulting net N is consistent (invariant) if and only if each of the subnets is consistent (invariant).

<u>Proof</u> : Let the incidence matrix A_i of a subnet N_i be given as:

$$ A_i = \begin{array}{c} \\ T_{i1} \\ T_{i2} \\ \vdots \\ T_{i(n-1)} \\ T_{in} \end{array} \begin{array}{c} \begin{array}{cccc} P_{i1} & P_{i2} \cdots\cdots & P_{i(n-1)} & P_{in} \end{array} \\ \begin{bmatrix} A_{11}^{(i)} & A_{12}^{(i)} \cdots\cdots & A_{1(n-1)}^{(i)} & A_{1n}^{(i)} \\ A_{21}^{(i)} & A_{22}^{(i)} \cdots\cdots & A_{2(n-1)}^{(i)} & A_{2n}^{(i)} \\ \vdots & & & \\ A_{(n-1)1}^{(i)} & A_{(n-1)2}^{(i)} \cdots & A_{(n-1)(n-1)}^{(i)} & A_{(n-1)n}^{(i)} \\ A_{n1}^{(i)} & A_{n2}^{(i)} \cdots\cdots & A_{n(n-1)}^{(i)} & A_{nn}^{(i)} \end{bmatrix} \end{array} $$

Since the transitions of T_{ij} are not connected to places of P_{ij} (def.4.4b), the submatrix $A_{ij} = 0$ for $i \neq j$ in A_i. Hence number of nonzero submatrices corresponding to row T_{ii} or column $P_{ii} = n$

and number of nonzero submatrices for row T_{ij} or column P_{ij}=(n-1) for $i \neq j$.

It can be shown that N_i is consistent if

$$\sum_{k=1}^{n} A_{ki}^{(i)T} X_{ik} = 0 \qquad (4.5)$$

and, $$\sum_{k=1}^{n} A_{kj}^{(i)T} X_{ik} = 0 \quad \text{for } j \neq i,\ j = 1,2,\ldots,n \qquad (4.6)$$

where, X_{ik} is a $|T_{ik}| \times 1$ column vector of positive integer elements. Similarly A_i is invariant if

$$\sum_{k=1}^{n} A_{ik}^{(i)} I_{ik} = 0 \qquad (4.7)$$

and, $$\sum_{k=1}^{n} A_{jk}^{(i)} I_{ik} = 0 \quad \text{for } j \neq i,\ j = 1,2,\ldots,n \qquad (4.8)$$

when, I_{ik} is $|P_{ik}| \times 1$ column vector of positive integer elements.

From the definition 4.1 and the conditions of connecting n subnets, the incidence matrix A of resultant net N can be obtained. Now it can be shown from the definition that N is consistent if

$$\sum_{k=1}^{n} A_{ki}^{(i)T} X_{ik} = 0 \quad \text{for } i = 1,2,\ldots,n \qquad (4.9)$$

and, $$\sum_{k=1}^{n} A_{kj}^{(i)T} X_{ik} + \sum_{k=1}^{n} A_{ki}^{(i)T} X_{kj} = 0,$$

$$\text{for } j \neq i,\ \ j = 1,2,\ldots,n \qquad (4.10)$$

Equation (4.9) satisfies when equation (4.5) is satisfied for all i's. Equation (4.10) is sum of equation (4.6) for i th and j th subnets. If equation (4.6) is satisfied then equation (4.10) will also be satisfied. Therefore, N is consistent if each

$N_1, N_2, \ldots, N_n$ is consistent. A similar treatment can prove for invariance.

4.2.2 DECOMPOSITION OF PETRI NETS

In this section a number of basic propositions involving decomposition of Petri nets are obtained and their use demonstrated. Consider a net N = (A,—). Let there be some transitions $t_{k1}, t_{k2}, \ldots, t_{ks}$ which are connected to same places. These can be combined into one subset $T_k = \{t_{k1}, t_{k2}, \ldots, t_{ks}\}$. All such subsets of net A can be obtained, call these $T_1, T_2, \ldots$ Now if there are places which are connected to same subsets $T_1, T_2, \ldots$, then these places are combined in a subset of places. All such subsets of places $P_1, P_2, \ldots$ can be identified. The columns and rows of A can be interchanged and combined into submatrices such that the columns and rows of new matrix A' correspond to P_i's and T_i's, respectively. The matrix A can easily be obtained. As an illustration let us consider a PN with given A in Fig.4.2

$$A = \begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{array} \begin{array}{c} \begin{array}{cccc} p_1 & p_2 & p_3 & p_4 \end{array} \\ \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 1 \\ -1 & 1 & 1 & -1 \end{bmatrix} \end{array}$$

Fig.4.2- Incidence Matrix of a Given PN

We find that ${}^{o}t_1^{o} = \{p_1, p_3\}$; ${}^{o}t_2^{o} = {}^{o}t_3^{o} = \{p_2, p_3\}$;

${}^{o}t_4^{o} = \{p_3, p_4\}$; ${}^{o}t_5^{o} = \{p_1, p_2, p_3, p_4\}$

Hence, $T_1 = \{t_1\}$; $T_2 = \{t_2, t_3\}$; $T_3 = \{t_4\}$; $T_4 = \{t_5\}$

Now ${}^{\circ}p_1^{\circ} = \{T_1, T_5\}$; ${}^{\circ}p_2^{\circ} = \{T_1, T_2, T_5\}$; ${}^{\circ}p_3^{\circ} = \{T_2, T_4, T_5\}$;

${}^{\circ}p_4^{\circ} = \{T_4, T_5\}$.

Since none of p's are connected to same subset of transitions,

$$P_1 = \{p_1\};\ P_2 = \{p_2\};\ P_3 = \{p_3\};\ P_4 = \{p_4\}$$

Thus,

$$A' = \begin{array}{c} \\ \{t_1\} \\ \{t_2, t_3\} \\ \{t_4\} \\ \{t_5\} \end{array} \begin{array}{c} \begin{array}{cccc} \{p_1\} & \{p_2\} & \{p_3\} & \{p_4\} \end{array} \\ \begin{bmatrix} [1] & [-1] & [0] & [0] \\ \begin{vmatrix} 0 \\ 0 \end{vmatrix} & \begin{vmatrix} -1 \\ 1 \end{vmatrix} & \begin{vmatrix} 1 \\ -1 \end{vmatrix} & \begin{vmatrix} 0 \\ 0 \end{vmatrix} \\ [0] & [0] & [-1] & [1] \\ [-1] & [1] & [1] & [-1] \end{bmatrix} \end{array}$$

Fig. 4.3- New Incidence Matrix

Proposition 4.6 : A given consistent (invariant) PN can be decomposed into n subnets each of which is consistent (invariant). The number n is obtained by selecting a minimum value which satisfies:

Max (number of nonzero submatrices in a row, column of A')

$$\leq 2(n-1)$$

Proof : First part directly follows from proposition 4.5. For the second part it can easily be shown that the maximum number of nonzero submatrices in the incidence matrix of resulting net N if all subnets $N_1, N_2, \ldots, N_n$ are pairwise connected (proposition 4.5) equals 2(n-1). Hence, the proof.

Proposition 4.7 : For a given PN to be decomposable into an subnets, the following must hold good for $i \neq j$; $i,j = 1,2,\ldots,n$

i) ${}^{\circ}P_{ij}^{\circ} \cap T_{ij} = P_{ij} \cap {}^{\circ}T_{ij}^{\circ} = \emptyset$

ii) ${}^{\circ}P_{ii}^{\circ} \cap T_{jj} = {}^{\circ}T_{ii}^{\circ} \cap P_{jj} = \emptyset$

and for $n \geq 3$, for $i \neq j \neq k$; $i,j,k = 1,2,\ldots,n$

iii) ${}^{\circ}P_{kk}^{\circ} \cap T_{ij} = {}^{\circ}T_{kk}^{\circ} \cap P_{ij} = \emptyset$

iv) ${}^{\circ}T_{ii}^{\circ} \cap {}^{\circ}T_{jj}^{\circ} \cap {}^{\circ}T_{kk}^{\circ} = {}^{\circ}P_{ii}^{\circ} \cap {}^{\circ}P_{jj}^{\circ} \cap {}^{\circ}P_{kk}^{\circ} = \emptyset$

Proof : (i) and (iii) are the basic conditions for interconnection (definition 4.4 and that of decomposition)

(ii) Since T_{jj} is the set of transitions exclusively in N_j, no transition of T_{jj} will be included in ${}^{\circ}P_{ii}^{\circ}$ i.e. the set of transitions connected to places exclusively in N_i, hence

$${}^{\circ}P_{ii}^{\circ} \cap T_{ij} = \emptyset$$

Similarly it can be shown that

$${}^{\circ}T_{ii}^{\circ} \cap P_{jj} = \emptyset$$

(iv) From definition 2.2,

$${}^{\circ}T_{ii}^{\circ} = (P_{i1} \cup P_{i2} \cup \ldots\ldots \cup P_{ij} \cup \ldots.)$$

$$= (P_{i1}, P_{i2}, \ldots., P_{ij}, \ldots.)$$

(because each place p is included in one and only one set)

Therefore,

$$ {}^{o}T_{ii}^{o} \cap {}^{o}T_{jj}^{o} \cap {}^{o}T_{kk}^{o} = P_{ij} \cap {}^{o}T_{kk}^{o} = \emptyset \quad \text{(from iii)} $$

Similarly

$$ {}^{o}P_{ii}^{o} \cap {}^{o}P_{jj}^{o} \cap {}^{o}P_{kk}^{o} = \emptyset $$

Q.E.D.

Remark : Propositions 4.5, 4.6 and 4.7 give a method to decompose a consistent (invariant) PN into smaller consistent (invariant) nets. It may be noted that each net is different and cannot be decomposed further. The above mentioned method is illustrated with the help of an example of Fig.4.2. The new matrix A' has been obtained in Fig.4.3. The maximum number of nonzero submatrices in any row or column is 4. Hence the number of subnets to which it can be decomposed into is given by (proposition 4.6).

$$ 4 \leq 2(n-1) $$

or $n = 3$

Therefore, the net N will be decomposed into 3 subnets. Let these be N_1, N_2 and N_3. The problem is now to find all P_{ii}'s, T_{ii}'s, P_{ij}'s and T_{ij}'s for $i \neq j$; $i,j = 1,2,3$.

From A' :

$$ {}^{o}\{t_1\}^{o} = \{p_1, p_2\} \; ; \; {}^{o}\{t_2, t_3\}^{o} = \{p_2, p_3\} \; ; \; {}^{o}\{t_4\}^{o} = \{p_3, p_4\} ; $$

$$ {}^{o}\{t_5\}^{o} = \{p_1, p_2, p_3, p_4\} $$

$$ {}^{o}\{p_1\}^{o} = \{t_1, t_5\} \; ; \; {}^{o}\{p_2\}^{o} = \{t_1, \{t_2, t_3\}, t_5\} \; ; $$

$$ {}^{o}\{p_3\}^{o} = \{\{t_2, t_3\}, t_4, t_5\} \; ; \; {}^{o}\{p_4\}^{o} = \{t_4, t_5\} $$

Applying 4.7a(i), it is found that

$$^{o}\{t_1\}^{o} \cap {}^{o}\{t_2, t_3\}^{o} \cap {}^{o}\{t_4\}^{o} = \emptyset$$

Hence, arbitrarily $T_{11} = \{t_1\}$; $T_{22} = \{t_2, t_3\}$; $T_{33} = \{t_4\}$. It must be recalled that T_{ii} is the set of transitions exclusively contained in subnet N_i. Using $^{o}T_{ii}^{o} \,||\, {}^{o}T_{jj}^{o} = P_{ij}$, it is found that $P_{12} = {}^{o}T_{11}^{o} \cap {}^{o}T_{22}^{o} = \{p_1, p_2\} \cap \{p_2, p_3\} = \{p_2\}$

Similarly $P_{13} = \emptyset$ and $P_{23} = \{p_3\}$

Further, $^{o}T_{11}^{o} = \{P_{11}, P_{12}, P_{13}\}$

Hence $P_{11} = \{p_1\}$

Similarly $P_{22} = \{\emptyset\}$ and $P_{33} = \{p_4\}$

Remaining transition set $\{t_5\}$ is to be assigned. Obviously, $T_{ij} = \{t_5\}$. Only i and j are to be identified. For this (i) is used. i.e.

$$^{o}P_{ij}^{o} \cap T_{ij} = \emptyset$$

Hence, $T_{ij} = T_{13} = \{t_5\}$

The complete decomposition is shown in Fig.4.5

Hence,

$$A_1 = \begin{array}{c} \\ t_1 \\ t_5 \end{array} \begin{array}{c} \begin{array}{cc} p_1 & p_2 \end{array} \\ \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \end{array} \quad ; \quad A_2 = \begin{array}{c} \\ t_2 \\ t_3 \end{array} \begin{array}{c} \begin{array}{cc} p_2 & p_3 \end{array} \\ \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \end{array}$$

and

$$A_3 = \begin{array}{c} \\ t_4 \\ t_5 \end{array} \begin{array}{c} \begin{array}{cc} p_3 & p_4 \end{array} \\ \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \end{array}$$

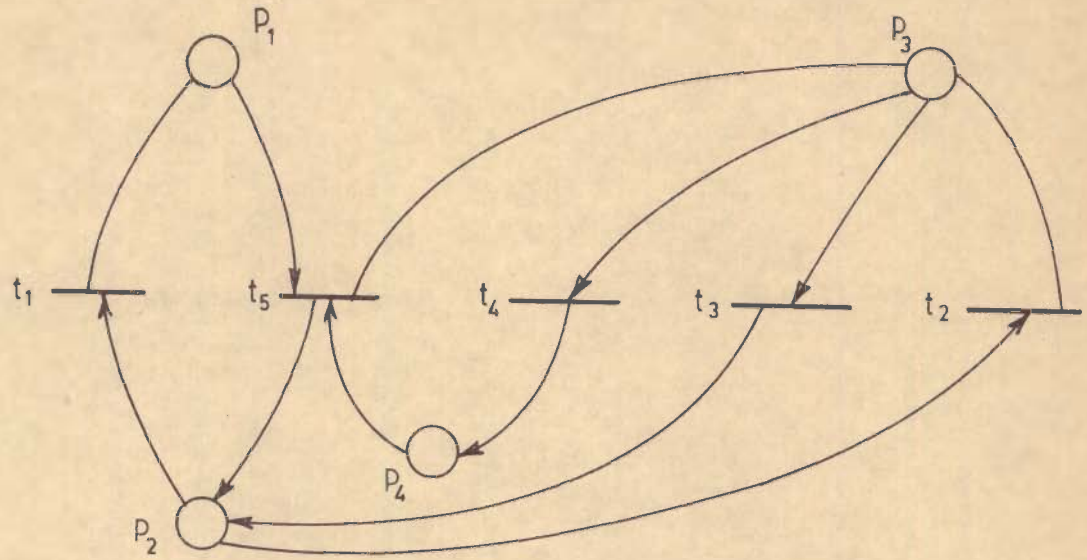


FIG. 4.4 _ PN WITH MATRIX A IN FIGURE 4.3

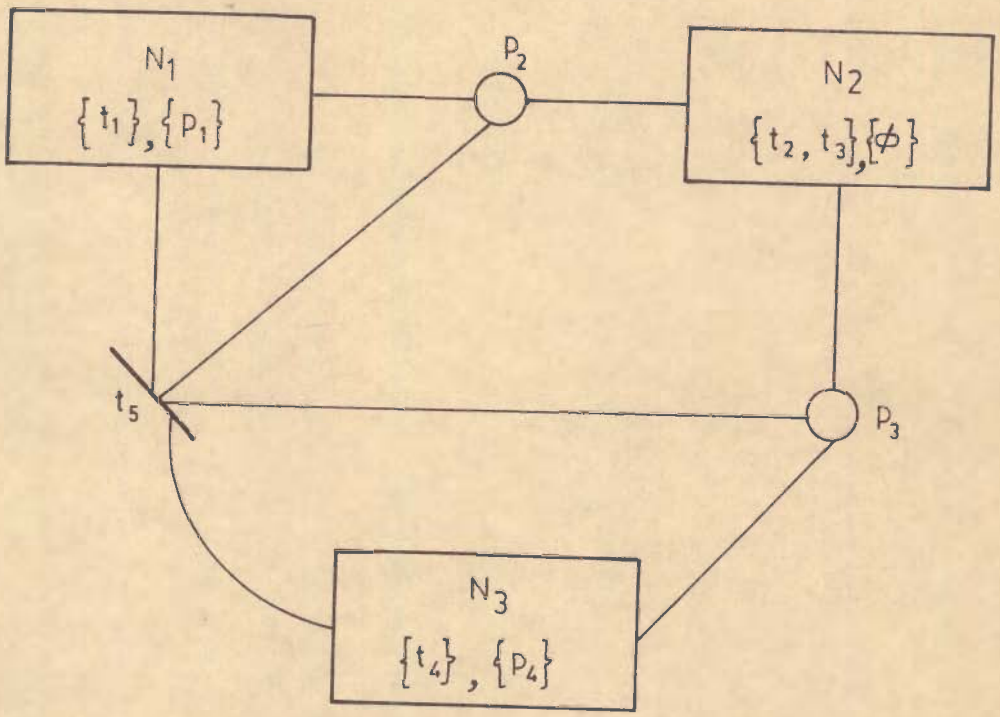


FIG. 4.5 _ DECOMPOSITION OF PN SHOWING IN FIG. 4.4

It can be seen that each of N_1, N_2 and N_3 is consistent and invariant which it should be according to proposition 4.5 and 4.6.

4.3 MINIMAL LEGAL FIRING SEQUENCES IN PETRI NETS

The reachability problem is one of the most important problems of Petri nets. In Section 2.6, it has been elaborated that while it cannot be solved by the reachability tree analysis technique, state equation method provides only a necessary condition for transforming an initial marking M_o to another M_n of a PN. Hence, to solve this problem it becomes necessary to find atleast one legal firing sequence (LFS) which can transform M_o to M_n. It is this determination of LFS which is studied here. In order to do so, Murata's state equations given in section 2.6.2 are produced below for easy reference:

$$M_k = M_{k-1} + A^T V_K \geq 0 \qquad (2.2)$$

$$A^T \Sigma = \Delta M \qquad (2.3)$$

It was also shown in the same section that the existence of solution Σ of equation (2.3) is a necessary condition for reachability. Further, solution Σ is not generally unique but a set of solutions. Thus, it is required to show that one or more of these solutions are legal (i.e. a LFS exists). Obviously trying all solutions is quite difficult task. To reduce this complexity it is shown in subsequent paragraph that only minimal firing count vector is sufficient to be considered. This will correspond to minimal legal firing sequences.

4.3.1 THEORY INVOLVED

Before the technique is given, following two theorems are postulated.

Theorem 4.1 : In a PN, a marking M_o can not reach another marking M_n, if

$$A\Delta M = 0 \qquad (4.11)$$

Proof : It has been shown [97] that M_o can not reach M_n if we can obtain a nonzero column vector V_n such that

$$\Delta M = B_f^T V_n$$

where B_f is $(|P|-r) \times |P|$ unimodular matrix orthogonal to A (r is the rank of A).

Hence, $A\Delta M = A B_f^T V_n = 0$

Theorem 4.2 : Let Σ_o be a minimum firing count vector. Then in a PN,

(a) If Σ_o is not executable, Σ is not executable

(b) If Σ_o is executable, then Σ is also executable and the legal firing sequences to execute Σ_o are minimal.

Proof : Any non-negative integral firing vector Σ can be expressed as $\Sigma = \Sigma_o + \Sigma_1$, when Σ_1 is another non-negative integral vector. Since Σ_o is also a solution of (2.3), $A^T \Sigma_1 = 0$.

Thus, (a) If Σ_o is not executable (i.e. no LFS exists with Σ_o), the LFS to execute Σ_1 even if it exists will not change the marking M_o to M_n and hence, Σ is not executable.

(b) If Σ_1 is not executable, then LFS for Σ is same as that of Σ_o. If Σ_1 is executable, then LFS for Σ_1 can be included

at any marking M_j which is encountered in changing the marking M_o to M_n. This is because LFS for Σ_1 will not change M_j. Hence, if LFS for Σ_o exists then there exists a LFS for Σ and minimal LFS for Σ equals the LFS of Σ_o.

Q.E.D

To solve equation (2.3) for Σ_o, let the rank of A be r. Then A can be partitioned as:

$$A = \begin{array}{c} \overset{|P|-r \quad\quad r}{\longleftrightarrow \quad\quad \longleftrightarrow} \\ \left[\begin{array}{cc} A_{11} & A_{12} \\ A_{21} & A_{22} \end{array}\right] \begin{array}{l} \updownarrow r \\ \updownarrow |T|-r \end{array} \end{array}$$

where A_{12} is a rxr non-singular matrix.

Then equation (2.3) can be written as

$$\begin{array}{c} |P|-r \\ r \end{array} \left[\begin{array}{cc} A_{11}^T & A_{21}^T \\ A_{12}^T & A_{22}^T \end{array}\right] \left[\begin{array}{c} \Sigma_1 \\ \Sigma_2 \end{array}\right] \begin{array}{l} \updownarrow r \\ \updownarrow |T|-r \end{array} = \left[\begin{array}{c} \Delta M_1 \\ \Delta M_2 \end{array}\right] \begin{array}{l} \updownarrow |P|-r \\ \updownarrow r \end{array} \qquad (4.12)$$

Eq.(4.12) is a set of ($|T|-r$) linear independent equations in $|T|$ unknown and $\Sigma = [\Sigma_1 \quad \Sigma_2]^T$ can be obtained as

$$\Sigma = \left[\begin{array}{c} -(A_{12}^T)^{-1} \; A_{22}^T \Sigma_2 + (A_{12}^T)^{-1} \; \Delta M_2 \\ \Sigma_2 \end{array}\right] \qquad (4.13)$$

Therefore, Σ can always be expressed in terms of ($|T|-r$) unknowns. Now the vector Σ_2 can be selected such that (i) the entries of Σ are nonnegative, and (ii) Σ contains maximum number of zeros. This Σ is called as minimal firing vector Σ_o.

4.3.2 DETERMINATION OF MINIMAL LEGAL FIRING SEQUENCE

The proposed technique for determination of minimal LFS is given in the form of following steps.

1. For the given PN, M_o and M_n, find A from definition (2.3), and $\Delta M = M_n - M_o$.
2. If $A\,\Delta M = 0$, M_o can not reach M_n. Go to step 13.
3. If no solution Σ of equation (2.3) exists, M_o will never reach M_n. Go to step 13.
4. Find Σ and then $\Sigma_o = [\sigma_{oi}]_{|T|x1}$ from (4.13). (The element σ_{oi} represents the number of times i th transition will fire in a minimal sequence to change marking M_o to M_n).
5. Find total number of firings n from

$$n = \sum_{i=1}^{|T|} \sigma_{oi}$$

6. (Construction of reachability tree starts)

   Put $j = 1$, and $M_{o,1} = M_o$ (initial node)

7. Find a set of transitions T such that

   $\sigma_{oi} \neq 0 \quad i \in T$

   for $i \in T$ and all j, find

   $$M_k(i) = M_{k-1,j} + A^T V_k$$

   where, $M_k(i)$ is a marking resulting from $M_{k-1,j}$ on firing transition i. $M_{k-1,j}$ is j th node of reachability tree before k th firing.
8. If $\forall\, i \in T$, $M_k(i) \not\geq 0$, no LFS exists. Go to step 13.
9. Identify all the markings $M_k(i)$ which have not appeared previously, and which satisfy $M_k(i) \geq 0$. Call these markings

$M_{k,1}$, $M_{k,2}$,.....$M_{k,s}$,....,$M_{k,l}$ nodes. Connect a node $M_{k,s}$ to a node $M_{k-1,j}$ by an arc i if M_{ks} results from $M_{k-1,j}$ by firing i th transition. Do it for every s and every j.

