

INFORMATION TO USERS

This was produced from a copy of a document sent to us for microfilming. While the most advanced technological means to photograph and reproduce this document have been used, the quality is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help you understand markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure you of complete continuity.
2. When an image on the film is obliterated with a round black mark it is an indication that the film inspector noticed either blurred copy because of movement during exposure, or duplicate copy. Unless we meant to delete copyrighted materials that should not have been filmed, you will find a good image of the page in the adjacent frame.
3. When a map, drawing or chart, etc., is part of the material being photographed the photographer has followed a definite method in "sectioning" the material. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.
4. For any illustrations that cannot be reproduced satisfactorily by xerography, photographic prints can be purchased at additional cost and tipped into your xerographic copy. Requests can be made to our Dissertations Customer Services Department.
5. Some pages in any document may have indistinct print. In all cases we have filmed the best available copy.

University
Microfilms
International

300 N. ZEEB ROAD, ANN ARBOR, MI 48106
18 BEDFORD ROW, LONDON WC1R 4EJ, ENGLAND

8023703

GELLER, JACOB BEN-ZION

THE USE OF PETRI NETS IN DETECTING AND MODELING
CONCURRENCIES

City University of New York

PH.D.

1980

University
Microfilms
International

300 N. Zeeb Road, Ann Arbor, MI 48106

18 Bedford Row, London WC1R 4EJ, England

Copyright 1980

by

Geller, Jacob Ben-Zion

All Rights Reserved

THE USE OF PETRI NETS IN DETECTING AND MODELING CONCURRENCIES

by

JACOB BEN-ZION GELLER

A dissertation submitted to the Graduate
Faculty in Engineering in partial fulfillment
of the requirements for the degree of Doctor of
Philosophy, The City University of New York.

1980

© COPYRIGHT BY
JACOB BEN-ZION GELLER
1980

This manuscript has been read and accepted for the Graduate Faculty in Engineering in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

4/30/80
date

Michael Anshel
Chairman of Examining Committee

4/30/80
date

Fredrick E. Thom
Executive Officer

Examination Committee

Professor M. Anshel, Chairman
Professor F. Beckman
Professor R. DiPaola
Doctor W. Gewirtz
Professor C. Kulikowski
Professor G. Lidor
Professor G. Ross

ABSTRACT

THE USE OF PETRI NETS IN DETECTING AND MODELING CONCURRENCIES

by

Jacob Ben-Zion Geller

Advisor: Professor Michael Anshel

In this thesis we explore the use of Petri Nets in the detection and modeling of concurrencies in asynchronous concurrent systems.

We consider Data Base Systems, Protection Systems, and Resource Management Systems. We show how each of these systems can be modelled as a dynamic Petri Net and how these models can be used to address problems in the respective areas. In each of these systems concurrency is an essential feature. In fact, many of the most difficult problems arising in these areas are results of their concurrent nature.

To study these problems we extend the theory of Petri Nets. We present an algebraic Petri Net in which we are able to consider markings over the nonnegative real numbers. In addition, we suggest further extensions to more general markings such as markings over vectors.

Of fundamental importance to us in all of our work is the detection of concurrencies in such systems.

We give two characterizations of concurrency in Petri Nets and show how they are equivalent. We also present algorithms for computing all of the concurrencies in a Petri Net.

We present our work towards developing a System for the Design of Asynchronous Concurrent Systems (SYSDACS). The proposed system consists of a design language, compiler, and query facility.

Through the use of the system a user can:

- design an asynchronous concurrent system at a very high level;
- investigate his design using a very high level query language;
- produce a control module which will execute his system according to his design.

To Jane

ACKNOWLEDGEMENTS

I sincerely thank my thesis advisor, Professor Michael Anshel, for his longstanding support and encouragement throughout my academic career, particularly during the research and the preparation of this document. I also wish to thank his wife, Ziva, and their children, Iris and Daphne, for the patience they showed during my many intrusions into their family life.

I would like to thank the faculty who served on my thesis committee, as well as other faculty members, such as Professor Louis Weinberg, for the time and support they have given me. I would also like to express my appreciation for the educational opportunity provided me by the City University of New York, in particular City College.

I owe my parents David and Relly and my brother and sister, Meir and Sara, a deep felt thanks for their love and support during some difficult times. A very special note of appreciation goes to Gabriel and Tamir.

Lastly I express my greatest thanks and appreciation to my fiancée Jane whose love and support have helped me throughout. I particularly appreciate the many hours she spent helping in the preparation of this document.

TABLE OF CONTENTS

	PAGE
Abstract	iv
Acknowledgements	vii
Table of Contents	viii
List of Figures	ix
Chapter	
1. Introduction	1
2. Contributions to the Theory of Petri Nets	7
2.1 Partially Ordered Addition Systems	7
2.2 An Algebraic Setting for Petri Nets	10
2.3 Morphisms	14
2.4 Subsystems	17
2.5 Real Systems	20
2.6 Coverability Theorem	22
2.7 Concurrency	26
2.8 Exclusion Systems	33
3. Applications of Petri Nets	37
3.1 Protection Systems	37
3.2 Data Base Systems	50
3.3 The Banker's Problem	59
4. Toward a System for the Design of Asynchronous Concurrent Systems	65
4.1 Introduction	65
4.2 The Design Language	67
4.3 The Compiler	71
4.4 The Control Module	79
4.5 The Query System	86
4.6 Examples	87
5. A Survey of Petri Nets and Their Applications	91
5.1 History	91
5.2 Computability	99
5.3 Process Synchronization	118
5.4 A Pipeline Processor	131
5.5 A Communication Protocol	132
6. Conclusions and Future Research	134
Bibliography	136

LIST OF FIGURES

	PAGE
1-1 Petri Net	2
1-2 Interpreted Petri Net	3
1-3 Two-Stage Pipeline Adder	3
1-4 Finite State Machine Model	4
1-5 Two-Stage Pipeline Adder (joined)	5
2-1 An Algebraic Petri Net	13
2-2 A Petri Net Morphism	16
2-3 A Subnet	19
2-4 A Coverability Tree	25
2-5 An Extended Coverability Tree	29
2-6 A SPLIT Petri Net	31
2-7 An R-Exclusion System	34
2-8 R-Attachment	35
3-1 Protection Command Petri Net	41
3-2 A "Plugged-in" Protection Command	43
3-3 Take-Grant System	45
3-4 Take-Grant System Derivation	46
3-5 Algebraic Take-Grant System	47
3-6 Algebraic Take-Grant System Derivation	49
3-7 Sample Relational Data Base	51
3-8 Query Net 1	54
3-9 Query Net 2	55
3-10 Query Net System	56
3-11 Reduced Query Net System	58
3-12 The General Banker's Problem	61
3-13 The Restricted Banker's Problem	62
3-14 A Petri Net Model of the Banker's Problem	63
4-1 PL/1 Procedures for the Producer-Consumer Problem	81
4-2 Output of SYSDACS Control Module	84
4-3 Time Graph of SYSDACS Operation	85
4-4 SYSDACS Pipeline Processor Net	90
5-1 Switching Element	91
5-2 Representation of Conditions and Events	93
5-3 Multiple Holdings	94
5-4 RNCM	100
5-5 RNCM Weakly Computing Multiplication	101
5-6 RNCM and Equivalent Vector Addition System	105
5-7 Petri Net Weakly Computing Multiplication	106
5-8 Petri Net to Weakly Compute a Polynomial	109
5-9 Petri Net Weakly Computing Multiplication	111

	PAGE
5-10 Petri Nets With Distinguished Places	114
5-11 Overlapped Petri Nets	116
5-12 Producer-Consumer Petri Net	124
5-13 Mutual Exclusion Petri Net	126
5-14 Readers-Writers Petri Net	127
5-15 Simulation of Readers-Writers Problem	128
5-16 Pipeline Processor Petri Net	131
5-17 A Communication Protocol Petri Net	132