10. Put $\sigma_{oi} = \sigma_{oi} - 1$ for i ε T and $M_k(i) \geq 0$

11. For k = 1 , n; repeat steps 6, 7, 8, 9 and 10 (Reachability tree is complete).

12. Determine all the paths from node M_o to M_n in terms of arcs. Each path gives a minimum LFS

13. Stop.

The procedure is illustrated with thelp of following example.

Example 4.1 : Consider the PN of Fig.4.6a. The minimal LFS to transform $M_o = [10000]^T$ to $M_n = [00001]^T$ are to be determined.

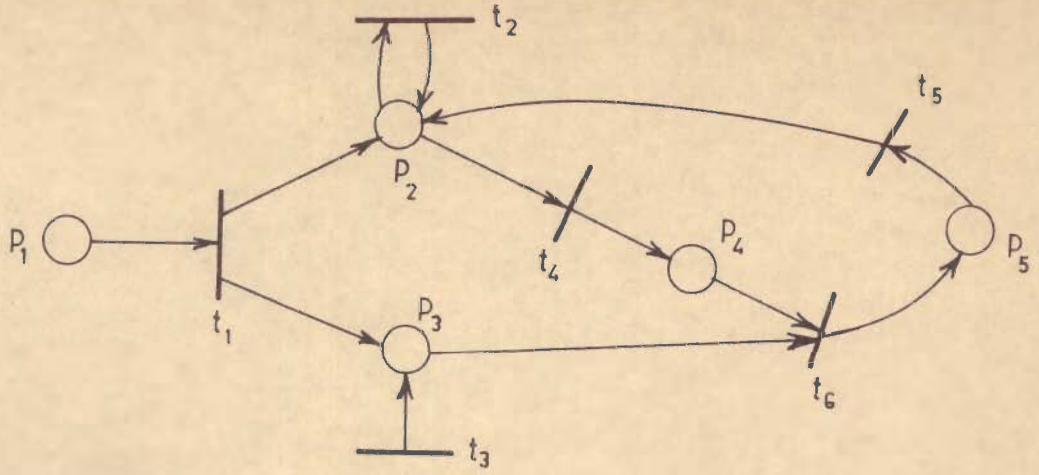
Using the proposed technique, we proceed as

Step 1:

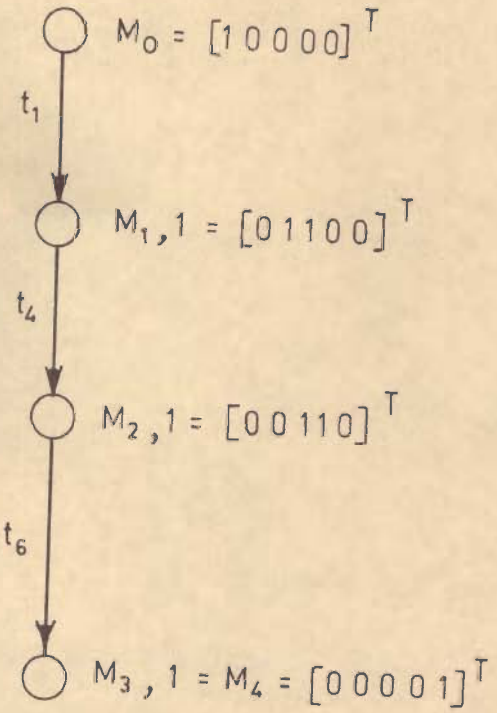
$$A = \begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \end{array} \begin{array}{c} \begin{array}{ccccc} p_1 & p_2 & p_3 & p_4 & p_5 \end{array} \\ \begin{bmatrix} -1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & -1 & -1 & 1 \end{bmatrix} \end{array}, \text{ and } \Delta M = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{array} \begin{array}{c} \\ \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \end{array}$$

Step 2: $A\,\Delta M = [1\ 0\ 0\ 0\ -1\ 1]^T \neq 0$

Step 3,4: Solving eq.(2.3) we get $\Sigma = [1\ \sigma_2\ \sigma_{6-1}\ \sigma_6\ \sigma_{6-1}\ \sigma_6]^T$ and Σ_o is obtained by putting $\sigma_2 = 0$, $\sigma_6 = 1$. Hence $\Sigma_o = [1\ 0\ 0\ 1\ 0\ 1]^T$



(a) A GIVEN PN



(b) REACHABILITY TREE FOR MINIMAL LFS

FIG. 4.6 - A PN AND ITS REACHABILITY TREE FOR MINIMAL LFS

Step 5: $n = 3$

Steps 6,7,8,9 and 10 give for k=1, j=1, $M_{o,1} = [0\ 1\ 1\ 0\ 0]^T = M_o$

$$T = \{t_1, t_4, t_6\}$$

$$M_1(t_1) = [1\ 0\ 0\ 0]^T + \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = [0\ 1\ 1\ 0\ 0]^T$$

Similarly,

$M_1(t_4) = [1\ -1\ 0\ 1\ 0]^T$, and $M_1(t_6) = [1\ 0\ -1\ -1\ 1\ 0]^T$

Since $M_1(t_4)$ and $M_1(t_6)$ both are < 0, t_4 and t_6 can not fire at M_o. Hence the only node available at k = 1, is $M_{1,1} = [0\ 1\ 1\ 0\ 0]^T$. In reachability tree it is connected to M_o by an arc t_1.

Step 10: $\sigma_{ot_1} = 0$ and $\sigma_{ot_4} = \sigma_{ot_6} = 1$.

Step 11: Similarly repeating steps 6, 7, 8, 9, 10 and 11 for k = 2,3, we find the reachability tree as shwon in Fig. 4.6b.

Step 12: There is only one path from M_o to M_n and hence minimal LFS is $t_1 t_4 t_6$.

4.4 STATE EQUATION REPRESENTATION OF LOGIC OPERATIONS THROUGH PETRI NETS [75]

It is observed that Petri nets cannot model systems in which logic operations other than AND and OR is involved. A detailed discussion about this has already been made in Section 1.3. This modeling limitation is overcome by defining a NOT transition. Now a logical question that arises is how to analyse such Petri nets. The state equation by Murata [97] cannot be directly employed for this purpose because it takes care of AND and OR logic only. In the following paragraphs it is shown how this problem can be solved by obtaining a generalized state equation for logic operations such as NOT, NAND, NOR and EX-OR. The transition executing these operations are represented by different symbols (Figs. 4.7, 4.8a, 4.9, 4.10). Once a PN with generalized state equation is obtained, it becomes a versatile tool that can be used in a variety of applications in analysing the behaviour of computer systems. As a prelude to the discussion, consider Fig.4.11 which is a part of CDC 6400 PN representation at first level of abstraction [105]. Fig. 4.11 is a slightly expanded form of Fig.1.1 of Section 1.3. A part of the section is given here again for maintaining the readability.

It represents the conditions and events for advancing a job from staging queue to input queue. The event corresponding to program ISI (Fig.4.11) might be executed provided the conditions given below are met.

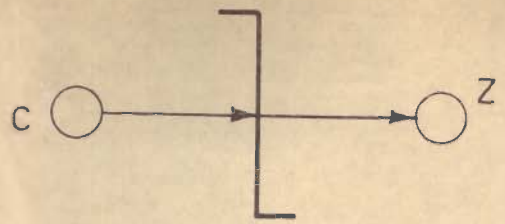


FIG. 4.7_ NOT OPERATION

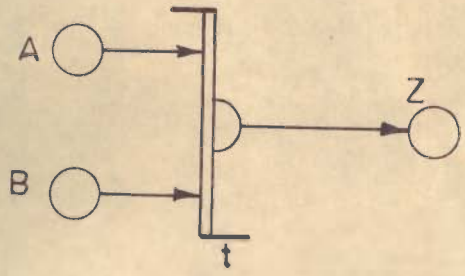


FIG. 4.8 a _ NAND OPERATION

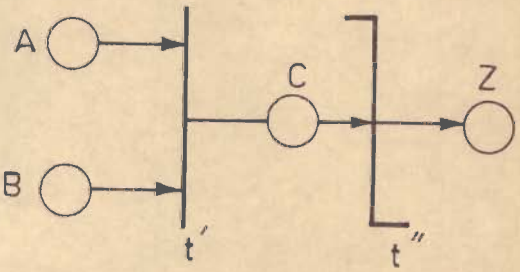


FIG. 4.8 b _ EXPANDED FORM OF FIG. 2

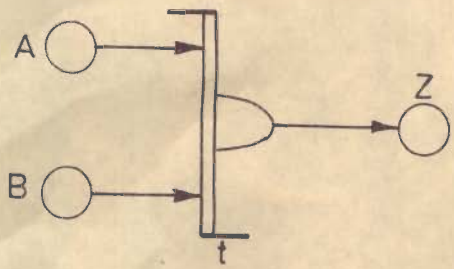


FIG. 4.9_NOR OPERATION

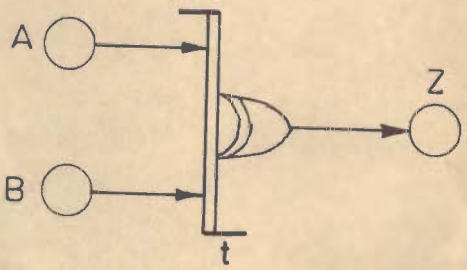


FIG. 4.10 _ EX-OR OPERATION

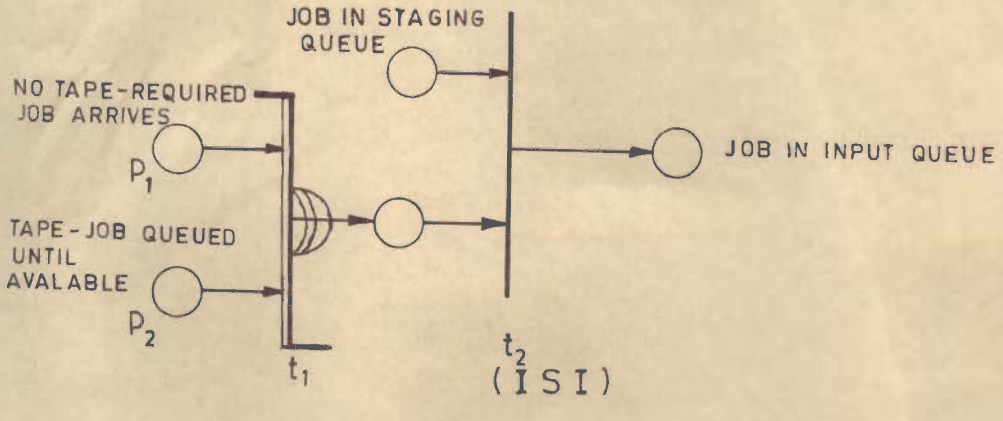


FIG. 4.11 _ EXPANDED FORM OF FIG. 1.1

"Job in staging queue AND (logic only one of the conditions (i) 'No tape required-job advances' and (ii) 'tape-job queued until tape available' is satisfied"

Such a system cannot easily be represented by only logic operations. It is represented partly by logic operations (such as EX-OR and AND) and partly by operations representing the execution of the program ISI. The second type of operations is represented by PN because of advantages discussed in Section 1.2. If logic operations are represented by any other method, the analysis becomes unwieldy if not impossible because of the interaction of two different types of model. Hence, it is of significant importance to represent both types of operations by a single model and to express all the logic operations through state equations of PN.

It is interesting to note that the state equations given by Murata are a special case of the generalized state equations proposed here.

4.4.1 GENERALISED STATE EQUATION

The convention adopted here is to represent the presence of an uncomplemented (complemented) variable by the presence (absence) of a token. Thus a place can have atmost one token. As a result of some matrix operations a token with negative sign may appear on a place. Physically this means the removal of a token from the corresponding place. However, a token cannot be removed from a place containing no token. Hence besides obeying binary addition rules on the token, following three rules are

additionally observed throughout the text.

i) $0 + (-1) = 0$

ii) $1 + (-1) = 0$

iii) $(-1) + (-1) = -1$

4.4.1.1 NOT OPERATION

Consider Fig.4.7. There are only two possible states i.e. either C or Z contains a token and vice-versa. In the former, if an uncomplemented variable arrives at C, the transition should not fire in order to have the next state same as earlier one. However, if a complemented variable arrives, the transition should fire such that the state changes. Similarly in the latter, transition will fire only when there is a change in input token. These can be combined into one state equation

$$M_{k+1} = M_k + A^T D_K V_K \qquad (4.14)$$

where, M_{K+1}, M_K, A^T and V_K have usual meaning as defined (Section 2.6.2) and $D_K = [\Delta m_{ink}]_{|T|x|T|}$ with element Δm_{ink} given by

Δm_{ink} = (Token at k th firing minus token prior to k th firing) at a place input to transition i

The transition will fire only when there is a change in the tokens at the input place.

4.4.1.2 NAND OPERATION

Consider Fig.4.8b which is an expanded form of Fig.4.8a. Since t' represents AND operation, it will fire when all its inputs contain tokens. The transition t" executes NOT operation

and will fire only when there is a change in token at place C. The state equations representing NAND operation to give the next state M_{K+1} from the previous state M_k, can be written as

$$M_{K+1} = M_K + A^{*T} D_K V_K \qquad (4.15)$$

where, M_k is column vector whose element m_{jk} gives the number of tokens on place j prior to k th firing.

A^{*T} is a transpose of $2|T| \times |P|$ modified transition-to-place incident matrix such that

$$A^{*T} = [a'_{ji} \quad a''_{ji}]_{|P| \times |2T|}$$

The elements of A^{*T} are given by $a'_{ji} = -1$, if place j is an input to transition i = 0, otherwise; $a''_{ji} = 1$, if place j is an output to transition i = 0, otherwise. D_K is $2|T| \times 2|T|$ diagonal matrix, called characteristic matrix, whose element d_{ik} is given by

$$d_{ik} = \begin{bmatrix} 1 & \vdots & 0 \\ \cdots & \vdots & \cdots \\ 0 & \vdots & 1-\Delta m_{oik} \end{bmatrix}$$

where Δm_{oik} is change in the number of tokens at output place of i th transition prior to k th firing and, V_K is $2|T| \times 1$ column vector whose element vector v_{ik} is given by

$$v_{ik} = [\, v'_{ik} \quad v''_{ik} \,]^T$$

The firing of a transition t_i is infact, a sequence of firing of two transitions t_i' and t_i''. When t_i' (t_i'') is made to fire v_{ik}' (v_{ik}'') is '1' , otherwise '0' . The transitions t_i' and t_i'' are enabled to fire iff

$$M_K + \bar{A}^{*T} V_K' \geq 0 \qquad (4.16)$$

where, $\bar{A}^{*T} = [a'_{ji}]$ and $v_k' = [v'_{1k} \quad v'_{2k} \ldots\ldots v'_{TK}]^T$

$$1 - \Delta m_{oik} \neq 0 \qquad (4.17)$$

For Fig.4.8b the order of M_k, A^{*T}, D_K and V_K are 3 x 1, 3 x 2, 2 x 2, and 2 x 1, respectively and Δm_{oik} equals $(1-m_{Ck} + m_{Ak} \cdot m_{Bk})$. The number of equations in (4.15) can be reduced by eliminating the terms corresponding to place C. Hence for NAND (Fig.4.8a). Eqn.(4.15) still holds with the value of Δm_{oik} given by

$$\Delta m_{oik} = m_{oik} - \prod_1 m_{lik}$$

where m_{oil} (m_{lik}) is the number of tokens on output place(l th input place) to i th transition, and $\prod$ is binary multiplication.

4.4.1.3 NOR OPERATION

Based upon similar reasoning, it can be shown that (4.15) represents the state equation for NOR operation (Fig.4.9). However, in this case Δm_{oik} equals $m_{oik} + \bigvee_1 m_{lik}$ where $\bigvee$ is a binary addition. The transition t_i' is enabled to fire if for any place j, condition (4.16) is satisfied and t_i'' is enabled if (4.17) is satisfied.

4.4.1.4 EX-OR OPERATION

Similarly, the state equation (4.15) can be shown to be representing EX-OR operation (4.10). D_K for this case is obtained by making Δm_{oik} equal to $1+m_{oik} - \bigvee_1 m_{lik} + \prod_1 m_{lik}$. The transition t_i' is enabled to fire if for exactly one place j,

condition (4.16) is satisfied and enabling condition for t_i" is given by (4.17).

It is evident from the above that equation (4.15) can represent all the logic operations. Depending upon the type of logic operation a transition t_i performs, D_K can be obtained. It is also possible to represent NOT by equation (4.15) by making

$$D_K = \begin{bmatrix} 1 & 0 \\ 0 & \Delta m_{ok} \end{bmatrix}$$

However, Δm_{oik} and firing rules will be different for different operations. These are summarized in Table 4.1

Logic operations	Δm_{oik}	Firing rules	
		t fires if 4.16 is satisfied	t is fired if 4.17 is satisfied
NAND	$m_{oik} - \prod_1 m_{lik}$	for all input places	-do-
NOR	$m_{oik} + {}_1 m_{lik}$	for any input place j	-do-
EX-OR	$1+m_{oik} - \bigvee_1 m_{lik} + \prod_1 m_{lik}$	for exactly one input place j	-do-
AND(OR)	0	for all input places	-

TABLE 4.1

It is interesting to note that if

$$D_K = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

(4.15) reduces to Murata's state equation [97].

The above concepts are illustrated with the help of an example (Fig.4.11).

Example:

For Fig.4.11, it is desired to find (i) all the markings (token distribution) and (ii) the time required by the job to reach the input queue from staging queue.

(i) Assuming that a job in staging queue does not require a tape, A^{*T} and M_K are obtained as

$$A^{*T} = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{array} \begin{array}{c} \begin{array}{cccc} t_1' & t_1'' & t_2' & t_2'' \end{array} \\ \begin{bmatrix} -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

$$M_K = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{array} \begin{array}{c} \\ \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \end{array}$$

From Table 4.1

$$d_{1k} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

and

$$d_{2k} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

When t_1 fires, $V_{1k} = [1\ 1\ 0\ 0]^T$ and next marking is obtained from (4.15) as $M_{K+1} = [0\ 0\ 1\ 1\ 0]^T$. Similarly when t_2 fires $V_{2k} = [0\ 0\ 1\ 1]^T$ and $M_{k+2} = [0\ 0\ 0\ 0\ 1]^T$. In this case M_{k+2} is the final marking.

(ii) For the job in staging queue to be transferred to input queue both the transitions t_1 and t_2 have to fire (obtained from (i)). Hence, total time (T) required by the job to reach input queue is given by $T = T_1 + T_{ISI}$, where T_1 is the time at which the job is available in staging queue, and T_{ISI} is the time required for the execution of the program represented by t_2. (Time required by the logic operations is assumed to be negligible).

4.5 CONCLUSION

In this chapter the problems of interconnection and decomposition of Petri nets, reachability, and analysis of PN with additional transitions exhibiting NOT operations have been studied. The interconnections have been dealt with reference to invariance and consistency. Many propositions have been postulated and their utility shown. It has also been demonstrated how invariant and consistent PN can be decomposed into smaller nets

with the desired properties. Unlike the existing technique [137] (Section 2.6.2), the proposed method of decomposition of a PN does not require the solution of state equation. The derivation of positive integer-valued solution of a state equation is a complex task. In this sense, the proposed method is better. Another unique feature of the approach adopted here is that it is possible to know 'apriori' the number of subnets into which the PN decomposes.

The reachability problem has been solved by finding minimal legal firing sequence. The proposed technique combines to advantage both the analysis methods namely, reachability tree and state equation approach. It has been established that if a LFS exists for minimal firing count vector Σ_o, then there exists a LFS for Σ. This avoids an exhaustive search of atleast one of the many solutions of Σ which may correspond to a LFS. The determination of an expression for Σ is needed in both the cases. To determine Σ_o from Σ does not involve large additional efforts. On the other hand, the proposed method requiresless computations to find LFS which turns out to be minimal. Further, the verification of reachability is easier because of lesser length of minimal LFS.

A generalized state equation for PN model of logic operations has been obtained. This overcomes the limitation of Murata's representation, as NOT and logic NAND gates could not be given PN representation with earlier formulation. The analysis of a computer system using this proposed technique is shown with the

help of an example. It is simple and adaptable on computer. It appears that the detection and isolation of fault in the system can also be done by extending Murata's method in which KVL has been applied to marked graph.

CHAPTER V

ON THE APPLICATION OF PETRI NETS TO COMPUTER HARDWARE AND SOFTWARE

5.1 INTRODUCTION [73]

This chapter investigates PN as a tool that can be applied in number of instances. The basis of study is state equation of Petri nets which is given again for the purpose of clarity as follows:

$$A^T \Sigma = \Delta M \qquad (2.3)$$

It is shown here that eq.(2.3) is very powerful and applicable to a wider class of problems in computer hardware and software. This results in solving many problems by obtaining a solution Σ of (2.3). The problems only differ in the class of the above equation in which ΔM can have different type of column vector. Some of the problems are identified in the following specific classes:

Class 1 : Any two entries of ΔM have unit values, one having -1 and other +1, respectively while remaining entries are 0's. This class is useful in finding all the simple paths between two nodes of a graph.

Class 2 : All the entries of ΔM are 1's. This class is used in determining all the maximal compatable class in control memory bit optimization of microprogrammed computers. The

detailed discussion of this is deferred until the next chapter.

Class 3 : All the entries of ΔM are 0's. This is useful for code optimization of assembly language programs generated in the process of compilation [63]. Instructions, locations and datas of assembly language code (generated after the parsing and code generation in compiler) is represented by transitions, places and tokens of PN, respectively.

In this chapter only the problems which are covered by class 1 are taken up. The subsequent sections find the enumeration of simple paths between two nodes of a graph. Furthermore, the importance of simple paths in the problems of terminal reliability of computer network and program complexity evaluation is investigated and the solutions of these problems through Petri nets are proposed.

5.2 ENUMERATION OF SIMPLE PATHS BETWEEN TWO NODES OF A GRAPH [71]

Often all simple paths between two specified nodes are needed. The computation of reliability between two stations of a computer network is just one of the many examples. This section tackles precisely this problem. In literature, are available many methods for this purpose. The adjacency matrix technique [30] is one of them. The adjacency matrix $X = [x_{ij}]$ of a directed graph with V vertices is V x V matrix, where $x_{ij} = 1$, if there is an edge directed from vertices v_i and v_j, and 0 otherwise. The L th power of the adjacency matrix X^L is

the number of simple directed paths of length L- edges between each pair of vertices v_i and v_j. It has been shown that the successive power technique [30], [128] requires V^4 matrix operations. Rubin [133] has applied Warshall ordering technique [149] to find all simple paths, resulting in V^3 matrix operations. In this section the concept of PN is applied. Here no multiplication is involved, only vector additions are needed.

5.2.1 FORMULATION

Let G = (V,E) be a finite directed graph without multiple edges or self-loops. V(1,2,....,V) is the list of vertices and E = (e_1, e_2,....,e_E) is the list of edges of G. The graph G can be represented by a Petri net by replacing the nodes by places and arcs by transitions. The transitions are assumed to fire in the direction of the arc. Hence to find all simple paths from a node r to a node k of G, it is sufficient to determine which of the transitions of the corresponding Petri net would fire in order that a marking which has the only token at place k is reachable from a marking which has the only token at place r. This is achieved by using the state equation (2.3). In this case eq.(2.3) reduces to

$$A_1 \Sigma = \Delta M \qquad (5.1)$$

where, $A_1 = [a_{ij}]_{V \times E}$ is the incidence matrix of the graph G whose element a_{ij} is given by

$$a_{ij} = \begin{cases} -1 & \text{if } j\text{th arc is oriented away from node } i \\ 1 & \text{if } j\text{th arc is oriented towards the node } i \\ 0 & \text{otherwise} \end{cases}$$

$\Delta M = [m_i]_{Vx1}$ is a column vector whose element m_i is given by

$$m_i = \begin{cases} -1 & \text{for } i = r \\ 1 & \text{for } i = k \\ 0 & \text{otherwise} \end{cases}$$

and, $\Sigma = [e_j]_{Ex1}$ is a column vector whose element e_j is 0, when j th arc is not included in the path.