CHAPTER 1

INTRODUCTION

Since the early 1960's many different models of concurrent computation have been developed. One such model, the Petri Net (or equivalently Vector Addition Systems), has been considered by many authors in a variety of settings:

- Communication Networks; Petri[26] , Merlin and Farber [24]
- Computability; Hack [15,16]
- Computer Systems Design; Dennis [8], Ramchandani [27]
- Formal Languages; Ginsburg [13] , Crespi-Reghezzi and Mandrioli [6]
- Infinite Groups; Anshel [1,2]
- Parallel Program Schemata; Karp and Miller [20]
- Systems Analysis; Holt [19]
- Uniform Recurrence Equations; Karp, Miller and Winograd [21]

The popularity of Petri Nets is in large part due to their ability to describe large systems in a succinct manner.

In graphically presenting Petri Nets we represent transitions as bars, places as circles, and tokens as dots. Consider the following Petri Net with transitions t_1, t_2, t_3 , and places x_1, x_2, x_3 , where place x_3 has one token in it.

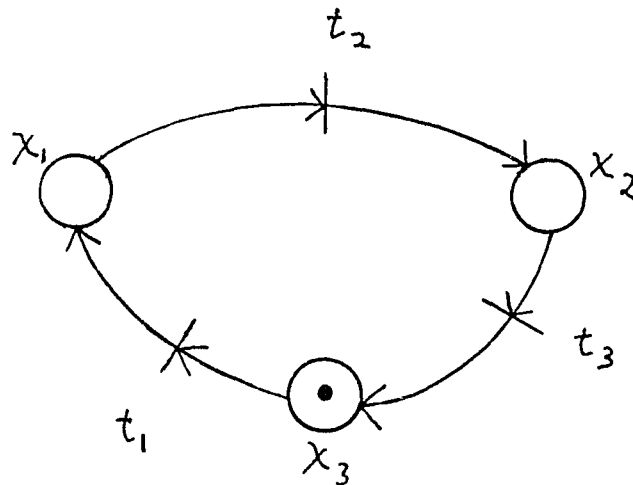


Figure 1-1

We say transition t_3 has input (output) place x_2 (x_3). Similarly, place x_1 has input (output) transition t_1 (t_2). A particular configuration of tokens in a Petri Net is called a marking.

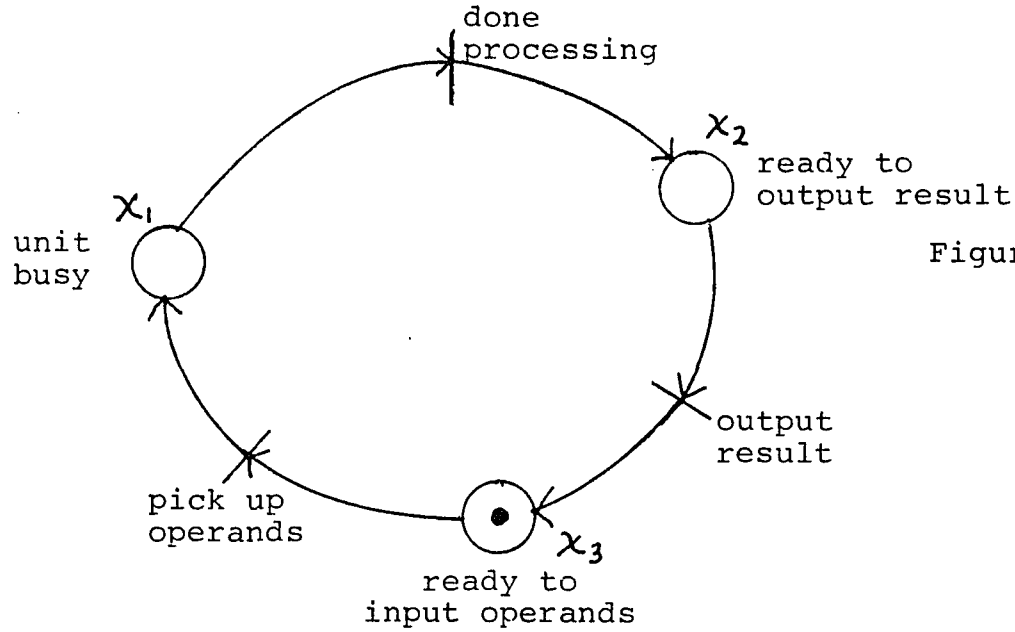
One area of interest is the flow of tokens through the net. This flow is governed by a firing rule as follows:

A transition t_i is enabled to fire (fireable) iff all input places of t_i contain at least one token.