As the paths are simple, e_j can atmost be 1.

Equation (5.1) contains V-1 independent equations in E unknowns. Each feasible solutions of (5.1) will correspond to one simple path.

5.2.2 SOLUTION

A path of length L in G is a non-empty sequence of L edges. The path is simple if all edges are distinct. Thus, the solutions of (5.1) can be obtained by column operations of A_1. Any L column vectors of A can be added and if the addition equals ΔM, then a simple path of length L is given by the branches whose indices correspond to these L columns. All such L combinations of columns will give all possible simple paths of length L. The number of iterations atmost will be equal to $\sum_{L=1}^{V-1} \binom{E}{L}$ because the minimum number of branches in a simple path will be V-1. However, it is possible to reduce the number of operations considerably

by considering the properties of matrix A. There may be certain nodes to which are connected either outgoint or incoming branches. Two or more columns which contain -1's (or 1's) for any row can not be added because the addition can never be equal to ΔM the element of which corresponding to a row is -1 or 0. Such sets of columns can be identified from A_1. Hence only those L ($L \geq 2$) columns are considered out of which any two columns do not belong to the set as obtained above. Based upon the above discussion following are the steps to find all simple paths between two specified nodes.

1. Find A_1 and ΔM from (5.1)
2. From A_1 find all the sets S_i for $1 \leq i \leq N$ of columns corresponding to each row which contains -1's (or 1's)only.
3. Compare all the columns of A_1 with ΔM. If any column equals ΔM, the branch with index which corresponds to the particular columns is the path of length 1.
   Put L = 2.
4. If any two columns of L belong to a single S_i obtained in step 2, GO TO step 8.
5. Add L columns. If the addition equals ΔM, the indices corresponding to these L columns are attached to the branches to give simple path of length L.
6. Repeat step 4 and 5 for all possible L combinations.
7. Make L = L + 1 until L = V-1 GO TO step 4.
8. Stop.

The proposed procedure is illustrated with the help of an example.

Example : Let the graph [133] G = (V,E) contain V = (1,2,3,4,5), and edges

$e_1 = (1,2)$; $e_2 = (1,3)$; $e_3 = (1,4)$; $e_4 = (2,4)$; $e_5 = (2,5)$; $e_6 = (4,5)$.

All simple paths between node 1 and 5 are to be obtained. Using the proposed technique we get the following:

Step 1

$$A_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \end{array}, \text{ and } \Delta M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Step 2 $S_1 = \{1,2,3\}$ and $S_5 = \{5,6\}$

Step 3 It is found that none of the columns equal ΔM, hence there is no path of length 1.

Step 4,5,6 L = 2

It is found that columns 1 and 5 can be added and simple path of length 2 is (e_1, e_5). Another simple path of length 2 is (e_3, e_6).

Step 7 Similarly it is obtained that the simple paths of length 3 is (e_1, e_4, e_6).

There is no other simple path.

5.2.3 MAXIMUM ITERATIONS NEEDED

In the preceeding section, a method to reduce the number of operations on A_1 has been given. This section finds what is the maximum number of iterations required in the proposed technique to determine simple paths between two nodes of a graph. Those columns of A_1 which equal ΔM can first be deleted. This is because they will never contribute to other simple paths. This will require M iterations. Now the remaining matrix can always be arranged and partitioned by interchanging the columns such that the resulting matrix.

$$A_1^* = [A_{in} \vdots A_b \vdots A_o]$$

where A_{in} is $V \times E_1$ matrix with all the entries in the row of initial node as -1's, A_o is $V \times E_3$ matrix with all the entries in the row of terminal node as +1's. A_b is the $V \times E_2$ remaining matrix. Obviously $E_1 + E_2 + E_3 = E$. Now the path of length L can be obtained by adding one column of A_{in}, $(L-2)$ columns of A_b and one column of A_o. No two or more columns of A_{in} or A_o could be added to equal ΔM. Thus, the maximum number of iterations to find path of length $L(L \geq 2) = E_1 E_3 \binom{E_2}{L-2}$.

Therefore, maximum number of iteration to find all simple paths

$$= E + \sum_{L=2}^{V-1} E_1 E_3 \binom{E_2}{L-2}$$

$$= E + E_1 E_3 \left(1 + \sum_{i=1}^{V-3} \binom{E_2}{i}\right)$$

Obviously, this will be much less than $\sum_{L=1}^{V-1} \binom{E}{L}$.

Having discussed the enumeration of simple paths through PN, the following sections are devoted to the application of such a concept to two important problems of computer science.

5.3 TERMINAL RELIABILITY OF A COMPUTER NETWORK [72]

One of the problems that arises in a computer network is to compute, in an efficient and systematic manner, the terminal reliability between a given pair of stations. It is the probability that there exists atleast one path between these two nodes. The solution for such a problem exists in literature [23], [39], [58], [77], [104]. An alternative procedure exploiting the reachability concept of PN is proposed here. In the proposed technique the computer network is represented as in [39] by probabilistic oriented graphs with weighted arcs and unweighted nodes, the nodes assigned to be reliable stations and the arcs represent the unreliable connections having a weight equal to its probability of existence. Further, it is assumed that there is no co-relation between failures of different links and reliabilities are time invariants. Graphs having parallel arcs are excluded and assumed to be combined into one arc.

In order to find the terminal reliability of a probabilistic graph, a Boolean function F (sum-of-product form) is constructed such that each term of F corresponds to one simple path the enumeration of which has been discussed in the preceeding section. Obviously, the terms of F are not disjoint. Therefore, it is not possible to replace each term by corresponding term of

probabilities. Hence disjoint terms of F are obtained by making use of firing rules of PN.

5.3.1 PROBABILISTIC GRAPH AND DETRRMINATION OF BOOLEAN FUNCTION

A probabilistic graph consists of unweighted nodes and weighted arcs. The nodes represent reliable stations and the arcs represent unreliable connections having weight equal to its probability of existence. The arcs are, in general, oriented but for some arcs no orientation is given. For such an arc the probability of existence is same in both the directions.

A probability graph can be drawn where the probability of existence P_{ij} of arc (i,j) is given by defining a stochastic variable x_{ij} having 0,1 as definition domain, and stochastic determination of x_{ij} as :

$$P(x_{ij} = 0) = 1 - p_{ij} = q_{ij},$$

$$P(x_{ij} = 1) = p_{ij}$$

In order to find the terminal reliability, first all the simple paths between terminal nodes are obtained as terms in Boolean sum-of-products function (SOP) F . Secondly the disjoint terms are obtained and the desired Boolean function F is determined. The terminal reliability P is then straight forwardly computed by means of the following correspondence.

$$x_{ij} \longrightarrow p_{ij},$$

$$\bar{x}_{ij} \longrightarrow q_{ij}$$

Boolean sum _____ Arithmatic sum

Boolean product _____ Arithmatic product

Simple paths of the probabilistic graph can be obtained by the technique described in Section 5.2. However, some arcs in the graph may not have any orientation. These arcs can be represented by transitions which can fire in both the direction. Hence, $A_1 = [a_{ij}]$ of eq.(5.1) is modified with elements a_{ij} as

$$a_{ij} = \begin{cases} -1, & \text{if arc } j \text{ is oriented away from node } i \\ 1, & \text{if arc } j \text{ is oriented towards node } i \\ \pm 1, & \text{if arc } j \text{ is not oriented but is incident at } i \text{ th node} \\ 0, & \text{otherwise} \end{cases}$$

Further, the extra addition rules of the columns are imposed as $(\pm 1) + (\pm 1) = 0$; $-1 + (\pm 1) = 0$; $1 + (\pm 1) = 0$.

For illustration, consider the following example.

Example 5.1: For the graph G of Fig.5.1 [39] all the simple paths between terminals 1 and 4 are to be obtained.

Using the technique of Section 5.2.2 with modified A_1 and extra addition rules we proceed as follows:

Step 1

$$A_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} x_{12} & x_{13} & x_{23} & x_{24} & x_{34} \end{array} \\ \begin{bmatrix} -1 & -1 & 0 & 0 & 0 \\ 1 & 0 & \pm 1 & -1 & 0 \\ 0 & 1 & \pm 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

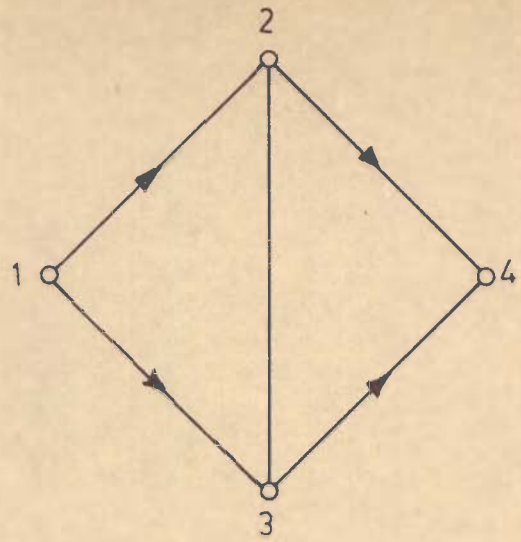


FIG. 5.1_ A PROBABILISTIC GRAPH

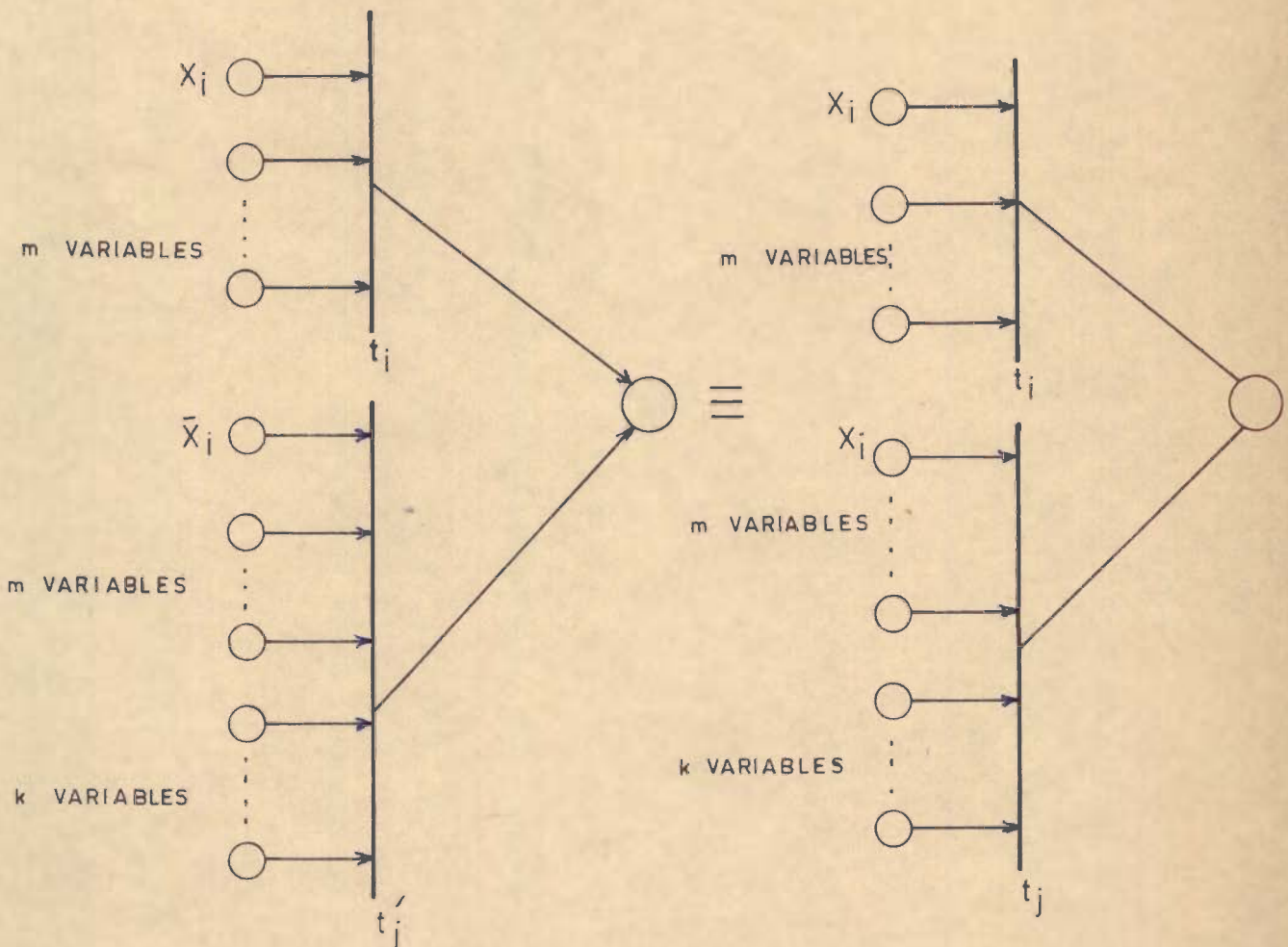


FIG. 5.2 _ PN PRESENTATION OF SIMPLE PATHS WITH CONCURRENCY AND CONFLICT IN TRANSITIONS

and

$$\Delta M = \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Step 2 : $S_1 = (x_{12}, x_{13})$; $S_4 = (x_{24}, x_{34})$

Step 3 : It is found that none of the columns equals ΔM, hence there is no path of length 1.

Step 4,5,6 : $L = 2$. It is found that addition of column corresponding to x_{12} and x_{24} equals ΔM giving thereby a path of length 2 as $(x_{12}x_{24})$. Other simple path of length 2 is $x_{13}x_{34}$.

Step 7 : Similarly simple paths of length 3 are $(x_{12}x_{23}x_{34})$ and $(x_{13}x_{23}x_{34})$. There is no other simple path.

Hence, simple paths are $x_{12}x_{24}$, $x_{13}x_{34}$, $x_{12}x_{23}x_{34}$ and $x_{13}x_{23}x_{24}$. The Boolean function (SOP form) f is thus given by

$$F = x_{12}x_{24} + x_{13}x_{34} + x_{12}x_{23}x_{34} + x_{13}x_{23}x_{24}.$$

5.3.2 DETERMINATION OF DISJOINT TERMS OF F AND PROBABILITY

Each term of F can be represented by a transition whose inputs are the logic variables of that term. The number of transitions will be equal to the number of terms in F. For the two terms t_i and t_j, the PN representation of F is shown on the right hand side of Fig.5.2.

The $m(m \geq 1)$ variables are common to both the transitions. The variables x_i is the input only to t_i and k variables (not including x_i) are input only to t_j. Obviously, t_i and t_j are not disjoint because both the transitions can fire simultaneously. It can be seen easily that as far as the availability of output is concerned, left hand side and right hand side of Fig.5.2 are equivalent. But t_i and t_j' can not fire simultaneously making the transition disjoint.

It may be noted that one of the transitions corresponding to a term in function f is to be considered as a basis to find the disjoint transitions. In order to reduce the number of computations the transition in which input contains minimum number of variables is selected as the basis. Let such a transition be t_i and one of the inputs to it be x_j. Any transition which has one of the inputs x_j will be disjoint with t_i. Consider another transition t_k $(k \neq i)$. Three possibilities can exist for the inputs to t_k,

i) variable is present as x_j

ii) variable is present as $\bar{x}_j$, and

iii) variable x_j is absent.

For case (i) obviously t_k is not disjoint to t_i. However, because the simple paths are different, there will be atleast one variable which is input to t_i, and not to t_k and t_k is made disjoint to t_i by adding complement input of that particular variable to t_k.

Since the function F contains only uncomplemented terms, case (ii) will not be present in the beginning but in subsequent computations this situation may arise. In this case, t_k is disjoint to t_i.

For case (iii) an input $\bar{x}_j$ is added to t_k,

Hence, for every input variable to transition t_i all the transitions disjoint to t_i are obtained. Ignoring t_i, consider the remaining transitions. Obviously they may not be disjoint to each other. The above process is then repeated.

A procedure based on above discussion to obtain all disjoint terms from simple paths is as under:

Step 1: Represent all simple paths in tabular form where a row corresponds to a path and a column to a variable. If a variable j is present in a path i, then the entry for (i,j) in table is 1. Otherwise the entry is * (where * denotes the absence of the variable in the path)

Step 2 : Select a row in the table for which maximum number of *'s appear. Call it a basis row. If any row does not contain any * , then it does not take part in further computation but corresponds to one of the terms in disjoint function.

Step 3 : For the basis row, select a coloumn j of no * entry. Determine the complement of the entry, say C_j. Obtain a new table whose entries in columns other than j th column for the remaining rows are same as in the table obtained in Step 1. The entry for j th column and i th

row is determined by multiplying the entry (i,j) of the preceeding table by C_j according to following rule,

0.0 = 0

1.1 = 1

1. * = 1

0. * = 0

However, for any row i and column j, if the multiplicant is complement of C_j, the i th row is not entered in the new table.

Repeat the step 3 for all no. * entries of the basis row and add rows to the new table. Note that the new table does not contain the basis row.

Step 4 : Go to step 2 and repeat till there is only one row in the new table.

Step 5 : Find conjunctive functions for each basis row (including that of final table) and the rows without any * by putting the uncomplemented variables for '1' and complemented variables for '0' in corresponding columns. The function of all disjoint terms is, thus, the disjunction of all conjunctive functions.

The above procedure is illustrated with the help of example of Fig.5.1.

Step 1 : Table 5.1 is obtained as

Variables		x_{12}	x_{24}	x_{13}	x_{34}	x_{23}	
rows	1	1	1	*	*	*	Basis row
	2	1	*	*	1	1	
	3	*	*	1	1	*	
	4	*	1	1	*	1	

Table 5.1

Step 2,3 : Arbitrarily row 1 is selected as basis row(another choice is row3)and $C_{12} = 0$, $C_{24} = 0$.

Considering first $C_{12} = 0$ and then $C_{24} = 0$, table 5.2 is obtained as

Variables	x_{12}	x_{24}	x_{13}	x_{34}	x_{23}	
rows	0	*	1	1	*	
	1	0	*	1	1	
	*	0	1	1	*	Basis row

Table 5.2

Step 4,5 : The table 5.2 does not contain only one row and hence repeating step 2, the row 4 of table 2 is the basis row and $C_{24} = 1$, $C_{13} = 0$, $C_{34} = 1$, new table 5.3 is obtained.

Variables	x_{12}	x_{24}	x_{13}	x_{34}	x_{23}	
rows	1	0	0	1	1	One of the solution
	0	1	1	1	*	
	0	1	1	*	1	Basis row

Table 5.3

Proceeding in a similar way the following final table consisting of the three basis rows and the fully assigned rows gives the required solution.

Variables	x_{12}	x_{24}	x_{13}	x_{34}	x_{23}
rows	1	1	*	*	*
	*	0	1	1	*
	1	0	0	1	1
	0	1	1	1	0

Table 5.4

Therefore, the function **F** is given by

$$F = x_{12}x_{24} + \bar{x}_{24}x_{13}x_{34} + x_{12}\bar{x}_{24}\bar{x}_{13}x_{34}x_{23} + \bar{x}_{12}x_{24}x_{13}x_{23} + \bar{x}_{12}x_{24}x_{13}x_{34}\bar{x}_{23}$$

From the correspondence of the variables with probability, the total probability P is given by

$$F = p_{12}p_{24} + q_{24}p_{13}p_{34} + p_{12}q_{24}q_{13}p_{34}p_{23} + q_{12}p_{24}p_{13}p_{23} + q_{12}p_{24}p_{13}p_{34}q_{23}.$$

It can be seen that the result is not the same as obtained in [39]. The difference is because of the selection of the basis

rows. However, the solution obtained by the proposed technique will be more nearer to the optimal solution because the basis rows are selected such that minimum number of no * entries are in the row.

5.4 PROGRAM COMPLEXITY EVALUATION

Testability and maintainability are two important aspects of software systems. A technique for program modularization such that the resulting modules are testable and maintainable, has already been adopted [87]. This is based on program complexity - a quantitative index of the structural properties of programs. During the creation of software modules, the complexity is calculated. Lesser it is, easier is the program debugging. This puts a restriction on program complexity by setting an upper limit. When this limit exceeds, either subfunctions are recognized and modularized or the software is redesigned. Another significant application of complexity is its use for developing test strategies and selecting test datas.

The measures of program complexity can be divided into two groups:

1. The complexity of program control. Here basic paths, cyclomatic number, path lengths and number of paths have been defined as complexity metrics [87], [134].

2. Number of basic operations or execution time to compute the program result [79].

5.4.1 COMPLEXITY METRICS

Complexity metrices can be thought of as a measure of 'psychological' complexity in the sense that the lower value of this gives a confidence in testability and maintainability of software systems. On the other hand the execution time is a measure of performance complexity.

The complexity metrics in program control is defined with reference to a directed graph which is associated with the given program. Each node in the graph corresponds to a block of code in the program when the flow is sequential and the arcs correspond to branches taken in the program. The graph has unique entry and exit nodes. In the following paragraphs different complexity measures and their importance are discussed.

Cyclomatic number, in a strongly connected graph G is the maximum number of linearly independent circuits [87]. This corresponds to the number of program constructs which are the basic units to be tested.

Basic paths are those when taken in combination will generate every possible path in the program. As any program with a backward branch potentially has an infinite number of paths [87], testing of all possible paths is impractical. Therefore, for correct execution of the program only basic paths are considered for testing.

Number of paths and path lengths are important for debugging and maintainance. If a large number of paths exists, then the program will be difficult to debug and maintain, and should

be divided into smaller modules. Further, shorter is the path length, lesser is the difficulty encountered in testing.

There exists in literature an algorithm for the evaluation of complexity metrics [134]. This involves the following automated steps : (a) drawing a directed graph as the program is written, (b) identifying the graph tree and determining fundamental circuits by adding at a time a link to the tree, (c) generating the ringsum from the rows of fundamental circuit matrix and determining paths of interest with the help of this and adjacency matrix and its powers, and (d) identifying or computing complexity matrics from (a) and (b).

In this section a new complexity metric namely simple paths between entry and exit nodes and independent directed circuits is defined. It is shown that these two can evaluate all the complexity measures defined in the preceeding paragraph. Further, a PN approach is applied to evaluate these.

A simple path is an open path in which an edge appears only once and no node is traversed more than once. Each simple path between the entry and exit nodes of the graph will form a fundamental circuit if these two nodes are connected for the purpose of making the graph strongly connected. For illustration purpose consider Fig.5.3 where node 1 and 5 are entry and exit nodes, respectively. The dotted line between nodes 5 and 1 is to make the graph strongly connected i.e. there exists a path joining any pair of arbitrary distinct vertices. The simple path are 1245, 135 and 1635. Corresponding to simple paths the

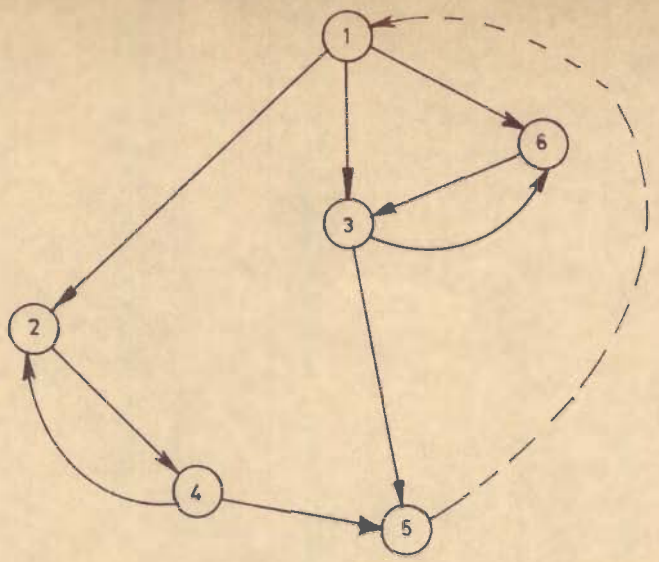


FIG. 5.3 _ A PROGRAM GRAPH

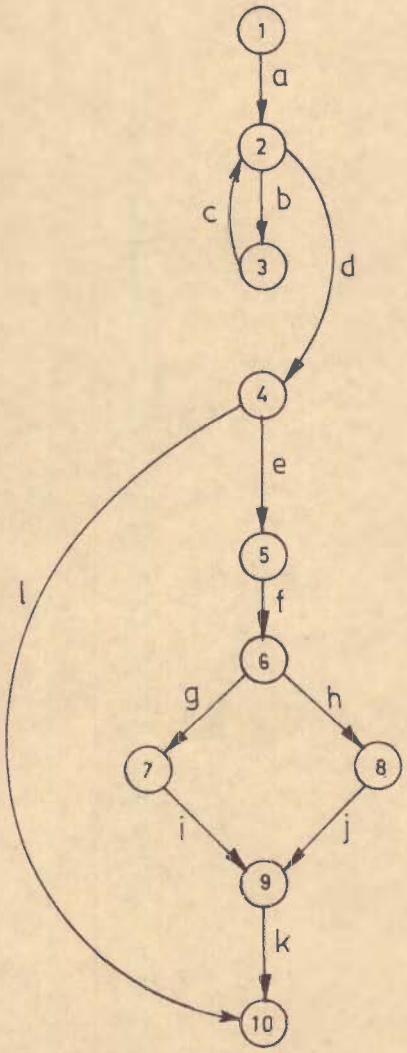


FIG. 5.4 _ THE GRAPH FOR ALGOL PROGRAM IN FIG. 5.5

fundamental circuits are 12451, 1351, and 16351. Obviously there will be some directed circuits not included in any of these. In the example, 242 is such a circuit. Hence, it is clear that the total number of fundamental circuits will equal the number of simple paths between the terminal nodes added with the number of directed circuits in the graph without providing any feedback path. In the example under consideration, directed circuits are 242, and 363. Hence, the cyclomatic number is $3 + 2 = 5$.

It can easily be shown further that all the paths can be obtained by a linear combination of simple paths and directed circuits. For the example, the set B is a basis of program paths as set of simple paths and dicircuits given above will generate path 13635 as $13635 = 135 + 363$. Obviously the basic units to be tested are simple paths and dicircuits. The length and number of the simple paths will play an important role in testing. Smaller the length and fewer the number, easier it is for debugging and maintainability. Same is true for dicircuits.

Having given the importance of the new complexity measure, the next question is how to evaluate it. Generation of simple paths have been discussed in 5.2. The same algorithm can be applied to program graph without feedback loop. The enumeration of dicircuits in the graph is implicitely included in the algorithm. The incidence matrix of a graph is its structural property and will not change for the two enumerations. However ΔM will be different in the two cases. Consider a directed circuit and transform it into a PN as in section 5.2. If the

token put on any place p moves around many places after successive single firing of transitions and finally arrives at p, then it has travelled a directed circuit. Here the change in marking ΔM is Zero. Therefore, a directed circuit is obtained by solving for Σ the equqtion (5.1) i.e.

$$A_1 \Sigma = \Delta M \qquad (5.1)$$

where $\Delta M = 0$. In the algorithm of single paths (Section 5.2), if ΔM in steps 3 and 5 is 0, then the solution Σ is a dicircuit. The other iterations are same in both the cases. As an illustration let us consider the same ALGOL procedure and corresponding program graph in [134] reproduced in Fig.5.5 and Fig.5.4, respectively. The nodes and the edges of the graph correspond to the circled number and statements between them respectively.

From the definition of incidence matrix A_1, it is obtained as

$$A_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} a & b & c & d & e & f & g & h & i & j & k & l \end{array} \\ \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

```
                PROCEDURE TEST CONDITIONS;
                COMMENT TEST ALL CONDITIONS FOR MEMBER IDENTIFIED
                BY CURRENT MODE;
①
                COMMENT IF ALL CONDITIONS HOLD ADD MEMBER TO
                LINKED TEST;
                   BEGIN
                   INTEGER A,I;
                   LOGICAL FAIR;
                   FAIR : = TRUE;
                   I  : = 1;
②               WHILE ((REQUEST(I) = "Q") AND (FAIR = TRUE) DO
                   BEGIN
                   FAIR : = MATCHING(I);
                   I : = I +1;
③                  END
   ④            IF FAIR = TRUE THEN
⑤                  BEGIN
                   A : = ALLOCATE;
   ⑥               IF LIST POINTER = NIL THEN LIST_POINTER: = A
                                          ⑦
       ⑧           ELSE SETCDR 1 (LAST, A);
                   LAST: = A;
                   SETCDR1 (LAST, NIL);
                   SETCAR1 (LAST, CDR2 (CURRENT_MODE+1));
⑨                  END
          ⑩        END TEST_CONDITIONS;
```

Fig. 5.5 - An ALGOL Procedure

For simple paths, $\Delta M = [-1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1]^T$

and, for dicircuits, $\Delta M = [\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$

Using the technique of Section 5.2

Simple paths are obtained as

1. adl of length 3
2. adefgik of length 5
3. adefhjk of length 5

and only one dicircuit be obtained.

Hence, the cyclomatic number is 3 + 1 = 4 which tallies with the result given in [134].

5.4.2 EXECUTION TIME [138]

Kodres [79] defined a new measure of complexity, execution time of a program and proposed a method to evaluate it. In his algorithm unlike traditional representation of program flow charts by directed graphs, the concept of flow graph is used. The sequence of functional statements are represented by arcs and control points in program as vertices. Kodres after showing similarities between problem arising in programming, discrete system analysis in engineering and network flow problem arising in operation research, applied the relationship between tree branches and links to find an execution time expression.

The object of this section is to highlight the application of Petri nets to evaluate execution time of a program. Since each block of a program flow chart represents either some operation or some condition, it can be represented either by a transition

or a place. The complete flow chart which is made up of several blocks can, therefore, be represented by a PN. A detail discussion of such a transformation has already been discussed in Section 2.4 on software modeling.

In conventional Petri nets, it is assumed that the output of a transition after firing is immediate and that the firing does not take any time. Deviating from this conventional approach we shall introduce a concept of time to firing of each transition. Let us assume that a transition t_i requires T_i time to execute the operation which it represents. This means that the output of the transition t_i will appear after a time T_i of starting of firing of t_i. Hence in order to find the total execution time for a program we must determine how many times each of the transition has to fire. The total execution-time is, thus, given by

$$\text{Total execution-time} = \sum_{i=1}^{t} \sigma_i \, T_i \qquad (5.2)$$

where, σ_i is the number of times the transition t_i fires in the execution of the program. Note that σ_i is an element of the firing count vector. Therefore, the computation of total execution time can be obtained by solving for Σ the state eq.(2.3) (also given in the preceeding section) and then using eq.(5.2). Since the initial and final conditions of flow chart are known, ΔM can be obtained and (2.3) can be solved for positive integer-valued solution.

As an example, consider the flow chart of [79] shown in Fig. 5.6 and represented by PN in Fig. 5.7.

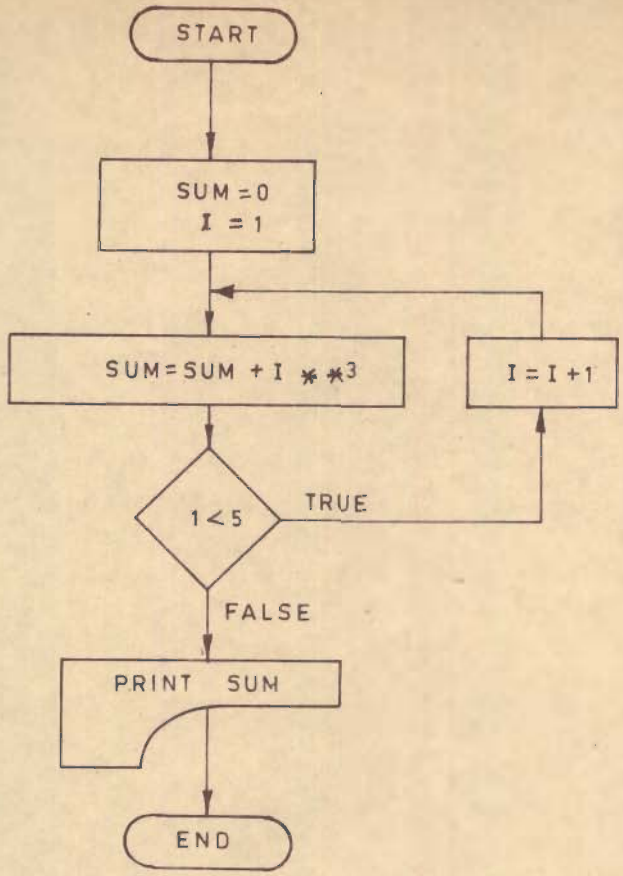


FIG. 5.6 _ A FLOW CHART

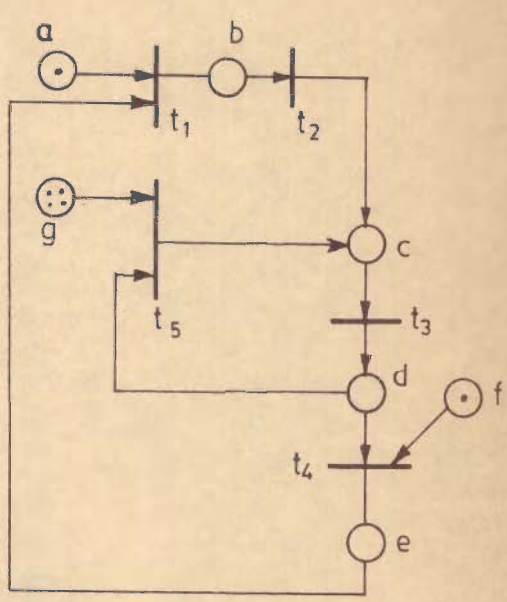


FIG. 5.7_ PN REPRESENTATION OF FLOW CHART IN FIG. 5.6

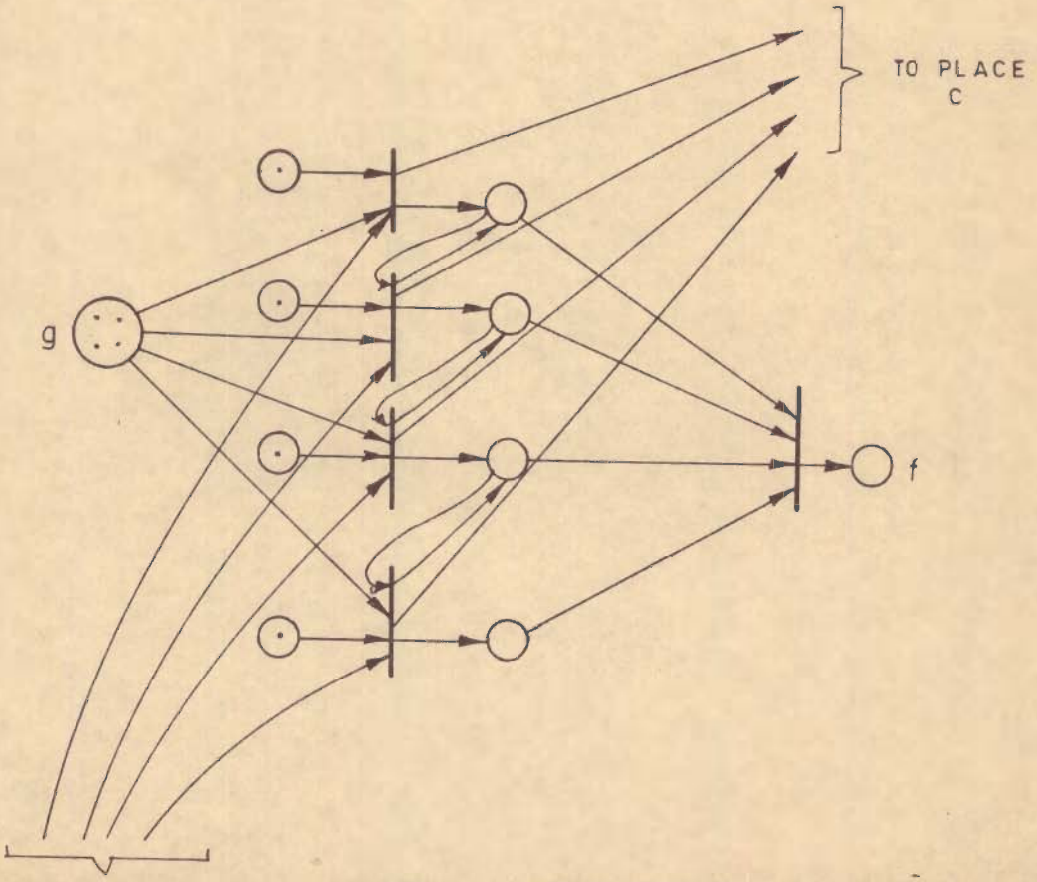


FIG. 5.8 _ A REPRESENTATION OF t_5 AT LOWER LEVEL OF ABSTRACTION

The transitions corresponding to the sequences of operations are as follows:

$t_1 \sim$ START

$t_2 \sim$ SUM = 0, I = 1

$t_3 \sim$ SUM = SUM + I * *3

$t_4 \sim$ PRINT SUM

$t_5 \sim$ I = I + 1

The tokens at places correspond to various conditions. The token at place 'a' means a request for start is made. The program 'starts' means that firing of t_1 will take place only when a request for start is made and the printer is available. A token at 'e' represents that the printing is over and the printer is free. The place 'g' contions four tokens to represent I < 5. A token at 'f' represents, the condition I < 5 False. This token will arrive at f only after the transition t_5 has stopped firing. The arrival of token at f can be obtained by Petri net shown in Fig.5.8. However, this net is represented in Fig.5.7 in a modified way by a single transition t_5. The advantage of PN to represent different levels of abstraction may be noted here. By having a single transition t_5 with input place g, complexity of net is reduced. Then,

$$A = \begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 1 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & -1 \end{bmatrix} \end{array}$$

$$M_o = \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{matrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 4 \end{bmatrix}, \text{ and } \quad M_n = \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

$$\text{Hence } \Delta N = M_n - M_o = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ -4 \end{bmatrix}$$

Solving eq.(2.3) for Σ,

$$\Sigma = \begin{matrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{matrix} \begin{bmatrix} 1 \\ 1 \\ 5 \\ 1 \\ 4 \end{bmatrix}$$

Hence, total execution time = $T_1 + T_2 + 5T_3 + T_4 + 4T_5$

The result tallies with that of [79].

5.5 CONCLUSION

In this chapter, the application of Petri nets to some of the problems of computer systems has been described. It has been

shown by demonstrating that many computer hardware and software problems could be represented by different classes of Murata's state equation of PN the solution techniques of which are essentially the same. The state equation, thus, forms the nucleus.

A technique for solving the state equation has been given in the procedure to enumerate all simple paths between two nodes of a graph. Neverthless the same technique could be used, as demonstrated, for the problems where ΔM is different than for simple paths. The technique is novel in the sense that only a single matrix is needed. Also, it is easily adoptable on computer. Moreover, only vector additions are required and no matrix multiplication is involved. This considerably reduces the computational time. The complexity of the computation has also been calculated in terms of maximum iterations needed.

The problem of computation of the terminal reliability of a computer network has been solved by finding, first a Boolean function in terms of simple paths whose enumeration is through the state equation, and then its disjoint terms. Obviously the advantages of the proposed technique over the existing ones are the same as given in the above paragraph. Since the basis rows in finding the disjoint terms are selected with minimum number of literals, the solution obtained will require lesser terms and, therefore, will be more nearer to the optimal solution compared to other techniques.

In the problem of program complexity evaluation, presented are simple paths and dicircuits as a measure of 'psychological'

complexity and execution time as a measure of performance complexity. The advantages of simple paths and dicircuits as complexity measure have been highlightened. Their enumeration has been achieved by the technique of solution of state equation of PN. Unlike the existing techniques [134], no tree, no generation of ring-sums over the rows of fundamental circuit matrix and no adjancency matrix alongwith its powers are needed. The incidence matrix A_1 of the graph which is used in the state equation contains all the necessary information. The degree of vertex as an indicator of program complexity is a direct evaluation from the matrix A_1. The other complexity measures have been shown to be directly obtained from the state equation.

CHAPTER VI

PETRI NET APPROACH TO DEVEPLOMENT OF MICROPROGRAMMED COMPUTER

6.1 INTRODUCTION

Microprogramming is one of the important aspects of modern day computers. Apart from increasing the flexibility it has made possible the process of emulation. Microprogram optimization is, however, necessary to be employed to increase the efficiency for the applications of microprogramming. Of various optimization strategies adapted, only bit optimization in the design of control memory and data path optimization in the modular interconnection of systems have practical advantages. These have been reviewed in Chapter III. This chapter investigates the role of Petri nets in the development of systems employing microprogramming particularly with reference to the above two optimizations. The algorithms to solve these two optimizations are also proposed.

6.2 BIT OPTIMIZATION

The problem of bit optimization in the control memory of a microprogrammed computer is essentially the grouping of microcommands such that a minimum number of encoding bits is required without loosing the parallelism in microcommands. The existing techniques and the related concepts were presented in Section 3.3.1. An important conclusion of the research in this field is

that the problem is NP-complete. Therefore, instead of finding a general solution, the attempt should be made in solving reasonable subcases. It is in this regard that a technique for bit optimization is proposed here. Given a ROM specification, the strategy involved is first to enumerate all the maximal compatible classes (MCCs) of microcommands and then to place microcommands in blocks such that

i) each block is a subset of an MCC

ii) no microcommand can be added to any block without increasing the bit required to encode the block i.e. each block contains exactly $(2^{n_i} - 1)$ microcommands (one place is fixed for No operation N_{op} where $n_i = 1,2,\ldots,n_{max}$. The value of n_{max} is decided by the number of microcommands in the largest MCC

iii) there are maximum possible number of blocks with $(2^{max} - 1)$ microcommands, maximum possible number of blocks, with $(2^{max-1} - 1)$ of the remaining microcommands and so on.

Obviously, such an arrangement, not necessarily, yields an optimal bit solution. Hence, a condition for bit reduction is obtained. A procedure is given and illustrated with the help of an example.

6.2.1 ENUMERATION OF MAXIMAL COMPATIBLE CLASSES OF MICROCOMMANDS [74]

As pointed out in Section 3.3.1, the enumeration of maximal compatible classes (MCCs) of microcommands has been

obtained using methods suggested in minimization of sequential machines [22], [80]. However, since these methods require the construction of compatibility charts and graphs, they are not easily implemented on digital computers. This provided a motivation for an alternative technique which is presented in this section. In this technique MCCs are obtained by recasting the given description of ROM into PN representation and then solving the state equation of the PN. A reference of this was made in 5.1.

6.2.1.1 FORMULATION

Each word of a ROM can be thought of as represented by interconnections of transitions and a place of a PN. The microcommands are represented by transitions, and the word by an input place. As two microcommands appearing together in any word are incompatible, no two corresponding transitions can fire simultaneously. This is obtained by putting a token on the place. A transition fires if its input place contains exactly one token and on firing, it removes the tokens from input places. Obviously, the complete ROM can be represented by such interconnection of places and transitions obeying the above firing rule. An MCC of length L is a set of L microcommands no two of which belong to one single word and to which no microcammands can be added without violating this constraint. Hence, to enumerate all the MCCs it is sufficient to determine L transitions of the corresponding Petri net which must fire in order that the tokens from the input places are removed. This is achieved by solving for Σ the following Murata's state equation described in Section 2.6 for a PN

$$A^T \Sigma = \Delta M \qquad (2.3)$$

Since, the transitions are fired to remove the tokens from the places, the change in marking ΔM in this case is given by $\Delta M = -U$ where U is $W_1 \times 1$ unit vector. Eq.(2.3) is, therefore, modified to

$$A_1 \Sigma = U \qquad (6.1)$$

where $A_1 = [a_{ij}]_{W_1 \times N_c}$ is the input connection matrix of ROM having W_1 irredundant words and Nc microcommands. The elements a_{ij} are given by

$a_{ij} = 1$ if jth microcommand is contained in ith word
$= 0$ otherwise

and Σ is a $N_c \times 1$ column vector whose element is 1 if jth microcommand is included in an MCC, 0 otherwise.

6.2.1.2 ENUMERATION PROCEDURE

Eq.(6.1) contains W_1 independent equations in Nc unknowns and each feasible solution Σ will correspond to one MCC. The solution technique of Chapter V can be directly applied to solve the equation. Instead, another technique which requires fewer enumerations is proposed here. This takes advantage of the property that the elements of A_1 in this case, are either 1's or 0's. No two columns having 1's in one row can be added to equate unit vector. Let the matrix A_1 be rearranged in columns such that first k_1 columns correspond to the microcommands of the largest instruction, the next k_2 columns correspond to the remaining microcommands of the largest remaining instruction and

and so on. Now A_1 can be partitioned as

$$A_1 = [A_{11} \vdots A_{12} \vdots \cdots\cdots \vdots A_{1s}]$$

where A_{1i} is $W_1 x k_1$ matrix. Obviously there is atleast one row in A_{1i} which contains all 1's. Therefore, columns of A_{1i} cannot be added to give the solution. For solving (6.1) the columns of A_{1i} and A_{ij} for $j \neq i$ are to be added. To find the maximum iterations, let us consider that columns of A_{11} are added to columns of A_{12}. This will require $k_1 \cdot k_2$ additions. However, for comparing with U only those added columns which donot have any entry more than 1 are considered. For three submatrices A_{11}, A_{12} and A_{13} the number of such iterations will be $(k_1k_2 + k_1 + k_2)k_3$. Proceeding in the same fashion, the maximum number of additions required to solve (6.1) will be

$$\underbrace{(\ldots((((\ldots\ldots\ldots(k_1k_2+k_1+k_2)k_3+k_3)k_4+k_4)\ldots\ldots+k_{s-1})k_s}_{(s-2 \text{ brackets})}.$$

Based upon the above discussion the following steps will find all MCCs.

1. Given a ROM description, delete the words with micro-commands which are contained in other words.
2. Determine $A_1 = [a_{ij}]$ such that $a_{ij} = 1$ only if jth microcommand is present in ith word, 0 otherwise
3. Partition A_1 by rearranging the columns as

   $A_1 = [A_{11} \vdots A_{12} \vdots \cdots\cdots \vdots A_{1s}]$ such that

   $k_1 \geq k_2 \geq \cdots\cdots \geq k_s$ where k_j is the number of 1's which all the columns of A_{1j} have for atleast one row.

4. Put L = 2
5. Add L columns, taking 1 column each from $A_{11}, A_{12}, \ldots, A_{1s}$ such that no two of them belong to any one of deleted words. If addition equals unit vector, then the set of microcommands corresponding to these columns gives MCC of length L.
6. Repeat step 4 for all possible combinations of L
7. Repeat step 5 and 6 for L = L + 1 until L = s.
8. Stop.

The above procedure is illustrated with the help of an example.

Example 6.1 : Let the description of ROM [135] and given in Table 3.1 be again considered. All the MCCs are to be obtained :

Word	Microcommands
1	a,b,c,d,e,f
2	c,g,h,i
3	a,b,h,i,j
4	d,h,k
5	f,h

Table 3.1

Using the proposed technique, the steps are given below.

Step 1 : The microcommands of word 5 are contained in the other words. Hence the word 5 will not be considered and,

Step 2,3 :

$$A_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccccccccc} a & b & c & d & e & f & g & h & i & j & k \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Step 4,5,6 : Two columns taking one from each portion only can be added but columns h and f can not be added because they belong to one deleted word 5. We get MCCs of length 2 as (d i) and (e h).

Step 7 : It results that MCCs of length 3 are (a g k), (b g k), (c j k), (d g j), (e i j) and that of 4 are (e g j k), (f g j k).

Having given a method to enumerate all MCCs, the procedure for bit optimization is given in the following section.

6.2.2 PROCEDURE

Let Nc, m and L_m be the number of microcommands in the ROM specification, the largest microinstruction and the largest MCC, respectively.