The firing of the transition consists of removing one token from each of its input places and adding one token to each of its output places.

In the net in Figure 1-1 we see that transition t_1 is enabled. Its firing would result in removing the token from x_3 and placing it in x_1 . In this new marking only transition t_2 would be enabled.

We assign interpretations to transitions and places in modeling systems. Consider the following interpretation of the net in Figure 1-1. [27]



This can be viewed as a stage of a pipeline process. Consider the following two-stage pipeline adder.

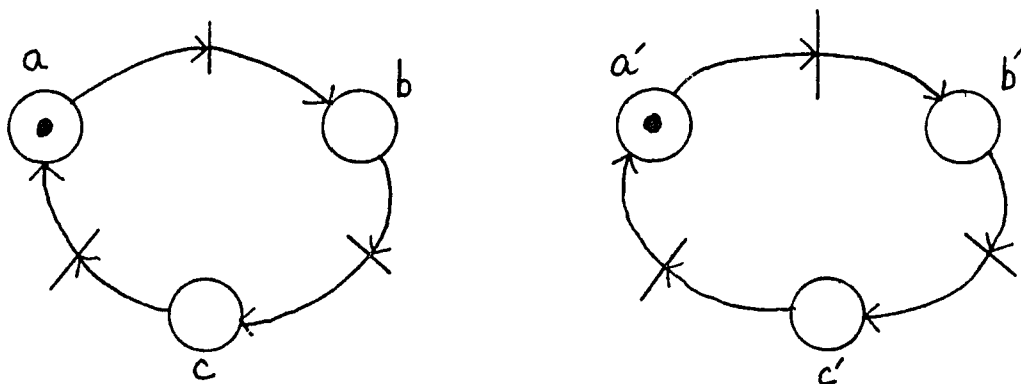


Figure 1-3

We assume that the states of the two machines are labelled a, b, c and a', b', c' . The corresponding finite state

machine having nine states and eighteen transitions is shown in Figure 1-4.