We can always select a largest number n_{max} such that the following inequality is satisfied

$$(2^{n_{max}} - 1) \leq L_{max} < (2^{n_{max}+1} - 1) \qquad (6.2)$$

Let $$N_{max} = 2^{n_{max}} - 1$$

The inequality (6.2) can be written as

$$N_{max} \leq L_{max} \leq 2N_{max} \qquad (6.3)$$

As the largest microinstruction contains m microcommands which are incompatible, we require atleast m blocks to accomodate all the microcommands. These blocks are called 'essential blocks'. Now it is required to have maximum number of disjoint blocks each with N_{max} microcommands. Since each essential blocks has one microcommand, (Nc -m) microcommands are to be accomodated in blocks each of which can have additional (N_{max} -1) microcommands. Hence, assuming that compatability permits, maximum number of N_{max}-element-blocks (g_1) is given by

$$g_1 = \left\lfloor \frac{n-m}{N_{max} - 1} \right\rfloor \qquad (6.4)$$

where $\lfloor x \rfloor$ is the largest integer $\leq x$

The remaining $Nc-m-g_1 (N_{max} -1)$ microcommands are to be adjusted in maximum number of blocks g_2 of $(2^{n_{max}-1} -1)$ microcommands. Since,

$$2^{n_{max}-1} -1 = \frac{(N_{max} + 1)}{2} - 1$$

$$g_2 = \left\lfloor \frac{Nc - m - g_1(N_{max} -1)}{\left(\frac{N_{max} + 1}{2}\right) - 2} \right\rfloor$$

Next maximum blocks of $(2^{n_{max}-2} -1)$, $(2^{n_{max}-3} -1)$,....$(2^{n_{max}-k} -1)$, ... elements are selected and microcommands are allocated until no microcommand is left. On the termination of this process we shall have $g_1, g_2, \ldots, g_k$, blocks of $(2^{n_{max}} -1)$, $(2^{n_{max}-1} -1)$,.... ...., $(2^{n_{max}-k} -1)$ microcommands respectively. The number of

bits (B) needed to encode these are, then, given by

$$B = g_{11} + \sum_{i=1}^{k} g_i \lceil \log(2^{n_{max}-i+1}) \rceil$$

or

$$B = g_{11} + \sum_{i=1}^{k} g_i (n_{max} - i + 1) \qquad (6.5)$$

where g_{11} is the number of blocks having one microcommand each.

$$g_k = \left\lfloor \frac{Nc-m- \sum_{i=1}^{k-1} \left[g_i \left(\frac{N_{max}+1}{2^{i-1}} - 2 \right) \right]}{\left(\frac{N_{max}+1}{2^{k-1}} - 2 \right)} \right\rfloor \qquad (6.6)$$

The process will terminate when either $\frac{N_{max}+1}{2^{k-1}} - 2 = 0$ (i.e. $k = n_{max}$) or all the microcommands have been allocated. Hence the maximum number of iteration equals n_{max}.

When we actually allocate the microcommands according to the method described, MCCs may be such that we may not have $g_1, g_2, \ldots, g_k$ disjoint blocks of $2^{n_{max}}-1$, $2^{n_{max}-1}-1, \ldots, 2^{n_{max}-k}-1$ microcommands, respectively. But the actual minimal possible number of such a block i maybe less than g_i for one or many i's. In general, $g_{i\,act} \leq g_i$ for $1 \leq i \leq k$. In such a case equation (6.5) serves as a lower bound for the number of bits and no matter what we do, the number of bits can never be further reduced. This will also give to what extent the bit optimization should be tried for good engineering reduction.

The allocation obtained in preceeding section is not necessarily a minimal bit solution. Therefore, the following

theorem is postulated.

Theorem 6.1 : If all the microcommands are allocated to maximum possible number of $(2^{n_{max}} - 1)$ microcommands blocks, followed by maximal possible numbers of $(2^{n_{max}-1} - 1)$ microcommands and so on, then the number of bits can be reduced if and only if microcommands are shifted in accordance with compatability, from any block j to any block i such that

$$N_i \geq N_j$$

$$\text{and,} \quad (L_i - N_i) \geq \frac{3}{4}(N_j + 1).$$

where L_i, N_i and N_j are number of microcommands in MCC associated with i th block, number of microcommands allocated to i th and j th block, respectively.

Proof : Let us consider two blocks i and j containing N_i and N_j microcommands respectively. Let L_i and L_j be the number of microcommands in the MCCs associated with i and j block, respectively. Let $N_i > N_j$

(a) Let microcommands be shifted (assuming compatibility permits) for i th block to j th. The maximum number of shift could be $(L_j - N_j)$ because no more than L_j microcommands can be accomodated in j th block. Hence the number of microcommands in i th block after shift is $N_i - (L_j - N_j)$.

From inequality (6.3), maximum $(L_j - N_j) = N_j$ and minimum $(L_j - N_j) = 0$.

Hence, $(N_i - N_j) \leq N_i - (L_j - N_j) \leq N_i$

or $(2^{n_i} - 2^{n_j}) \leq N_i - (L_j - N_j) \leq (2^{n_i} - 1)$

Since $n_i > n_j$ (because $N_i > N_j$)

$$2^{n_i} - 2^{n_j} \geq (2^{n_i-1} - 1)$$

Therefore,

$$(2^{n_i-1} - 1) \leq N_i - (L_j - N_j) \leq (2^{n_i} - 1)$$

The minimum number of bits to encode $N_i - (L_j - N_j)$ microcommands, thus, equals $\lceil \log_2(2^{n_i-1} - 1 + 1) \rceil$ or $(n_i - 1)$. In other words the maximum reduction in bits for encoding i th block microcommand is 1 .

Now the increase in number of bits to encode j th block microcommand

$$= \lceil \log_2(L_j + 1) \rceil - \lceil \log_2(2^{n_j} - 1 + 1) \rceil$$

$$= n_j + 1 - n_j = 1 \qquad \text{because } (2^{n_j} - 1) \leq L_j < (2^{n_j+1} - 1)$$

Therefore, shifting microcommands from blocks with more microcommands to block with lesser number of microcommands will never reduce the bits.

(b) Let the microcommands be shifted to i th block. The maximum number of microcommands which could be accomodated in i th block is L_i. The increase in i th block is 1. These $(L_i - N_i)$ microcommands may come from either one block or more blocks.

i) Suppose these come from j th block. Then the reduction in 2 bits for j th block is possible if

$$(L_i - N_i) \geq N_j - \left(\frac{N_j + 1}{4} - 1\right)$$

$$\geq \frac{3(N_j + 1)}{4} \qquad \text{(A)}$$

ii) Suppose these come from two blocks each having N_j microcommands, then the reduction in 2 bits are from each block

is possible if

$$(L_i - N_i) \geq 2\left[N_j - \left(\frac{N_j+1}{2} - 1\right)\right] \geq (N_j+1) \qquad \text{(B)}$$

in both the cases (i) and (ii) the net reduction in bit is 1, but condition (A) gives a lower bound. Hence, if

$$(L_i - N_i) \geq \frac{3}{4}(N_j + 1)$$

then there is possibility of reducing the bits.

Q.E.D.

It may be noted that the essential blocks which contain only one microcommand must not be considered in the process of bit reduction.

Based upon the above discussion, given Nc, m and all MCCs, following are the steps for bit optimization:

1. Find L_i (the number of microcommand in i th MCC) for all i. Determine also the number of microcommand (L_{max}) in the largest MCC.
2. Find n_{max} such that
   $(2^{n_{max}} - 1) \leq L_{max} < (2^{n_{max}+1} - 1)$
3. Put $N_{max} = 2^{n_{max}} - 1$.
4. Determine maximum number of $(2^{n_{max}} - 1)$ element blocks (g_1); $(2^{n_{max}-1} - 1)$ element blocks (g_2) etc. from

$$g_1 = \left\lfloor \frac{Nc - m}{N_{max}-1} \right\rfloor, \text{ and}$$

$$g_K = \left\lfloor \frac{Nc - m - \sum_{i=1}^{k-1} g_i\left(\frac{N_{max}+1}{2^{i-1}} - 2\right)}{\left(\frac{N_{max}+1}{2^{k-1}} - 2\right)} \right\rfloor$$

5. Identify all the MCCs having microcommands $\geq N_{max}$

6. Identify the pairs c_i, c_j of MCCs obtained in step 5 covering the number of microcommands $\geq 2N_{max}$. If no such pair exists, form a block of microcommands of an MCC. Consider each disjoint MCC separately and go to step 8.

7. From the pairs of step 6, determine g_1 disjoint blocks of N_{max} microcommands. If it is not possible, find the maximal possible blocks (g_1') from different set of blocks.

8. Delete the microcommands contained in the blocks from all the MCCs. Call these reduced MCCs. Delete all the reduced MCCs which are subsets of other reduced MCCs.

9. Put
$$N_{max} = \frac{N_{max} + 1}{2^k} - 1, \text{ and}$$
$$g_1 = g_{1+k}$$

10. Repeat steps 5,6,7,8,9 for k = 1,2,.... until $N_{max} = 0$ or $k = n_{max}$ which is earlier.

11. The remaining microcommands are to be allocated to blocks of single microcommands.

12. Select a block j having microcommands ($N_j \geq 3$) for essential blocks and ($N_j \geq 1$) for nonessential blocks. For each of the block i ($\neq j$) containing microcommands $N_i \geq N_j$, test the condition $(L_i - N_i) \geq \frac{3}{4}(N_j + 1)$. If for every i and j the condition is not satisfied go to step 13, otherwise shift $(L_i - N_i)$ microcommands from j th block to i th block if compatibility permits. Without considering the blocks which have taken part in shifting operation, repeat for another

j's until no shifting is possible.

13. Determine number of bits (B) for the blocks obtained earlier from

$$B = \sum_{i=1}^{b} \lceil \log_2(|C_i|+1) \rceil$$

where $|C_i|$ is the number of microcommands in i th block and b is the number of blocks.

14. Stop.

6.2.2.1 DISCUSSION

Steps 1,2,3,5,8,9,10,11 and 13 are simple. Step 4 requires atmost n_{max} iterations which is much smaller than Nc and hence computations are fewer. The number of computations in the first iteration of step 6 will be $\frac{u(u-1)}{2}$ where u is the number of MCCs. In subsequent iterations, the number of MCCs will reduce and, thus, computations will be decreasing. Step 7 is nothing but the determination of the nodes of the largest complete polygon obtained from MCCs of step 6 and then finding the disjoint blocks of N_{max} microcommands.

In fact step 12 requires maximum computation and it is this step which is exponential in nature. However if (L_i-N_i) is small for many i then the number of computations are not too many. In other words the procedure proposed here will be very effective if '(L_i-N_i) is small for all or many i's'.

Example 6.2 For illustration purpose, let us consider the same running example 6.1. The optimal bit solution is to be obtained. MCCs obtained in 6.2.1 are

C_1 = (e g j k); C_2 = (f g j k); C_3 = (f i k); C_4 = (a g k);

C_5 = (b g k); C_6 = (c j k); C_7 = (d g j); C_8 = (e i k);

C_9 = (d i); C_{10} = (e h)

Using the procedure, we proceed as follows:

Nc = 11 ; m = 6

Steps 1,2,3 and 4 give

$L_1 = L_2 = L_{max} = 4$;

$L_3 = L_4 = L_5 = L_6 = L_7 = L_8 = 3$

$L_9 = L_{10} = 2$

$n_{max} = 2$; $N_{max} = 3$

$$g_1 = \left\lfloor \frac{11-6}{2} \right\rfloor = 2;$$

i.e. we shall have 2 blocks of 3 elements and 5 blocks of 1 element each.

Therefore $B_o = 2 \lceil \log_2(3+1) \rceil + 5 = 4 + 5 = 9$

Step 5,6 give pairs (C_1, C_3); (C_2, C_8); (C_3, C_7); (C_7, C_8)

Step 7 Since $g_1 = 2$, no largest complete polygon is needed. The disjoint blocks corresponding to each pair will give allocation of 6 microcommands to 2 blocks each having 3 microcommands. The disjoint blocks are obtained as:

1. (e g j), (f i k)
2. (f g j), (e i k)
3. (d g j), (f i k)
4. (d g j), (e i k)

Step 9,10,11 give the following allocation of microcommands:

1. (e g j), (f i k), (a), (b), (c), (d), (h)

2. (f g j), (e i k), (a), (b), (c), (d), (h)

3. (d g j), (f i k), (a), (b), (c), (e), (h)

4. (d g j), (e i k), (a), (b), (c), (f), (h)

Step 12. Consider allocation 1. Put $j = 1$, then $N_j = 3$. Only 2nd block ($i = 2$) contains 3 and L_2, N_2 corresponding to second block are $L_2 = 4$ and $N_2 = 3$. Hence $(L_i - N_i) = 4-3 = 1$. But $\frac{3}{4}(N_j+1) = 3$. Therefore, as $(L_i - N_i) \geq \frac{3}{4}(N_j+1)$, there is no possible reduction in bits. For $j = 2$, $i = 1$ again this condition is not satisfied.

Similarly we find that the condition for bit reduction is not satisfied for each of the above allotment. But in allocation 3, e and h can be combined without changing the number of bits and a solution (dgj),(fik),(a),(b),(c),(eh)is obtained.

Step 13. No. of bits needed to encode in each case equals 9.

The results obtained tally those in [47].

The list of all minimal bit solutions obtained are shown in Table 6.2.

1.	(e g j), (f i k), (a), (b), (c), (d), (h)
2.	(f g j), (e i k), (a), (b), (c), (d), (h)
3.	(d g j), (f i k), (a), (b), (c), (e), (h)
4.	(d g j), (e i k), (a), (b), (c), (f), (h)
5.	(d g j), (f i k), (a), (b), (c), (e h)

Table 6.2 - Minimum Bit Solutions for Sample Microprogram of Table 6.1

The previous section presented a method for bit optimization and obtained control fields as minimal bit solutions. These bits can further be reduced in number by employing the technique of bit steering [86]. This method has been outlined in Section 3.3.1.7. In the ensuing section an extended Petri net approach [9] for detection of bit steering is adapted. The strategy involved is to represent control fields by extended PN. The token distributions in places corresponding to microcommands are, then obtained so as to satisfy some properties and results interpreted.

6.2.3 BIT STEERING AND EXTENDED PN

To test for bit steering let C_1, C_2,....,C_n represent n control fields. Exactly one microcommand of each control field C_i is excited at a time to execute a microinstruction. The execution of all instructions of a ROM is, thus, a collection of such executions. Since we are interested only in bit optimization, the order of execution is not important. This activity can be very conveniently represented by an extended Petri net. To understand this let us consider Fig.6.1 which is a slightly modified version of extended PN [9]. The transition t is enabled to fire if there exists an i_1, $1 \leq i_1 \leq m_1$ with tokens in a_{1i_1}, $\# a_{1i_1} = 1$ AND there exists an i_2, $1 \leq i_2 \leq m_2$ with $\# a_{2i_2} = 1$ AND.....AND there exists an i_n, $1 \leq i_n \leq m_n$ with $\# a_{nm_n} = 1$. When t_1 fires all the tokens are removed from input places and one token is put to output place Z. From the above firing rule of transition t, it is clear that t is enabled to fire if any of a_{11}, a_{12},....,a_{1n_1} AND any of a_{21}, a_{22},...,a_{2n_2} AND.....AND

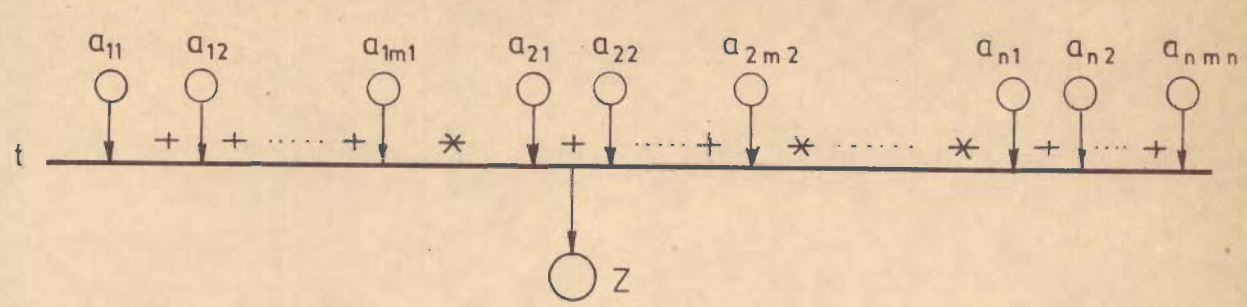


FIG. 6.1_ AN EXTENDED PN

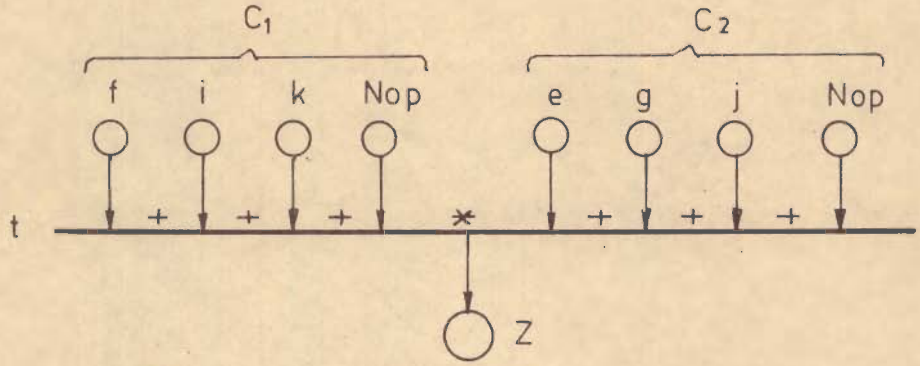


FIG. 6.2 a _ REPRESENTATION OF CONTROL FIELDS USING EXTENDED PN

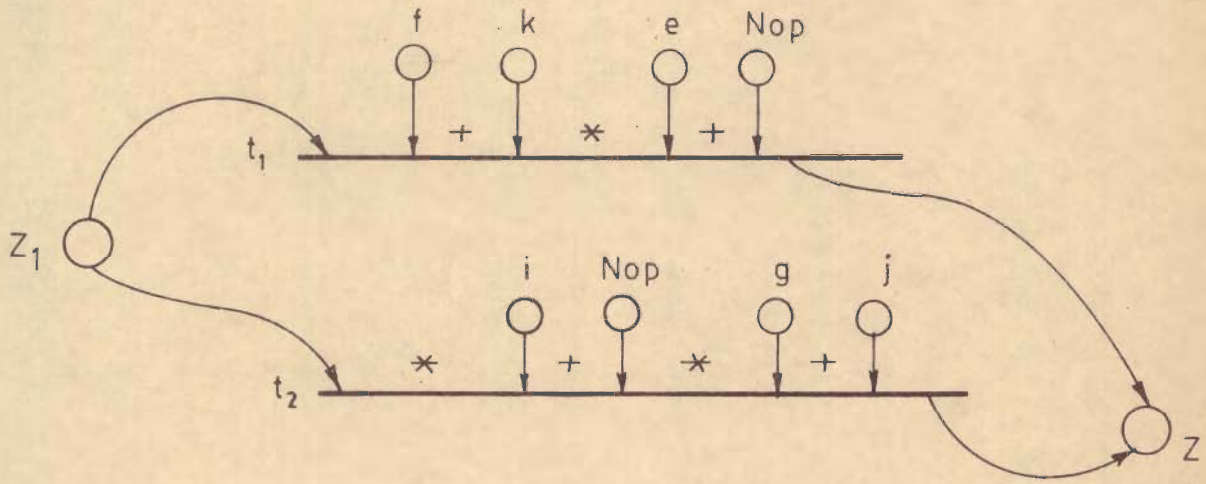


FIG. 6.2 b _ DECOMPOSITION OF TRANSITION t IN FIG. 6.2 a

any of $a_{n1}, a_{n2}, \ldots, a_{nm_n}$ contains a token. This is exactly the mechanism of the execution of a ROM word. Therefore, places $a_{i1}, a_{i2}, \ldots, a_{im_i}$ can be thought of as microcommands of a field C_i and each firing.

With a veiw to detect bit steering, only those control fields which need be considered, are represented by extended PN. As an example, consider solution 1 (Table 6.2) obtained for sample microprogramming of Table 6.1. As the maximum number of bits in steering field is one less the number of bits in the smallest amongst original fields, only two fields $C_1 = (f\ i\ k)$ and $C_2 = (e\ g\ j)$ are considered for bit steering and their representation by extended PN is shown in Fig.6.2a. Two places each corresponding to N_{op} operation are also shown because they are included in control fields in the determination of minimal bits. It may be noted that the total number of possible combinations of tokens in input places to enable t to fire is 16 but actual available combination is 5 i.e. as many as the number of words in ROM. This is simply due to missing combinations which are not forming any microinstructions. For example f and g do not occur in any microinstruction. Therefore tokens in f and g will not simulteneously appear.

Let t be decomposed in t_1 and t_2 as shown in Fig.6.2b. It is noted that t_1 and t_2 will not fire concurrently because only one of f, k, e and N_{op} can have a token. Further, no execution of any microinstruction is missing because k does not have any token at the same time g or j has, and i does not have token at the time e or N_{op} has (Table 6.1). Therefore, by decomposing

the net the token combinations are reduced to 8. The place Z_1 obviously corresponds to steering fields and is introduced in order to use the same token distribution of f, k for i and N_{op}, and that of e, N_{op} for g,j. The encoding bits to generate tokens for f,k and e, N_{op} are 1 each. One bit is also needed to generate token for Z_1. This makes total number of bits required as 3 rather than 4 for Fig.6.2a. It is important that Z_1 be present, otherwise same token distribution used for the elements of same field C_i appearing as input places to different transitions will not result in distinct excitation of microcommands. Obviously, the token distribution for maximum input places of C_i to any of the transitions must be considered. However, a point worth noting is that such a decomposition may not be possible in some cases. Hence, the above concepts are generalized and a theorem is postulated to test if suggested possibility exists.

Let n_i and n_s be the number of bits needed to encode field C_i and maximum steering bits respectively. Let n_t be the number of transitions into which the extended PN representing control fields considered for bit steering, is decomposed such that

i) each transition has places with minimum token distribution,

and ii) the places $p \varepsilon C_{ik}$ and $p' \varepsilon C_{j\ell}$ do not appear together in any ROM microinstruction for $k \neq \ell$ where C_{ik} is the subset of places of C_i appearing as input to k th transition.

Theorem 6.2 : The decomposition of extended PN representing fields $C_1, C_2, \ldots, C_n$ will result in lesser number of encoding bits if and only if

(a) $1 < n_t \leq 2^{n_s}$

and (b) $\forall\, i \in \{1,2,\ldots,n\}$, $p_{imax} < 2^{(n_i - n_s)}$ where p_{imax} is the maximum number of places of C_i appearing in any of the transitions.

Proof : (a) When $n_t = 1$, obviously no decomposition is possible. The number of bits required to ensure conflict amongst n_t transitions, is $\lceil \log t \rceil$ but the number of bits available for steering is n_s, therefore,

$$1 < n_t < 2^{n_s}$$

(b) The number of bits required for generation of tokens for p_{imax} places is $\lceil \log p_{imax} \rceil$. This will be maximum number of bits required for places of C_i appearing in any of the transitions. As the available bits are $(n_i - n_s)$, the reduction in bit is possible if

$$(n_i - n_s) > \lceil \log p_{imax} \rceil$$

i.e $$p_{imax} < 2^{(n_i - n_s)}$$

Q.E.D

Based upon the above theorem, the technique to test for bit steering and to encode steered sets, is given in the form of following steps:

1. Determine $n_i = \lceil \log(|C_i|+1) \rceil$ $\forall\, i \in \{1,2,\ldots,n\}$
2. Determine $n_s = \min(n_1, n_2, \ldots, n_n) - 1$
3. Find the token distribution matrix D, as

$$D = [d_{ki}]_{|P| \times M_1}$$

where,

d_{ki} = 1 if k th microcommand is included in i th microinstruction of the ROM

= 0 otherwise

4. Interchange the rows of D and partition the matrix such that first $|P_1|$ rows correspond to microcommands of C_1, followed by $|P_2|$ rows of microcommands of $C_2, \ldots, |P_n|$ rows of microcommands of C_n.

5. Put k = 1

6. (Decomposition and determination of minimum token distribution) Find a new matrix by OR ing the columns of D for which $m_{ki} = 1$. Retain the other remaining column.

7. Repeat step 6 for k = k+1, |P|

8. (Testing of bit steering using Theorem 6.2)
Find the number of columns of new matrix, n_t.
If $n_t = 1$ or M_1, steering is not possible. Go to step 12.

9. If $n_t > 2^{n_s}$, no reduction in bits is possible. Go to step 12.

10. Find the maximum number of 1s (p_{ima_x}) appearing in any of the columns of i th partition.
If $p_{ima_x} > 2^{(n_i - n_s)}$ for any i, no bit reduction is possible. Go to step 12.

11. (Encoding)

(a) All the microcommands having 1 in same column of M should be assigned same steering code.

(b) Microcommands having 1 in the same partition and 1st column must be encoded. Then encoding for 2nd column microcommand and so on is done.

(c) If there is any microcommand which has 0 in every coloumn, the encoding of such microcommand is done by assigning unused code combination.

12. Stop.

The number of iterations for ORing in step 6 will require $|P|$ times the number of search for 1's in a row. Hence, it is of the order of $O(|P| \log M_1)$. For step 7 the complexity of enumeration is, obviously, $O(n_t \log |P|)$.

The procedure is illustrated with the help of an example.

Example 6.3 : Let us consider solution 1 (Table 6.2) of the running example 6.1 for detection of bit steering. Following are the steps to do so.

1. For fields C_1 = (f i k) and C_2 = (e g j)
   $n_1 = n_2 = 2$
2. $n_s = \min(2,2)-1 = 1$

3,4.

$$D = \begin{array}{c} \\ f \\ i \\ k \\ N_{op} \\ e \\ g \\ j \\ N_{op} \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \left[\begin{array}{ccccc} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{array}\right] \end{array}$$

5,6. For k = 1, the new matrix is as follows:

$$\left[\begin{array}{cccc} 1,5 & 2 & 3 & 4 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{array}\right]$$

7. Repeating step 6 for k = 2,3,...,8, the new matrix is

$$\left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \\ \hline 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{array}\right]$$

8,9. $n_t = 2$ and $n_t > 2^{n_s}$

10. $p_{1max} = p_{2max} = 2$ and $p_{imax} > 2^{n_i - n_s}$

11. Let C be the common bit and B_1, B_2 are bits for elements of C_1 and C_2. Then

Step 11(a)

	C=1	C=0
f	1	0
i	0	1
k	1	0
N_{op}	0	0
e	1	0
g	0	1
j	0	1
N_{op}	1	0

;

Step 11(b,c)

		C=1	C=0	
B_1=1	f	1	0	
	i	0	1	B_1=1
B_1=0	k	1	0	
	N_{op}	0	0	B_1=0
B_2=1	e	1	0	
	g	0	1	B_2=1
	j	0	1	B_2=1
B_2=1	N_{op}	1	0	

.

The result tallies with that of [86]. The applicability of the proposed technique to 3 or more control fields is self evident. Only the number of partitions of D matrix will increase and the procedure remains essentially the same.

6.3 DATA PATH OPTIMIZATION [76]

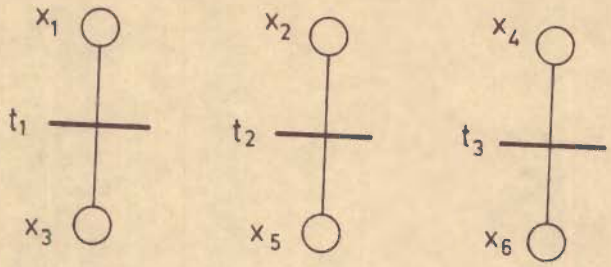
In the design of microprogrammed computers, a designer will often work out an overall hardware layout including registeres, logic units and allowed data paths, before writing the control program. It provides an opportunity in the very beginning to minimize the hardware by using same data paths. The number of data path also affects the flexibility in case of microprogramming a machine. The provision of several data paths results in high flexibility but prohibitively complex and costly implementation. On the other hand a single bus cannot be used if concurrent data transfers are taking place. Thus, a compromise is made between the two in data path optimization. These concepts have been

elaborated in Chapter III. The techniques for data path optimizationshave also been described. In this section the concept of Petri nets is exploited to solve this problem. Two approaches have been taken up here. These are as follows:

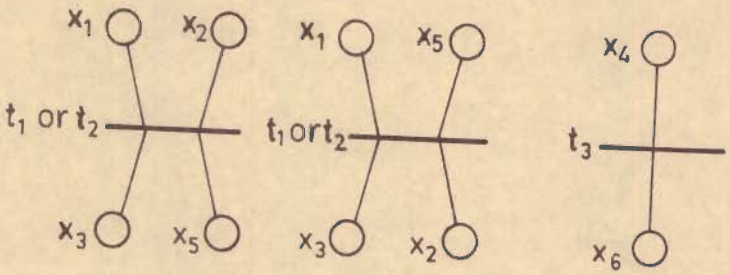
6.3.1 1- INVARIANT PN APPROACH

Given data transfers in the form of transfer matrix, this approach solves the problem of data path optimization only partially. It simply finds feasible solutions which are defined as solutions allowing concurrent data transfers with minimum number of buses regardless of the cost. From these, the generation of complete solutions is direct and can be obtained as in [85] the outline of which has been given in Section 3.4.3.

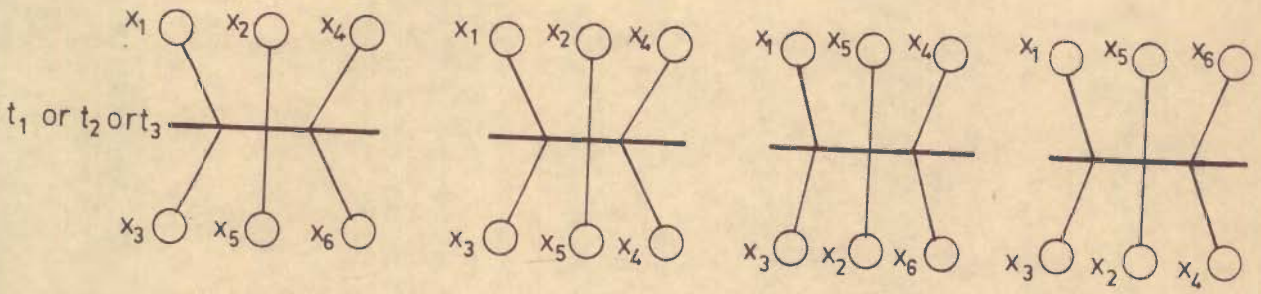
To find the feasible solution, the proposed approach starts with incompatible simulteneous sets of data transfers (ISTs). As defined in 3.4.3, an IST is a set of data transfer variables, no two of which can take place on the same bus. There appears to be a one to one correspondance of ISTs with 1-invariant PNs (Definition 2.18). A 1-invariant PN (1-IPN) is one in which the total number of tokens in any marking is 1. Therefore, the places of a 1-IPN can be thought of representing the data transfer variables of an IST. The number of transitions in 1-IPN is n-1 when n is the number of elements in the corresponding IST. For example, the ISTs $\{x_1, x_3\}$, $\{x_2, x_5\}$ and $\{x_4, x_6\}$ of the transfer matrix of a typical system (Fig. 6.3) [85] are shown in Fig.6.4a.



(a) PN REPRESENTATION OF I- INVARIANTS CORRESPONDING TO I S Ts (FIG. 6.3)



(b) MERGING-IN OF TRANSITIONS t_1 AND t_2



(c) MERGING - IN OF TRANSITIONS t_1, t_2, t_3

FIG. 6.4 _ PN REPRESENTATION OF I- INVARIANTS AND MERGING-IN OF TRANSITIONS

	A	B	C	D	
A		1 X_1	2 X_2		
B	1 X_3		3 X_4		
C	2 X_5	2 X_6		X_7	
D					4 X_8
E				4 X_9	

Fig. 6.3 TRANFER Matrix

To find feasible solutions, the transitions of different 1-IPNs are merged-in step by step. This is shown in Fig.6.4b. The transition t_1 and t_2 are merged in two possible ways. Now to the transition of each PN obtained, the transition t_3 can be merged as shown in Fig.6.4c in all possible ways to yield feasible solutions. It is noted that each set of places in a PN obtained is a feasible solution. The same solution have been obtained in [85] for ISTs under consideration.

The above concept of merging-in of transitions form the basis of the proposed technique which is given in the form of following steps

1. Find the number of places, n_{max} in the largest 1-IPN
2. Find the number of places, n_i in the i th 1-IPN.
   Find all n_i's.
   Put i = 1.
3. Find the number of places, n_{c_i} common to the largest and i th 1-IPN.

4. Generate all $(n_i - n_{c_i})$ out of $(n_{max} - n_{c_i})$ possible combinations of the uncommon places of the largest 1-IPN

5. Generate all the sequences for $(n_i - n_{c_i})$ uncommon places of i th 1-IPN.

   With places of one combination of step 4, join places of a sequence at corresponding positions. Repeat it for all sequences.

6. Repeat step 5 for all combinations of step 4.

   Include common places as separate elements of the sets obtained.

7. Repeat step 3 to 6 until all the places or all i's are exhausted.

8. If there are some 1-IPNs not considered, then test if there is a contradiction of this in results obtained. If so then delete those sets.

   Each of the remaining set will give a feasible solution.

9. Stop.

It should be noted that the number of maximum generations in step 4 and 5 will be $\sum_i \binom{n_{max} - n_{c_i}}{n_i - n_{c_i}}$ and $\sum_i (n_i - n_{c_i})!$ respectively. In step 7, maximum iterations are

$$\sum_{i=1} \left[\frac{(n_{max} - n_{c_i})!}{(n_{max} - n_i + 1)!} \right]$$

These are the worst case iterations and many will be abandoned in subsequent iterations. No compatability chart, graph or minimum covering is needed as in [85].

6.3.2 PN APPROACH

In the second approach the problem of data path optimization is reformulated in the domain of PN. It is found that PN offers a natural, logical and convenient tool to represent the data transfer among the modules of a digital system. As there are event taking place in the modules, they can easily be represented by transitions of a PN. The events of a module start only after receiving data from other modules. This can be represented by a place or set of places connected between modules. The presence of a data is represented by a token in an appropriate place. Thus, a direct analogy between the PN and the data transfers between various modules of a system can be established by following rule (R1):

i) Represent each module i by a transition i

ii) Connect two transitions t_i and t_j through a place if and only if there exists a data transfer between the corresponding i th and j th module. The arc between the transition t_i and the place is oriented away (towards) the transition if t_i is sending (receiving) data. Label the places.

iii) A simulteneous data transfer is identified by a concurrency identifier $k = 1,2,\ldots$ etc. as superscript placed on the label of places. All transfer with same k will be performed concurrently. In PN representation, the places having same k cannot have tokens at the same time. A data transfer occurring at the same time is represented without any identifier.

As an illustration, Fig.6.5 is a PN representation of data transfer of a digital system adopted from [85] and shown in Fig.6.3.

The transfer of data from a module takes place by firing the corresponding transition. A transition is enabled if there exists atleast one input place with a token. A transition without input place is also assumed fireable. This corresponds to a transmitting module. The transition on firing removes the tokens from input places and puts one token to each of the output places. It must be noted that each place contains exactly one input and one output arc. Each arc represents one interface. The problem of optimal interconnection of modules is, thus, the merging of places in the corresponding PN such as to minimize total number of arcs connected while abiding by the constraint of concurrent transfers.

The strategy adapted here is to merge-in places according to following rules:

i) As the data sent on the firing of a transition is same, all the output places can be merged into one place irrespective of whether or not the places are representing current data transfers. This will reduce the number of arcs by $(m_{ot} - 1)$ where m_{ot} is the number of output places connected to transition t. This is shown in Fig.6.6a.

ii) Let there be m concurrent places input to a transition t as shown in Fig.6.6b. If concurrency identfiers $k_1, k_2, \ldots, k_m$ are same, then places cannot be merged because the transition has to fire distinctly m times.

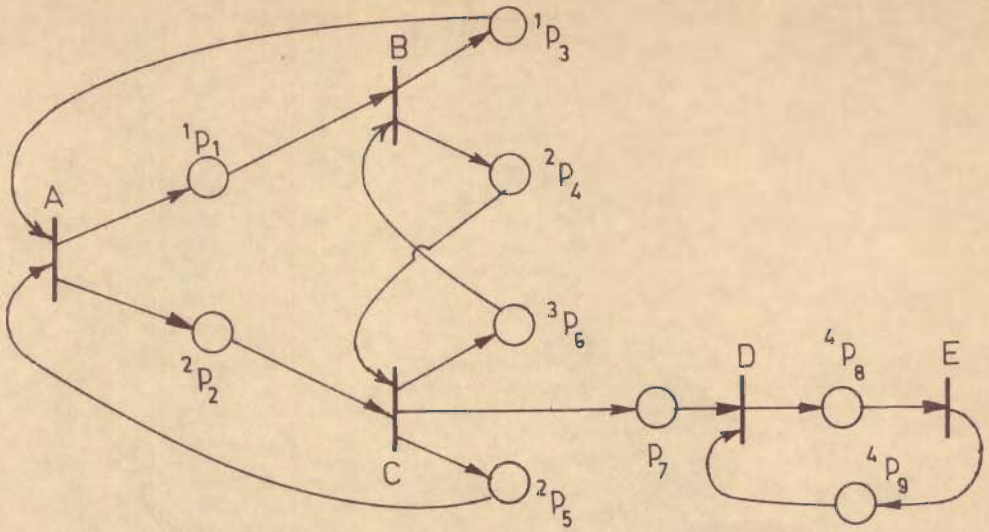
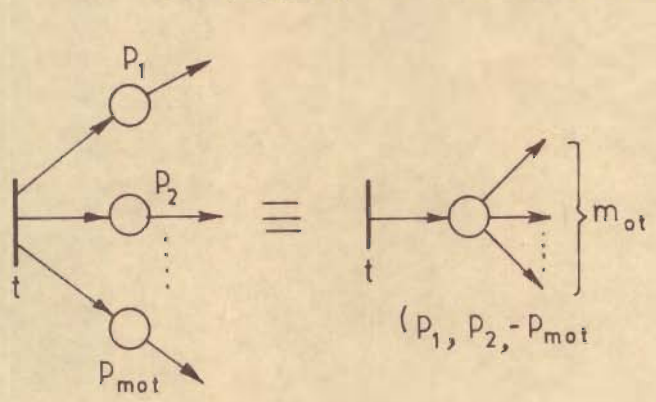
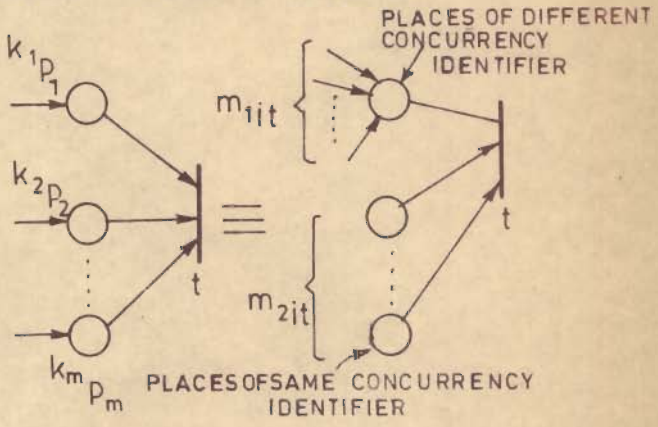


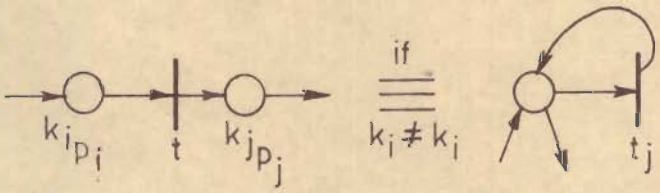
FIG. 6.5_ A PN REPRESENTATION OF DATA TRANSFER



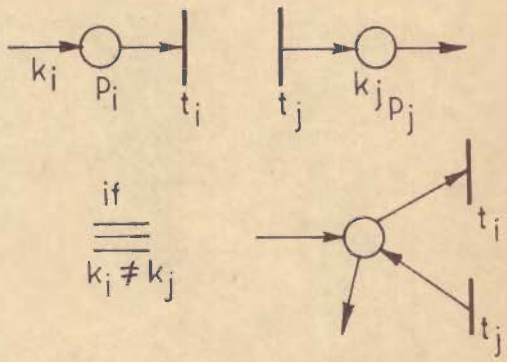
(a) MERGING-IN OF OUTPUT PLACES OF A TRANSITION



(b) MERGING-IN OF INPUT PLACES OF A TRANSITION



(c) MERGING-IN OF INPUT AND OUTPUT PLACES OF A TRANSITION



(d) MERGING-IN OF UNCONNECTED PLACES

FIG. 6.6._ RULES FOR MERGING-IN PLACES IN PN FOR DATA PATH OPTIMIZATION

However, if some identifiers, say m_{lit} are different they will have token at the same time and can be combined as shown in 6.6b figure. The number of arcs and hence, the interfaces are reduced by $(m_{lit}-1)$.

iii) Input and output places of a transitions can be merged if their concurrency identifiers are different. This will, however, not reduce the number of interfaces but will require lesser number of buses. This is shown in Fig.6.6c.

iv) Unconnected places can be combined into one provided their concurrency identifiers are different (Fig.6.6d).

The above rules in conjunction with each other can solve the problem of data path optimization. Before giving the technique it is, however, worthwhile to introduce some more concepts and recall a few which have been discussed in Chapters II and III.

Definition 6.1 : The set of all places representing concurrent transfers and which are output of different transitions are called Nonmergable Simulteneous Set of Places (NSP).

For the PN of Fig.6.5, NSPs are $\{p_1,p_3\}$, $\{p_2,p_4\}$, and $\{p_3,p_6\}$.

It has been defined in Chapter II (Definition 2.2) that ${}^{o}t$ (t^{o}) are set of input (output) places of transition t.

Definition 6.2 : (a) The largest subset of ${}^{o}t(t^{o})$ with concurrent places as input (output) is called concurrent compatible set pf places ${}^{o}t_c(t^{o}_c)$.

(b) The subset of ${}^{o}t(t^{o})$ places without concurrency is called as input (output) nonconcurrent compatible set ${}^{o}t_{Nc}$ (t^{o}_{Nc}).

For example, in Fig.6.5

$${}^{o}A = \{p_3, p_5\} = {}^{o}A_c \text{ and } {}^{o}A_{Nc} = \emptyset$$

Based upon the above considerations the proposed technique to solve data path optimization problem is given in the form of following steps:

1. Find the PN representation of the digital system and data transfers among them from R1.
2. $\forall\, t \in T$, Find ${}^{o}t$, t^{o}, ${}^{o}t_c$, t_c^{o}, ${}^{o}t_{Nc}$, t_{Nc}^{o} from definitions 2.2 and 6.2.
3. Find NSPs from definition 6.1
   Find N_{Bmin} = number of elements in the largest NSP.
4. If as many as N_{Bmin} sets of ${}^{o}t_c$ and t_c^{o} cover all the concurrent places, identify them and go to step 13.
5. (a) Put a label t_i, $i = 1,2,\ldots|T|$ to each transition.
   (b) Put i = 1
   (Starting with merged output places of each transition, here it is found, how many input places of connected transitions are merged)
   Select one t_{ic}^{o} from step 2. For each $p \in t_{ic}^{o}$ find p^{o} (p^{o} will contain only one transition).
   (c) Select a p^{o}, say it is t_j. Find ${}^{o}t_{jc}$ from Step 2.
   (d) Delete those places of ${}^{o}t_{jc}$ which are in any of NSPs with the element of t_{ic}^{o}. Call the new set ${}^{o}t_{jc}$ as ${}^{o}t'_{jc}$. Find $t_{ic}^{o} = t_{ic}^{o} \cup {}^{o}t'_{jc}$.
   (e) Repeat (c), (d) for all j and find all t_{ic1}^{o}, t_{ic2}^{o}, ... ,etc.

6. Put i = i+1, |T| where |T| is number of transitions, and repeat step 5(b), (c), (d) and (e).

7. Find minimum number, say N_B mutually exclusive sets of 6 to cover all concurrent places. If N_B does not exist go to step 10.

8. Is $N_B = N_{Bmin}$

   If yes go to step 13

9. Combine N_B sets obtained in step 7 to N_{Bmin} sets such that no two elements of any of NSP are in combined set. If this combination is possible, go to step 13.

10. (For each of the merged places obtained in the proceeding steps, here it is found how many output places of connected transitions can be merged)

    For each $p \varepsilon t^o_{ic}$ obtained from step 7, there is a unique transition t_j such that $p^o = \{t_j\}$. Find t^o_{jc} and delete those elements which are in any of NSPs with those of t^o_{ic}. The set with remaining elements is called $t^{o'}_{jc}$. Find

    $$t^o_{ic} = t^o_{ic} \bigcup t^{o'}_{jc} .$$

    Repeat it for all p's and all t^o_{ic}.

11. Repeat step 7, 8 and 9 for all i's.

12. If there is no place without concurrency identifier, go to step 17.

13. (This is for inclusion of nonconcurrent places)

    (a) Put i = 1

    (b) Select a t^o_{iNc} from step 2. For each $p \varepsilon t^o_{iNc}$ find p^o.

    (c) Find t_j such that $p^o = \{t_j\}$. Find t_{jNc} and t^o_{jNc}.

(d) Find

$$t^{o}_{iNc} = t^{o}_{iNc} \bigcup {}^{o}t_{jNc} \bigcup t^{o}_{jNc}$$

(e) Repeat (b), (c) and (d) for all j

14. Put i = i + 1, |T| and repeat (b), (c), (d), (e) of Step 13.

15. Select a t^{o}_{iNc}. For each element p ε t^{o}_{iNc}, find t^{o}'s and ${}^{o}t$'s which contain p. Find union of t^{o}'s and that of ${}^{o}t$'s such that in a set no two elements belongs to NSPs. Out of these two find which will reduce number of connections. Join t^{o}_{iNc} to the set obtained in step 11 to satisfy above condition.

Repeat it for all i.

16. Find the minimal interfaces by counting the number of arcs in t^{o}'s and ${}^{o}t$'s obtained from step 15.

17. Stop.

The technique is illustrated with the help of example of Fig.6.5.

Step 1,2: $A^{o}_{c} = \{p_1, p_2\}$; ${}^{o}A_{c} = \{p_3, p_6\}$; $A^{o}_{Nc} = {}^{o}A_{Nc} = \emptyset$

$B^{o}_{c} = \{p_3, p_4\}$; ${}^{o}B_{c} = \{p_1, p_5\}$; $B^{o}_{Nc} = {}^{o}B_{Nc} = \emptyset$

$C^{o}_{c} = \{p_5, p_6\}$; ${}^{o}C_{c} = \{p_2, p_4\}$; $C^{o}_{Nc} = \{p_7\}$; ${}^{o}C_{Nc} = \emptyset$

$D^{o}_{c} = \{p_8\}$; ${}^{o}D_{c} = \{p_9\}$; $D^{o}_{Nc} = \emptyset$; ${}^{o}D_{Nc} = \{p_7\}$

$E^{o}_{c} = \{p_9\}$; ${}^{o}E_{c} = \{p_8\}$; $E^{o}_{Nc} = \emptyset$; ${}^{o}E_{Nc} = \emptyset$

Step 3 : NSPs are $\{p_1, p_3\}$, $\{p_2, p_5\}$, $\{p_4, p_6\}$

$N_{Bmin} = 2$.

Step 4 : No two sets of t_c^o and ${}^o t_c$ cover all the concurrent places p_1 through p_6. Hence we go to step 5.