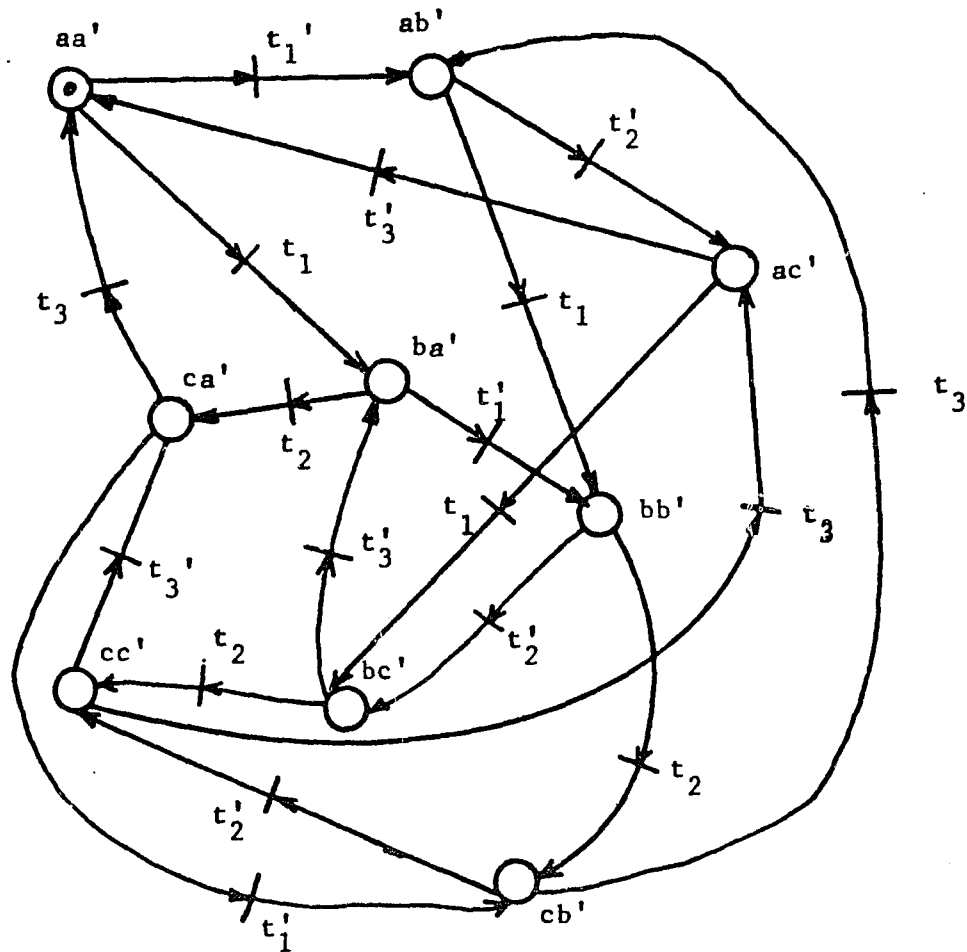


Figure 1-4

If we wanted to consider the first stage to have five functional units in parallel and the second stage eight functional units, we would have the initial marking with five tokens in place a and eight in place a'. The corresponding finite state machine of this Petri Net would need 3^{13} states.

Finally, we can more succinctly represent the two-stage adder by the following Petri Net.

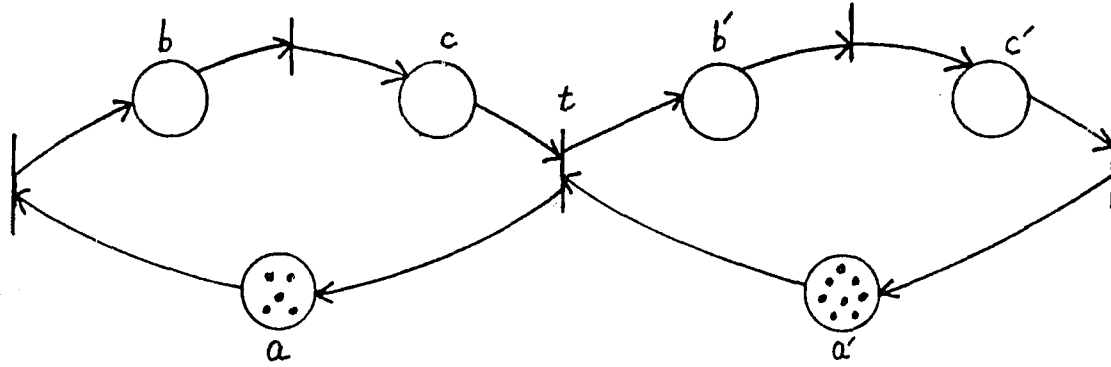


Figure 1-5

Note that transition t will only pass a token to b' if c is marked and a' is marked.

It is also of interest to ask what parallelisms or concurrencies the Petri Net gives rise to. This question will be considered in Section 2.7 of this thesis.

Despite the extensive literature on Petri Nets [25] no systematic theory has emerged. Here we consider a generalization of Petri Nets to the setting of partially ordered commutative monoids and begin a comparative study of these models and the problem of concurrency.

In Chapter 2 we present our contributions to the theory of Petri Nets. In Section 2.1 we define partially ordered addition systems which are the underlying information algebra of the Petri Net. In Section 2.2 we present an algebraic definition of Petri Nets based on marking algebras. Section 2.3 considers morphisms between Petri Net related structures. In Section 2.4 we define subsystems of the related structures. Section 2.5 presents an extension of classical Petri Nets to markings over the positive real numbers.

In Section 2.6 we consider a coverability theorem over the markings of a Petri Net. In Section 2.7 concurrency in Petri Nets is examined, and two equivalent definitions of concurrency are given. In Section 2.8 resource systems are considered. In particular we show that a class of resource systems are deadlock free.

In Chapter 3 we present applications of Petri Nets. We consider systems which are widely studied, showing how these systems can be modeled and studied via Petri Nets. In particular, in 3.1 we examine Protection Systems and issues of safety. In Section 3.2 we consider problems in Data Base Systems concerning consistent updating and deadlock prevention. In Section 3.3 we examine the Banker's Problem and show how our generalization of Petri Nets allows us to model such systems.

Chapter 4 presents our initial work on a system to aid designers of asynchronous concurrent systems.

In Chapter 5 we give a survey of applications of Petri Nets to various areas of Computer Science.

In Chapter 6 we present our conclusions and indicate future work.

CHAPTER 2

CONTRIBUTIONS TO THE THEORY OF PETRI NETS

2.1 Partially Ordered Addition Systems

The class of partially ordered addition systems is introduced below and various properties are established relating to these systems.

Recall that a commutative monoid is a triple $(S, +, 0)$ where $+$ is an associative, commutative binary operation on S and 0 , the identity or zero element satisfies the property that for all $x \in S$, $x+0=0+x=x$.

Motivated by Crapo and Rota [5] we define a subtractive system to be a triple $(S, -, 0)$ where $-$, is a partial binary operation on S such that for all $x \in S$, $x-x=0$ and if $x-0=0$ then $x=0$.

Definition 2.1 By an addition system (a.s.) we mean a quadruple $(S, +, -, 0)$ where

- (i) $(S, +, 0)$ is a commutative monoid
- (ii) $(S, -, 0)$ is a subtractive system
- (iii) If $x-y=z$ then $y+z=x$
- (iv) If $y+z=x$ then $x-y$ is defined

Examples of Addition Systems

- (i) $(\mathbb{N}, +, -, 0)$, $\mathbb{N}$ the nonnegative integers, $+$, $-$ ordinary addition and subtraction, and 0 the zero integer.
- (ii) $(\text{POW}(A), \cup, -, \emptyset)$, $\text{POW}(A)$ the power set of set A , $\cup$ denotes set union, $A_1 - A_2$ is the complement of A_2

in A_1 where $A_2 \subseteq A_1$, and undefined otherwise, and $\emptyset$ is the empty set.

- (iii) $(\sum m_i x_i, +, -, 0)$, $\sum m_i x_i$ the monomials with nonnegative integer coefficients, $+$, $-$ ordinary polynomial addition and subtraction, and 0 the zero monomial.

For S nonempty $(S, \leq)$ is a partially ordered set (poset) provided $\leq$ is a reflexive, antisymmetric, transitive binary relation on S . We say $(S, \leq)$ is compatible with an addition system $(S, +, -, 0)$ provided $x \leq y$ if and only if $y - x$ is defined.

Definition 2.2 By a partially ordered addition system

(p.o.a.s.) we mean a quintuple $(S, +, -, 0, \leq)$ where

- (i) $(S, +, -, 0)$ is an addition system
- (ii) $(S, \leq)$ is a poset compatible with $(S, +, -, 0)$.

A partially ordered commutative monoid is a quadruple $(S, +, 0, \leq)$ where $(S, +, 0)$ is a commutative monoid, $(S, \leq)$ is a poset and $x \leq y$ implies $x + z \leq y + z$ for all $z \in S$. The positive cone of $(S, +, 0, \leq)$ is the set of all $x \in S$, $0 \leq x$ (cf. Fuchs [12] for a general discussion of partially ordered algebraic systems). In the spirit of Eilenberg [10] we say that $(S, +, 0)$ is algebraically positive provided $x + y = 0$ implies $x = 0$ and $y = 0$.