Step 5 : Put $t_1 = A$, $t_2 = B$. $t_3 = C$, $t_4 = D$, $t_5 = E$

b) $i = 1$, $t_{1c}^o = A_c^o = \{p_1, p_2\}$. We find that $p_1^o = B$ i.e. $\{t_2\}$ and $p_2^o = \{t_3\}$

c) Select $p_1^o = \{t_2\}$; ${}^o t_{2c} = {}^o B_c = \{p_1, p_6\}$

d) p_6 is not in NSPs with either p_1 or p_2.

Hence $t_{1c1}^o = \{p_1, p_2, p_6\}$

e) repeating it for $p_2^o = \{t_3\}$, we find ${}^o t_{3c} = \{p_2, p_4\}$

Hence, $t_{1c2}^o = {}^o t_{3c2} = \{p_1, p_2, p_4\}$

Step 6 : Repeating it for $i = 2, 5$ we find

$t_{2c1}^o = \{p_3, p_4, p_5\} = {}^o t_{1c1}$

$t_{2c2}^o = \{p_2, p_3, p_4\} = {}^o t_{3c2}$

$t_{3c1}^o = \{p_3, p_5, p_6\} = {}^o t_{1c1}$

$t_{3c2}^o = \{p_1, p_5, p_6\} = {}^o t_{2c2}$

$t_{5c}^o = \{p_9\} = {}^o t_{4c}$

$t_{4c}^o = \{p_8\} = {}^o t_{5c}$

Step 7 : The mutually exlusive sets to cover all the concurrent places, are obtained as

1) $\{p_1, p_2, p_4\}$, $\{p_3, p_5, p_6\}$, $\{p_8\}$, $\{p_9\}$

2) $\{p_1, p_2, p_6\}$, $\{p_3, p_4, p_5\}$, $\{p_8\}$, $\{p_9\}$

3) $\{p_2, p_3, p_4\}$, $\{p_1, p_5, p_6\}$, $\{p_8\}$, $\{p_9\}$

Step 8 : $N_B = 4$, Hence

$N_B \neq N_{Bmin}$

Step 9 : Those obtained from step 7 can be combined in 2 sets, as the concurrency constraint is not violated. Hence, the sets are

1) $\{p_1, p_2, p_4, p_8\}$; $\{p_3, p_5, p_6, p_9\}$

or $\{p_1, p_2, p_4, p_9\}$; $\{p_3, p_5, p_6, p_8\}$

2) $\{p_1, p_2, p_6, p_8\}$; $\{p_3, p_4, p_5, p_9\}$

or $\{p_1, p_2, p_6, p_9\}$; $\{p_3, p_4, p_5, p_8\}$

3) $\{p_2, p_3, p_4, p_8\}$; $\{p_1, p_5, p_6, p_9\}$

or $\{p_2, p_3, p_4, p_9\}$; $\{p_1, p_5, p_6, p_8\}$

Now we go to step 13.

Step 13.14.15 : We have only $C^o_{Nc} = \{p_7\} = {}^oD_{Nc}$; Since

$C^o = \{p_5, p_6, p_7\}$ and ${}^oD = \{p_7, p_9\}$

We can combine these as $\{p_5, p_6, p_7 p_9\}$

Minimum connections are needed if p_7 and p_9 appear together and in $p_5 p_6 p_7 p_9$ also they appear together.

Hence in step 9, the minimal solutions are

1) $\{p_1 p_2 p_4 p_8\}$; $\{p_3 p_5 p_6 p_7 p_9\}$

2) $\{p_1 p_2 p_6 p_8\}$; $\{p_3 p_4 p_5 p_9 p_7\}$ or

$\{p_1 p_2 p_6 p_9 p_7\}$; $\{p_3 p_4 p_5 p_8\}$

3) $\{p_2 p_3 p_4 p_8\}$; $\{p_1 p_5 p_6 p_9\}$

Step 16 : Consider only (1) of step 15

No. of connections for $A^{o} = 1$, ${}^{o}A = 1$

$B^{o} = 2$, ${}^{o}B = 2$

$C^{o} = 1$, ${}^{o}C = 1$

$D^{o} = 1$, ${}^{o}D = 1$

$E^{o} = 1$, ${}^{o}E = 1$

Hence total number of connections to places

$= 6 + 6 = 12$

Hence number of interfaces needed is 12 which is same as in [85].

Similarly for sets (2) and (3) obtained in step 15 interfaces required are 12.

6.4 CONCLUSION

Problems in the area of microprogrammed computer design such as (i) bit optimization in control memory and (ii) data path optimization have been tackled in this chapter by the application of Petri nets. To solve the former, all maximal compatible classes of microcommands are obtained by state equation of Petri net. It has been shown that the method is simple and can be easily used on computers, unlike existing techniques, no graph or chart is needed. Only vector additions are required. From these MCCs, the minimal bit solutions are obtained in terms of control fields by placing the microcommands in blocks with certain conditions. The technique requires lesser computations compared to other methods if employed to a certain class of problem. The class has also been identified.

The bits have further been reduced by applying bit steering through extended PN concepts. The approach is novel and can take care of more than two control fields which was hitherto too difficult. The complexity analysis of the only existing technique has been given in Chapter III (comments of Section 3.3.1.7). On comparison, the proposed technique appears computationally better because only vector additions are needed. Further the order of complexity has shown to be lower in proposed technique.

As far as data path optimization is concerned two approaches using Petri net have been employed. In first only feasible solutions are obtained. The complexity analysis shows that in worst case, it is quite large but then the number of feasible solutions are also large in that case. Further no chart or graph is needed. The second approach is one of very few attempts in the optimization consideration of PNs. Here, the data transfers have been represented by a PN and concepts of optimization have been devised in terms of PN. Complexity analysis of this method, however, has not been given because the approach is heavily data dependent. But the technique appears to be simple and involves (hopefully) lesser amount of computations. It is due to the fact that the proposed method avoids from the very beginning, because of the inherent property, the generation of redundant solutions which are to be discarded after some amount of computation in the existing techniques. For example, a solution $\{p_1, p_4, p_5\}, \{p_2, p_3, p_6\}$ in [85] does not yield minimal solution and is discarded after computing cost associated with it and comparing it with other costs. Such redundant solutions are not at all generated in the proposed technique.

CHAPTER VII

SUMMARY AND CONCLUSIONS

7.1 INTRODUCTION

The application of Petri nets to design and analysis of computer systems is a significant development. The primary reason for this is that PN can easily show parallelism involved in the system. Furthermore, it can represent systems at different levels ranging from a more abstract level to that of more detailed modeling. Another significant property is that it provides topological as well as the dynamic behavior. The advent of fast computers has provided a further impetus to research in PN. Now it can make use of high speed computers as computational tool for modeling and analysing larger and more complex systems. More efficient design and analysis techniques are however needed. It is with this motivation that the present research has been carried out. Original contributions made in this thesis have been identified. Suggestions for further work have also been incorporated.

7.2 SUMMARY OF THE RESULTS

A critical review presenting the various phases of the Petri nets and their uses has been given first. Concepts and various significant results scattered over various reports, dissertations, conferences and journals have been collected with

a view to acquaint the reader with the importance of this powerful tool. The role of Petri nets and their applications to modern computer systems has been examined. Their limitations and the techniques to overcome them are also discussed.

Of significant importance in the design of microprogrammed computers are microprogram optimizations. These have been discussed next. The reasons for various optimizations and the justification for the study of only the control memory bit optimization and data path optimization have been given. The techniques for these two have been critically reviewed. This review removes any ambiguity that might arise in the mind of a reader and also acts as a prelude before an attempt is made to apply Petri nets to such optimizations. A comparison among various control memory bit optimization techniques has been made. The comments and complexity analysis of the only available work [86] on bit steering have been given in Section 3.3.1.7 to facilitate a comparison of the technique proposed in Chapter VI. The second endeavour, in order to compare the technique proposed for data path optimization (Chapter VI), resulted in the complexity analysis (Section 3.4.3) for switching theoretic approach described in [85].

The analysis of Petri nets is required to study the problems involved in the design and development of systems. However, it suffers from some drawbacks and an attempt has been made to overcome them. Following the viewpoint that many properties of a PN can be decided from its topology [137], a decomposition

technique for a large PN into smaller nets have been proposed in Section 4.2. This feature becomes of particular importance because the properties of a large PN are derived using the proposed propositions of interconnections of such nets. This removes the problem of unmanageability of a large PN. A state equation approach has been adapted because of its inherent possession of topological information in the corresponding incidence matrix. The decomposition technique proposed is a relatively improved procedure in the sense that it gives 'apriori' the number of elementary nets to which a PN with certain properties can be decomposed. Further, it does not require as in [137], the determination of integer-valued solution of linear equations -a computationally time consuming procedure, and subsequent iterations. Only union and intersection of the sets are needed.

Given a PN with two markings, it has always been a problem to decide if one marking is reachable from another. A necessary condition for reachability [97] is available but in the absence of a sufficient condition it is difficult to solve this problem. The primary contribution in this regard is that a novel technique to solve reachability is proposed in Section 4.3. A nonreachability condition which is computationally simpler than that given by Murata [97], has also been obtained. In the absence of the fulfilment of the above condition, the proposed technique first finds the firing count vector by solving the state equation of the PN. Minimum firing count vector is then obtained from which a reachability tree is constructed to find legal firing sequences.

A significant achievement has been made here in that the technique not only decides the reachability but also provides the information about the legal firing sequences of minimum length required to transform one marking into another. Because of this minimum length, the testing for the correctness of a PN becomes easier.

Another problem which is quite common in the theory of PN is the lack of proper analysis technique when an extension to overcome the limitation of modeling power of PN is incorporated. This has been solved by proposing a transition to execute a NOT operation and then finding a state equation representation for this. This transition alongwith the conventional PN can represent the system involving all the logic operations. A generalized state equation for such representation has been obtained. This makes possible the analysis of computer systems that was hitherto not possible. As an illustration, a part of the control of CDC 6400 [105] has been analysed by the generalized state equation. It has been further shown that Murata's state equation [97] is a special class of the proposed generalized equation.

The state equations appear to be a very powerful tool in the hands of computer designer. It has been shown in Section 5.1 that many problems, namely, enumeration of simple paths between two nodes of a graph, terminal reliability of a computer network, program complexity evaluation and enumeration of maximal compatible classes (MCCs) of microcommands in control memory bit optimization lie in one class or the other of the state equation. A solution technique for enumerating simple paths between two

nodes of a graph by solving the state equation has been proposed. It is novel, simple and easily implemented on computer. Unlike other techniques, it does not require matrix multiplication. Only vector additions are needed. A complexity analysis has also been presented. In Section 5.3, terminal reliability has been obtained first by finding simple paths through the above mentioned technique and then giving PN interpretation of its disjoint terms. The simplicity of the technique and determination of near optimal solution have been justified (Section 5.5). As far as program complexity evaluation is concerned, two complexity metrics namely, simple paths and directed circuits have been defined. These (Section 5.4) in conjunction with each other can find all other complexity metrics defined in[134].. Furthermore, the determination of these metrics is simple because the technique for determining simple paths can be judiciously used for determining directed circuits as well.

In the design of control memory of a microprogrammed computer, the optimal bit solutions have been obtained in terms of control fields. First MCCs are enumerated through the state equation of PN and then the microcommands of subset of MCCs are allocated to different blocks so as to have desired properties. (Section 6.2.2). The method to enumerate MCCs is simple. The number of iterations required has been calculated and is shown to be highly data dependent. The proposed method does not require any construction of compatibility chart and graph. This makes it easily adaptable on computer and saves additional computational efforts. The technique to allocate microcommands of MCCs into blocks

according to defined properties is simple. A theorem has been postulated to test and to shift the microcommands from one block to another if the allocation does not yield a minimal optimal solution. It is this shift which makes the proposed method laborious as is in the existing techniques. However, it has been established in this section that the number of shifts will be very few, if the number of microcommands allocated in each block is not much different from the length of MCCs from which the microcommands have been allocated. Thus, the method is especially suited for a class of ROM specification. An effort has, then, been made in this section to test the existence of a possibility of further reducing the number of bits in order to have a good engineering solution. This is done by detecting bit steering among control fields by representing them through extended PN. Token distributions of places corresponding to microcommands are obtained from control fields and the ROM specifications so as to fire the transition a minimum number of times. From these token distribution detection of bit steering and encoding of control fields are done. The proposed technique is better than the existing one [86] in the sense that no concurrency matrix and no grouping of entries are required. This needs only one Boolean matrix and column operations upon it. Further, this can solve with same case, the bit steering among three or more control fields which has been computationally very difficult in the earlier method. The complexity analysis of the proposed technique has also been made. This shows the superiority of the technique.

In the design of modular interconnections, two approaches have been adopted for data path optimization. Both the approaches utilise the concept of PN. In the first approach, only the feasible solutions as defined in [85] are obtained by representing each incompatible simultaneous transfer set (IST) by 1-invariant PN. With the transitions of largest 1-invariant PN, the transitions of other 1-invariant PNs are merged such that either all the places or all the invariants, whichever is earlier have been considered. Each PN obtained by such merging-in of transitions provides a set of feasible solution. The generation of feasible solutions by the proposed technique is better computationally because (i) it does not require construction of compability chart and graph, and (ii) no procedure for obtaining minimal cover is needed. The details have already been discussed in Chapter VI. The complexity analysis further establishes the superiority of the technique. In the other proposed approach, the data transfers among the modules have been represented by a PN. The data path optimization problem has, thus, been reframed as merging-in of places such that the transitions fire minimum number of times while abiding by the constraints of concurrency. A technique for this has been proposed and illustrated with the help of an example. As the technique is highly data dependent, no computational analysis for the technique has been carried out. The proposed technique, however, appears hopefully better than that of Mathialagan and Biswas in the sense that,because of its inherent property, (i) it does not require construction of compatibility chart

and graph and (ii) it does not generate at all a solution which might be abondened after comparing the cost of other solution obtained after a large number of iterations. It is the latter advantage which makes the second approach better than the first one.

7.3 SOME PROBLEMS FOR FURTHER INVESTIGATION

The PN concept has been reviewed and applied to some of the problems in the design and development of modern computer systems. However, there are still a number of problems in which further research is desirable.

i). Simple PN have been shown to represent a large variety of systems [49]. The corresponding incidence matrix possesses some unique properties which can be utilized in investigating some of the important problems like reachability, liveness, boundedness, safeness etc. A state equation approach seems to be effective. Further, investigation properties will lead to easy analysis of large simple nets.

ii). Both the analysis techniques, namely the reachability tree and the state equation approach lack the information of the sequence in which transitions should be fired to change one marking of PN into another marking (Section 2.6). An attempt has been made in this thesis to solve this problem by first solving the state equation and from that finding the minimal firing sequences (Section 4.3). It is worthwhile investigating if the change in marking

can be expressed as a linear function of firing sequence rather than as a function of number of times the transtions fire. Further, it will be interesting to find how the other firing sequences which produce the same change in marking, are related. This will facilitate to solve reachability problem directly.

iii). Detection and isolation of faults in systems are essential features. Any system can be represented by extended PN and thus generalized state equations proposed in Section 4.4 can be applied for fault detection and isolation. Since,

$$\Delta M = M_{k+1} - M_k = A^{*T} D_k V_k \qquad (4.18)$$

if there is a fault then $\Delta M - A^{*T} D_k V_k \neq 0$. To isolate the fault then becomes an easy task. It requires only comparison of two vectors. A thorough investigation is needed in this respect.

iv). Minimal cut-sets and spanning trees play important roles in computer network. Their enumeration can be done by applying state equation technique and finding the class to which they belong (Section 5.1). However, research is needed in this direction to find effective algorithms. The properties of corresponding A_1 matrix and ΔM need be studied.

v). Two program complexity metrices and their determination have been discussed in Section 5.4. It was shown that the evaluation of such metrices is easy. Further work in this regard can be taken to use these for the development.

and testing of software. Investigation for the generation of test and program verification is required.

vi). The cost of Read/Write store is fast approaching the cost of ROM, it will replace ROM for the bulk of control memory. In this regard investigation must be done for the optimization of control memory to determine how many routines should reside permanently in control memory and how many should be brought in on a dynamic basis. A cost/performance trade off will be required.

vii). One approach using bit steering in control memory design has been discussed in Section 6.2.3 to further reduce the number of optimized bits. Another approach could be to determine directly from ROM description which of the microcommands will not be used in the bit steering, no matter in what MCCs they belong to. This will limit the search for bit sharing. Further a study is needed to determine the lower bound on number of bits when bit steering is also used. This will afford a knowledge whether to attempt for bit optimization in order to have a good engineering solution.

viii). To use PN for performance evaluation of a system, a concept of time has to be introduced. One way is to attach time unit to each of the transitions, as is the case in Section 5.4.2. Another approach could be to introduce time as another variable on a place of the PN. Hence, the variables on a place will be number of tokens and the time at which they appear. For the safe marking,

there will be only time associated with the place after each firing of a transition. In such a case, the PN then can be represented by two dimension state equations. It is worthwhile to study which of the established properties of two dimension state space theory in network could be applied to PNs to decide many problems.

To conclude it can be said that Petri nets can offer a significant contribution in the design of modern computers especially with their increased use in parallel processing environment. In this thesis, the stress has been given to formulate various algorithms suited in the design and development of system without going into details of programmer's job of implementing them. However, the effectiveness of the algorithms developed has been shown through mathematical analysis of the complexities involved. This appears to be better approach in comparison to the exhaustive method of statistical evidence through programming. However, should it require to collect statistical data it would not be difficult because the algorithms developed can readily be implemented.

It is hoped that this research will open certain new vistas to several challenging problems in the forthcoming areas of computer science.

BIBLIOGRAPHY

1. Agerwala, T. 'An Analysis of Controlling Agents for Asynchronous Processes', Hopkins Computer Research Report No.35, Computer Science Program, John Hopkins Univ., Baltimore, Maryland, 85 pages, Aug. 1974.
2. Agerwala, T. 'Towards a Theory for the Analysis and Synthesis of Systems Exhibiting Concurrency', Ph.D. Dissertation, Dept. of Electrical Engineering, John Hopkins Univ., Baltimore, April 1975.
3. Agerwala, T. 'Timing and Priority Considerations in Concurrent Computer Systems', Proc. of the 1976 Conf. on Information Science and System, pp. 307-312, 1976.
4. Agerwala, T. 'Microprogram Optimization : A Survey', IEEE Trans. Comput., Vol.C-25, pp.962-973, Oct. 1976.
5. Agerwala, T. 'Putting Petri Nets to Work', Computer, pp.85-94, Dec. 1979.
6. Agerwala, T. and Choed-Amphai, Y.C. 'A Synthesis Rule for Concurrent Systems', Proc. of the 15th Design Automation Conf., Las Vegas, June 1978.
7. Agerwala, T. and Flynn. M. 'Comments on Capabilities, Limitations and 'Correctness' of Petri Nets', Proc. of the First Annu. Symp. on Computer Architecture; New York, ACM, pp 81-86, 1973.
8. Agerwala, T. and Flynn. M. 'On the Completeness of Representation Schemes for Concurrent Systems', Conf. on Petri Nets and Related Methods, MIT, 16 pages, July 1975.

9. Agerwala, T. and Flynn. M. 'Modeling with Extended Petri Nets', Unpublished, 18 pages, 1976.

10. Agrawala, A.K. and Rauscher, T.G. 'Microprogramming : Perspective and Status', IEEE Trans. Comput., Vol.C-23, pp 817-837, Aug. 1974.

11. Amin, A.T. and Murata, T. 'Characterization of Live and Safe Marking of a Directed Graph', 1976 Conf. on Information Sciences and Systems, John Hopkins Univ., Baltimore, March- April 1976.

12. Anderson, D., Sparacio, F. and Tomasulo, R., 'The IBM System/360 Model 91: Machine Philosophy and Instruction Handling, IBM Jl. of Research and Development, Vol.11, No.1, pp.8-24, Jan. 1967.

13. Astopas, F.F. and Plukas, K.I. 'Methods of Minimizing Computer Microprograms', Aut. Cont., Vol.5, pp.10-16, 1971.

14. Baer, J. 'A Survey of Some Theoretical Aspects of Multi-processing', Computing Surveys, Vol. 5, No.1, pp.31-80, March 1973.

15. Baer, J. 'Modeling for Parallel Computation: A Case Study', Proc. of the 1973 Sagamore Conf. on Parallel Processing, New York: IEEE, pp.13-22, Aug. 1973.

16. Baer, J. and Ellis, C. 'Model, Design and Evaluation of a Compiler for a Parallel Processing Environment', IEEE Trans. Software Engg., Vol. SE-3, No.6, pp.394-405, Nov. 1977.

17. Baer, J. and Koyama, B. 'On the Minimization of the Width of the Control Memory of Microprogrammed Processors', IEEE Trans. Comput., Vol. C-28, pp. 310-316, April 1979.

18. Baker Jr, H. 'Petri Nets and Languages: Computation Structures Group Memo 68, Project MAC, MIT, 6 pages, May 1972.

19. Baker Jr, H. 'Equivalence Problems of Petri Nets', Master's Thesis, Dep. Elect. Engg., MIT, 53,pages, May 1973.

20. Best, E. 'The SOLO Operating System Described by Petri Nets', ASM/8, Computing Laboratory, Univ. New Castle upon Tyne, England, Aug. 1976.

21. Best, E. and Schmid, H. 'Systems of Open Paths in Petri Nets', Lecture Notes in Computer Science, Vol.32, Berlin:Springer-Verlag, pp. 186-193, Sep. 1975.

22. Biswas, N.N. 'Introduction to Logic and Switching Theory', Gordon and Breach, New York, 1975.

23. Brown, D.B. 'A Computerized Algorithm for Determining the Reliability of Redundant Configurations', IEEE Trans. Rel., Vol.R-20, No.3, pp.121-124, Aug. 1971.

24. Chen, T. 'Overlap and Pipeline Processing', in H. Stone (Editor) Introduction to Computer Architecture, Chicago: Science Research Associates, pp.375-431, 1975.

25. Clark, R.K. 'MIRAGER : The Best yet Approach for Horizontal Microprogramming', Proc. ACM National Conf. Boston, Mass., pp. 554-571, Aug. 1972.

26. Commoner, F. 'Deadlocks in Petri Nets', Report CA-72 06-2311, Massachusetts Computer Associates, Wakefield, 50 pages, June 1972.

27. Commoner, F., Holt, A., Even, S. and Pnueli, A., 'Marked Directed Graphs', Jl. of Computer and System Sciences, Vol.5, No.5, pp 511-523, Oct. 1971.

28. Cooprider, L. 'Petri Nets and the Representation of Standard Synchronization', Dep. Computer Science, Carnegie-Mellon Univ., Pittsburgh, 30 pages, Jan. 1976.

29. Crespi-Reghizzi, S. and Mandrioli, D. 'Some Algebric Properties of Petri Nets', Alta Frequenza, Vol.45, No.2, pp.130-137, Feb. 1976.

30. Danielson, G.H. 'On Finding Simple Paths and Circuits in a Graph', IEEE Trans. Circuit Theory, Vol. CT-15, pp 294-295, Sept. 1968.

31. Das, S.R., Banerji, D.K. and Chattopadhya, A. 'On Control Memory Minimization in Microprogrammed Digital Computer', IEEE Trans. Comput., Vol. C-22, pp.845-848, Sept. 1973.

32. Dasgupta, S. and Tartar, J. 'On the Minimization of Control Memories', Information Processing Letters, Vol.3, No.3, pp 71-74, Jan. 1975.

33. Dasgupta, S. and Tartar, J. 'The Identification of Maximal Parallelism in Straight Line Microprogram', IEEE Trans. Comput., Vol.C-25, pp 986-992, Oct. 1976.

34. Dennis, J. (Editor), 'Record of the Project MAC Conference on Concurrent Systems and Parallel Computation', New York: ACM, 199 pages, June 1970.

35. Dennis, J. and Patil, S. 'Speed Independent Asynchronous Circuits ' Proc. Fourth Int. Conf. on System Sciences, Univ. Hawaii, Honolulu, Hawaii, pp 55-58, Jan. 1971.

36. Dewitt, D.J. 'A Control Word Model for Detecting Conflicts between Microprograms', Proc. 8th Annul. Workshop on Micro-programming, Oct. 1975.

37. Dijkstra, E. 'Solution of Problems in Concurrent Program Control', Communications of the ACM, Vol.8, No.9, 569 pages, Sept. 1965.

38. Dijkstra, E. 'Cooperating Sequential Processes', in F. Genuys (Editor), Programming Languages, New York: Academic Press, pp. 43-112, 1968.

39. Fratta, L. and Montanari, U. 'A Boolean Algebra Method for Computing Terminal Reliability of Communication Networks', IEEE Trans. Circuit Theory, Vol.CT-20, pp 203-211, May 1973.

40. Furtek, F. 'Modular Implementation of Petri Nets', Master's Thesis, Dep. Elec. Engg., MIT, Cambridge, 136 pages, Sept. 1971.

41. Furtek, F. 'A New Approach to Petri Nets', Computation Structure Group Memo 123, Project MAC, MIT, Cambridge, 26 pages, April 1975.

42. Genrich, H. 'The Petri Net Representation of Mathematical Knowledge', Internal Report 76-5, Institut für Informations System for Schung, Gesellschaft für Mathematik und Detenverarbeitung, Bonn, West Germany, 30 pages, May 1976.

43. Genrich, H. and Lautenbach, K. 'Synchronisationsgraphen', Acta Informatica, Vol.2, No.2, pp 143-161, 1973.

44. Genrich, H. and Lautenbach, K. 'Facts in Place/Transition-Nets', Lecture Notes in Computer Science, Vol.64, Berlin: Springer-Verlag, pp.213-231, Sept. 1978.

45. Glushkov,V.M. 'Automata Theory and Formal Microprogram Transformations', Kibernetica, Vol.1, No.5, pp 1-9, 1965.

46. Glushkov, V.M. 'Minimization of Microprograms and Algorithm Schemes', Kibernatica, Vol.2, No.5, pp. 1-3, 1966.

47. Grasselli, A. and Montanari, U. 'On the Minimization of Read Only Memories in Microprogrammed Digital Computers', IEEE Trans. Comput., Vol.C-19, pp 1111-1114, Nov. 1970.

48. Haberman, A.N. 'Synchronization of Communicating Processes', Communication ACM, Vol.15, No.3, pp 171-176, March 1972.

49. Hack, M. 'Analysis of Production Schemata by Petri Nets', Master's Thesis, Dep. Elect. Engg., MIT, Cambridge, 119 pages, Feb. 1972, Also Technical Report 94, Project MAC, MIT, Feb. 1972.

50. Hack, M. 'The Equivalence of Generalized (Multiple-Arc) Petri Nets and Ordinary (Single-Arc) Petri Nets', Computation Structures Group Note 9, Project MAC, MIT, April 1973.

51. Hack, M. 'Extended State-Machine Allocatable Nets (ESMA), an Extension of Free Choice Petri Net Results', Computation Structures Group Memo 78-1, Project MAC, MIT, 33 pages, May 1973.

52. Hack, M. 'A Petri Net Version of Rabin's Undecidability Proof for Vector Addition Systems', Computation Structures Group Memo 94, Project MAC, MIT, 12 pages, Dec. 1973.

53. Hack, M. 'Decision Problems for Petri Nets and Vector Addition Systems', Computation Structures Group Memo 95-1, Project MAC, MIT, 79 pages, Aug.1974. Also Technical Memo 59, Project MAC, MIT, 79 pages, March 1975.

54. Hack, M. 'The Recursive Equivalence of the Reachability Problem and Liveness Problem for Petri Nets and Vector

Addition Systems', Computation Structures Group Memo 107, Project MAC, MIT, 9 pages, Aug. 1974. Also Proc. of the 15th Annu. Symp. on Switching and Automata Theory, New York: IEEE pp 156-164, Oct. 1974.

55. Hack, M. 'Petri Net Languages', Computation Structures Group Memo 124, Project MAC, MIT, 128 pages, June 1975. Also Technical Report 159, Laboratory for Computer Science, MIT, 128 pages, March 1976.

56. Hack, M. 'Decidability Questions for Petri Nets', Ph.D. Dissertation, Dep. Elec. Engg., MIT, 194 pages, Dec. 1975. Also Tech. Report 161, Laboratory for Computer Science, MIT, 194 pages, June 1976.

57. Halatsis, C. and Gaitanis, N. 'On the Minimization of Control Store in Microprogram Computers', IEEE Trans. Comput., Vol. C-27, pp. 1189-1192, Dec. 1978.

58. Hansler, E. 'A Procedure for Calculating the Reliability of a Communication Network', IEEE Trans. Rel. Vol.R-25, pp 573-575, Dec. 1971.

59. Hebalkar, P. 'Deadlock Free Sharing of Resources in Asynchronous Systems', Ph.D. Dissertation, Dep. Elec. Engg., MIT, 185 pages, Sep. 1970. Also Tech. Report 75, Project MAC, MIT, 185 pages, Sep. 1970.

60. Holt, A. and Commoner, F. 'Events and Conditions', Record of the Project MAC Conf. on Concurrent Systems and Parallel Computation, New York: ACM, pp.1-52, June, 1970.

61. Holt, A., Saint, H., Shapiro, R. and Warshall, S. 'Final Report of the Information System Theory Project', Tech. Report

RADC-TR-68-305, Rome Air Development Centre, Griffiss Air Force Base, New York, 352 pages, Sept. 1968.

62. Huen, W. and Siewiorek, D. 'Intermodule Protocol for Register Transfer Level Modules: Representation and Analytic Tools', Proc. of the Second Annu. Symp. on Computer Architecture, New York: ACM, pp 56-62, Jan. 1975.

63. Hura, G.S., Khan, A.A., Grover, D., Singh, H. and Nanda, N.K. 'Optimization of Assembly Code Generation Using Petri Nets', Int. Journal of Electronics, Vol.49, No.5, pp. 427-431, Nov. 1980.

64. Izbicki, H. 'Report on Marked Graphs', Tech. Report 25-136, IBM Vienna Laboratories, Vienna, Austria, 37 pages, April 1973.

65. Jackson, L.W. and Dasgupta, S. 'The Identification of Parallel Microoperations', Information Processing Letters, Vol.2, pp. 180-184, April 1974.

66. Jayasri, T. and Basu, D. 'An Approach to Organized Micro-instruction which Minimizes the Width of Control Store Words', IEEE Trans. Comput., Vol.C-25, pp 514-521, May 1976.

67. Johnsonbaugh, R., Kao, M.C. and Murata, T. 'Additional Transformations of Live and Safe Marked Graphs', Proc. 17th Annu. Allerton Conf. on Commun., Control, and Computing, pp. 387-396, Oct. 1979.

68. Jump, J.R. 'Asynchronous Control Arrays', IEEE Trans. Comput., Vol.C-23, No.10, pp. 1020-1029, Oct. 1974.

69. Karp, R. and Miller, R. 'Parallel Program Schemata', Jl. of Computer and System Science, Vol.3, No.4, pp 167-195, May 1969.

70. Keller, R. 'Vector Replacement Systems: A Formalism for Modeling Asynchronous Systems', Tech. Report 117, Computer Science Laboratory, Princeton Univ., New Jersey, 57 pages, Jan. 1974.

71. Khan, A.A. and Singh, H. 'Petri Net Approach to Enumerate All Simple Paths of a Graph', Electronic Letters, Vol.16, No.8, pp 291-292, 10th April 1980.

72. Khan, A.A. Hura, G.S., Nanda, N.K. and Singh, H. 'A Petri Net Approach to Compute the Terminal Reliability of a Communication Network', Proc. Pacific Telecommunication Conf., Honolulu, Hawaii, pp A5-13 to A5-17, 12-14 Jan. 1981.

73. Khan, A.A., Hura, G.S., Singh, H. and Nanda, N.K. 'On the Determination of the Solution of a Class of Murata's State Equation of Petri Nets', Proc. IEEE, Vol.69, No.4, pp 466-467, April 1981.

74. Khan, A.A. and Singh, H. 'A Method for Enumerating Maximal Compatible Classes of Microcommands Using Petri Nets', Int. J. Electronics, Vol.50, No.3, pp 231-234, March 1981.

75. Khan, A.A., Hura, G.S., Singh, H. and Nanda, N.K., 'State Equation Representation of Logic Operations Through a Petri Net', Proc. IEEE, Vol. 69, No.4, pp 485-487, April 1981.

76. Khan, A.A. and Singh, H. 'Optimal Interconnections in the Design of Microprocessors and Digital Systems through Petri Net', Accepted for Presentation in 10th IFIP Conf. on System Modeling and Optimization, Aug. 31 to Sep. 4, 1981.

77. Kim, Y.K., Case, K. and Ghare, P.M. 'An Algorithm for Computing Complex System Reliability', IEEE Trans. Rel., Vol. R-21, No.4, pp 215-219, Nov. 1972.

78. Kleir, R.L. and Ramamoorthy, C.V. 'Optimization Strategies for Microprogramms', IEEE Trans. Comput., Vol.C-20, pp 783-794, July 1971.

79. Kodres, U.R. 'Discrete Systems and Flow Charts', IEEE Trans. Software Engg., Vol.SE-4, pp.521-525, Nov. 1978.

80. Kohavi, Z. 'Switching and Finite Automata Theory', McGraw Hill, New York, 1970.

81. Kosarju, S. 'Limitations of Dijkstra's Semaphore Primitives and Petri Nets', Operating System Review, Vol.7, No.4, pp 122-126, Oct. 1973.

82. Lautenbach, K. 'Liveness in Petri Nets', Internal Report ISF-75-02.1, Institut für Informationssystemforschung, Gasellschaft für Mathematik und Datenverarbeitung, Bonn, West Germany, 33 pages, July 1975.

83. Lautenbach, K. and Schmid, H. 'Use of Petri Nets for Proving Correctness of Concurrent Process Systems', Proc. of the 1974 IFIP Congress on Information Processing 74, pp 187-191, Aug. 1971.

84. Lien, Y. 'Termination Properties of Generalized Petri Nets', SIAM Jl. of Computing, Vol. 5, No.2, pp 251-265, June 1976.

85. Mathialagan, A. and Biswas, N.N. 'Optimal Interconnections in the Design of Microprocessor and Digital System', IEEE Trans. Comput., Vol.C-29, pp 145-149, Feb. 1980.

86. Mathialagan, A. and Biswas, N.N. 'Bit Steering in the Minimization of Control Memory in Microprogrammed Digital Computers', IEEE Trans. Comput., Vol.C-30, No.2, pp.144-147, Feb. 1981.

87. McCabe, T.J. 'A Complexity Measure', IEEE Trans. Software Eng., Vol.SE-2, pp. 308-320, Dec.1976.

88. McClure, R.M. 'Parallelism in Microprogrammed Controls', Proc. Int. Adv. Summer Inst. Microprogramming, Herman, Paris, pp 307-327, 1972.

89. Meldman, J. and Holt, A. 'Petri Nets and Legal Systems', Jurimetics Jl., Vol.12, No.2, pp 65-75, Dec. 1971.

90. Merlin, P. 'A Study of the Recoverability of Computing Systems', Ph.D. Dissertation, Dep. Inf. and Comp. Sc., Univ. California, Irvine, California, 181 pages, 1974. Also Technical Report 58, Dep. Inf. and Comp. Sc., Univ. California, 181 pages, 1974.

91. Merlin, P. 'A Methodology for the Design and Implementation of Communication Protocols', IEEE Trans. Commun., Vol.COM-24, No.6, pp 614-621, June 1976.

92. Merlin P. and Farber, D. 'Recoverability of Communication Protocols - Implications of a Theoretical Study', IEEE Trans. Commun., Vol. COM-24, No.9, pp 1036-1043, Sep. 1976.

93. Mischenko, A.T. 'The Formal Synthesis of an Automaton by a Microprogram I', Kibernetica, Vol.4, No.3, pp 24-31, 1968.

94. Mischenko, A.T. 'The Formal Synthesis of an Automaton by a Microprogram II', Kibernetica, Vol.4, No.5, pp 21-27, 1968

95. Misunas, D. 'Petri Nets and Speed Independent Design', Comm. ACM, Vol. 16, No.8, pp 474-481, Aug. 1973.

96. Montangero, C. 'An Approach to the Optimal Specification of Read Only Memories in Microprogrammed Computers', IEEE Trans. Comput. Vol. C-23, pp 375-389, April 1974.

97. Murata, T. 'State Equations, Controllability and Maximal Matchings of Petri Nets', IEEE Trans. Aut. Cont., Vol.AC-22, No.3, pp 412-416, June 1977.

98. Murata, T. 'Petri Nets, Marked Graphs, and Circuit-System Theory', IEEE Circuits and Systems Society Newsletter , Vol.11, No.3, pp 2-12, June 1977.

99. Murata, T. 'Circuit Theoretic Analysis and Synthesis of Marked Graphs', IEEE Trans. Circuits and Systems, Vol.CAS-24, No.7, pp 400-405, July 1977.

100. Murata, T. 'Relevance of Network Theory to Models of Distributed/Parallel Processing', Proc. of the 1979 Int. Colloquium on Circuits and Systems, Taiepei, Taiwan, 24-25 July 1979.

101. Murata, T. 'Synthesis of Decision-Free Concurrent Systems for Prescribed Resources and Performance', IEEE Trans. Software Engg., Vol.SE-6, No.6, pp 524-530, Nov. 1980.

102. Murata, T. and Church, R. 'Analysis of Marked Graphs and Petri Nets' by Matrix Equations', Research Report MDC 1.1.8., Dept. Inf. Engg., Univ. Illinois, Chicago, Nov. 1975.

103. Murata, T. and Koh, J.Y. 'Reduction and Expansion of Live and Safe Marked Graph', IEEE Trans. Circuits and Systems, Vol. CAS-27, No.1, pp 68-70, Jan. 1980.

104. Nelson, A.C., Batts, J.R. and Beadles, L.R. 'A Computer Program for Approximating System Reliability, IEEE Trans. Rel. Vol.R-19, pp 61-65, May 1970.

105. Noe, J. 'A Petri Net Model of the CDC 6400', Proc. ACM SIGOPS Workshop on System Performance Evaluation, New York: ACM, pp 362-378, April 1971.

106. Noe, J. 'Pro-Nets : For Modeling Processes and Processors', Technical Report 75-07-15, Dep. Comput. Sc., Univ. Washington, Seattle, July 1975.

107. Noe, J. 'Nets in Modeling and Simulation', Adv. Course on General Net Theory of Processes and Systems, Hamburg, Oct.1979, Also Lecture Notes in Computer Science, Berlin: Springer-Verlag, 1980.

108. Noe, J. and Kehl, T. 'A Petri Net Model of a Modular Micro-programmable Computer (LM^2)', Technical Report 75-09-01, Comput. Sc. Dep., Univ. Washington, Seattle, 23 pages, Sept. 1975.

109. Noe, J. and Nutt, G. 'Macro E-Nets for Representation of Parallel Systems', IEEE Trans. Comput., Vol.C-22, No.8, pp 718-727, Aug. 1973.

110. Nutt, G. 'The Formulation and Application of Evaluation Nets', Ph.D. Dissertation, Comput. Sc. Group, Univ. Washington, Seattle, 181 pages, July 1972. Also Tech. Report 72-07-02, Comput. Sc. Group, Univ. Washington, Seattle, 170 pages, July 1972.

111. Nutt, G. 'Evaluation Nets for Computer Systems Performance Analysis', Proc. of the 1972 Fall Joint Comput. Conf., Montvale, New Jersey: AFIPS Press, pp 279-286, Dec. 1972.

112. Pacas, S. 'A Design Methodology for Digital Systems Using Petri Nets', Ph.D. Dissertation, Univ. Texas at Austin, Austin, 1979.

113. Patil, S. 'Coordination of Asynchronous Events', Ph.D. Dissertation Dep. Elect. Engg., MIT, Cambridge, 234 pages, May 1970. Also Tech. Report 72, Project MAC, MIT, 234 pages, June 1970.

114. Patil, S. 'Closure Properties of Interconnections of Determinate Systems', Record of Project MAC Conf. on Concurrent System and Parallel Computation, New York : ACM, pp 107-116, June 1970.

115. Patil, S. 'Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination among Processes', Computation Structures Group Memo 57, Project MAC, MIT, 18 pages, Feb.1971.

116. Patil, S. 'Circuit Implementation of Petri Nets', Computation Structures Group Memo 73, Project MAC, MIT, 14 pages, Dec.1972.

117. Patil, S. 'Micro-Control for Parallel Asynchronous Computers', Computation Structures Group Memo 102, Project MAC, MIT, March 1975.

118. Patil, S. and Dennis, J. 'The Description and Realization of Digital Systems', COMPCON 72 : Sixth Annu. IEEE Computer Society Int. Conf. Digest of Papers, New York: IEEE, pp 223-226, Oct. 1972.

119. Peterson, J. 'Computation Sequence Sets', Jl. of Computer and System Science, Vol.13, No.1, pp. 1-24, Aug. 1976.

120. Peterson, J. 'Petri Nets', Computing Surveys, Vol.9, No.3, pp. 223-252, Sept. 1977.

121. Peterson, J. 'Petri Net Theory and Modeling of Systems', Prentice-Hall Inc., April 1981.

122. Peterson, J. and Bredt, T. 'A Comparison of Models of Parallel Compution', Inf. Processing 74, Proc. of the 1974 IFIP Congress, Amsterdam, North-Holland, pp 466-470, Aug.1974.

123. Petri, C. 'Kommunikation mit Automaten', Ph.D. Dissertation, Univ. Bonn, West Germany, 1962. Also MIT Memorandum MAC-M-212, Project MAC, MIT. Also Clifford F. Greene, Jr. (Translator) 'Communication with Automata', Supplement 1 to Tech. Report RADC-TR-65-379, Vol.1, Rome Air Development Centre, Griffis Air Force Base, New York, 89 pages, Jan. 1966.

124. Petri, C. 'Introduction to General Net Theory', Lecture Notes in Computer Science, Berlin: Springer-Verlag, 1980.

125. Ponstein, J. 'Self-Avoiding Paths and Adjacency Matrix of a Graph', SIAM, Vol.14, pp 600-609, 1966.

126. Postel, J. 'A Graph Model Analysis of Computer Communications Protocols', Ph.D. Dissertation, Comput. Sc. Dep., Univ. California, Los Angeles, 191 pages, 1974.

127. Postel, J. and Farber, D. 'Graph Modeling of Computer Communications Protocols', Proc. of the 5th Texas Conf. on Computing Systems, Univ. Texas, Austin, pp 66-77, Oct. 1976.

128. Ramamoorthy, C.V. and Ho, G.S. 'Performance Evaluation of Concurrent Asynchronous Systems by Petri Nets', Proc. of COMPSAC '79, Chicago, Nov. 1979.

129. Ramamoorthy, C.V. and Tsuchiya, M. 'A High Level Language for Horizontal Microprogramming', IEEE, Trans. Comput., Vol.C-23, No.8, pp 791-801, Aug. 1974.

130. Ramchandani, C. 'Analysis of Asynchronous Concurrent Systems by Petri Nets', Ph.D. Dissertation, Dep. Elect. Engg., MIT, Cambridge, 219 pages, July 1973. Also Tech. Report 120, Project MAC, MIT, 219 pages, Feb. 1974.

131. Reddi, S.S. 'A Parallel Computer with Centralized Control', IEEE Comput. Soc. Repository, R 76-22, Feb. 1976.

132. Roberston, E.L. 'Microcode Bit Optimization is NP-Comp IEEE Trans. Comput., Vol.C-28, pp 316-319, April 1979.

133. Rubin, F. 'Enumerating All Simple Paths in a Graph', IEEE Trans. Circuits and Systems, Vol.CAS-25, No.8, pp 641-642, 1978.

134. Schneidewind, N.I. 'Application of Program and Complexity Analysis to Software Development and Testing', IEEE Trans. Rel. Vol.R-28, pp 192-198, Aug. 1979.

135. Schwartz, S.J. 'An Algorithm for Minimizing Read Only Memories for Machine Control', Proc. 9th Annu. Symp. Switching and Automata Theory, pp. 28-33, 1968.

136. Shapiro, R. and Saint, H. 'A New Approach to Optimization of Sequencing Decisions', Annu. Review in Automatic Programming, Vol. 6, Part 5, pp 257-288, 1970.

137. Sifakis, J. 'Structural Properties of Petri Nets', Lecture Notes in Computer Science, Vol.64, Berlin : Springer-Verlag, pp 474-483, Sept. 1978.

138. Singh, H., Khan, A.A., Grover, D. and Nanda, N.K. 'On Petri Net Approach to Computer Hardware and Software', 3rd Polish-English Seminar on Real-Time Process Control, Warsaw, Poland, pp 300-311, 20-23 May, 1980.

139. Sitton, W.G. and Tartar, J. 'Deletion of Non-Essential Micro-operations', Proc. 22nd Texas Conf. on Computing Systems, pp 16.1-16.7, Nov. 1973.

140. Stabler, E.P. 'Microprogram Transformations', IEEE Trans. Comput., Vol.C-19, No.10, pp 908-916, Oct. 1970.

141. Tabendeh, M. and Ramamoorthy, C.V. 'Execution Time (and Memory) Optimization in Microprograms', Proc. 7th Annu. Workshop on Microprogramming, Palo Atlo, CA, Sept. 30- Oct.2, 1974.

142. Thieler-Mevissen, G. 'The Petri Net Calculus of Predicate Logic', Internal Report ISF-76-09. Institut für Informations-systemforschung, Gesellschaft für Mathematik und Datenverar-beitung, Bonn, West Germany, 60 pages, Dec. 1976.

143. Thorton, J. 'Design of a Computer : The Control Data 6600', Scott, Foresman and Co., Glenview, Illinois, 181 pages, 1970.

144. Tokoro, M. et al. 'An Approach to Microprogram Optimization Considering Resource Occupancy and Instruction Formats', Proc. 10th Annu. Workshop on Microprogramming, pp 92-108, 1977.

145. Torng, H.C. and Wilhelm, N.C. 'The Optimal Interconnection of Circuit Modules in Microprocessor and Digital System Design', IEEE Trans. Comput., Vol.C-26, No.5, pp 450-457, May 1977.

146. Tsuchiya, M. and Jacobson, T. 'An Algorithm for Control Memory Minimization', Proc. 8th Annu. Workshop on Microprogramming, pp 18-25, 1975.

147. Valette, R. 'Analysis of Petri Nets by Stepwise Refinement', Jl. of Computer and System Sciences, Vol.18, No.1, pp 35-46, Feb. 1979.

148. Valette, R. and Diaz, M. 'Top-Down Formal Specification and Verification of Parallel Control Systems', Digital Processes, No.4, pp 181-199, 1978.

149. Warshall, S. 'A Theorem on Boolean Matrices', Jl. ACM, Vol.19, pp. 11-12, 1962.

150. White, G.M. 'Modeling of Minicomputer I/O Devices by Petri Nets', Symposium on Mini and Micro Computers in Canada, 1976.

151. Wilkes, M.V. 'The Best Way to Design an Automatic Calculating Machines', Proc. Manchester Univ. Comput. Inaugur. Conf. pp 16-18, 1951.

152. Wilkes, M.V. and Stringer, J.B. 'Microprogramming and the Design of Control Circuits in an Electronic Digital Computer', Proc. of the Cambridge Philosophical Society, 49, Part 20, pp 230-238, April 1953.

153. Wilkes, M.V., Renwick, W. and Wheeler, D. 'The Design of Control Unit of an Electronic Digital Computer', Proc. IEE, Vol.105, Part B, pp 121-128, 1958.

154. Yau, S.S., Schawe, A.C. and Tsuchiya, M. 'On Storage Optimization of Horizontal Microprograms', Proc. 7th Annu. Workshop on Microprogramming, pp 98-106, 1974.