Lemma 2.1 Let $(S, +, 0, -, \leq)$ be a p.o.a.s. Then the following properties are seen to hold:

- (i) For $x, y \in S$, $x \leq y$ if and only if there is a z such that $x + z = y$.
- (ii) $(S, +, 0, \leq)$ is a partially ordered commutative monoid.
- (iii) S coincides with its positive cone (i.e., $0 \leq x$ for all $x \in S$).

(iv) $(S, +, 0)$ is algebraically positive.

Proof. (i) Suppose $x \leq y$. By Def. 2.1 (ii) we have $y-x$ is defined. Let $z=y-x$ and note that Def. 2.1 (iii) implies that $x+z=y$. Conversely suppose that $x+z=y$. By Def. 2.1 (iv) we have that $y-x$ is defined. Hence by Def. 2.2 (ii), $x \leq y$.

(ii) Suppose $x \leq y$. By (i) above $y=x+z$. Hence $y+w=(x+z)+w=(x+w)+z$. Again employing (i) above we obtain $(x+w) \leq (y+w)$.

(iii) Note that for all $x \in S$, $0+x=x$ together with Def. 2.1 (iv) implies $x-0$ is defined thus $0 \leq x$.

(iv) If $x+y=0$ then by Def. 2.1 (iv), $0-x$ and $0-y$ are defined. Hence $x \leq 0$ and $y \leq 0$. Combining this with (iii) above yields $x=y=0$.

Examples of p.o.a.s.'s

(i) $(N, +, -, 0, \leq)$ where $(N, +, -, 0)$ is the addition system given in the previous example and $\leq$ is ordinary order.

(ii) $(R, +, -, 0, \leq)$, R the positive real numbers, $+$, $-$ ordinary addition and subtraction, 0 the zero real number, and $\leq$ ordinary order.

(iii) $(\sum m_i x_i, +, -, 0, \leq)$ where $(\sum m_i x_i, +, -, 0)$ is the addition system given in the previous example and $\leq$ is a component-wise ordering.

Let $(S, +, -, 0, \leq)$ be a p.o.a.s. and suppose $w \notin S$. Let $S_w = S \cup \{w\}$ and define the augmentation of $(S, +, -, 0, \leq)$ by w to be the system $(S_w, +, -, 0, \leq)$ where:

(i) $(S_w, +, 0)$ is the commutative monoid obtained by extending $+$ by the rules; $w+x=x+w=w$ for all $x \in S_w$.

(ii) $(S_w, -, 0)$ is the subtractive system obtained by extending

- by the rules; for all $x \in S$, $w-x=w$ and $w-w=0$.
- (iii) $(S_w, \leq)$ is the poset arising from extending $\leq$ by the rule; $x \leq w$ for all $x \in S_w$.

Lemma 2.2 If $(S, +, -, 0, \leq)$ is a p.o.a.s. then its augmentation by w , $(S_w, +, -, 0, \leq)$ is also a p.o.a.s..

Proof. We first demonstrate that $(S_w, +, -, 0)$ is an addition system. We noted properties (i) and (ii) of Def. 2.1 above. To verify (iii) observe that if $x=w$ and $x-y=z$ then either $y \neq w$ and $z=w$ or $y=w$ and $z=0$. In either case $y+z=w=x$. To verify (iv) we need only consider the case where $y+z = x=w$ for otherwise $y, z \in S$ and since $(S, +, -, 0)$ is an addition system, $x-y$ is defined. Now $w-y$ is defined for all $y \in S_w$ so that (iv) holds and thus $(S_w, +, -, 0)$ is an addition system.

We must now show $(S_w, \leq)$ is compatible with $(S_w, +, -, 0)$. Let $x, y \in S_w$. We verify that $x \leq y$ if and only if $y-x$ is defined. We need only consider the cases where either $x=w$ or $y=w$.

Case 1: $x=w$; If $x \leq y$ then $y=w$ so that $y-x=w-w=0$. If $y-x$ is defined then $y=w$ and hence $x \leq y$.

Case 2: $x \neq w$ and $y=w$; Note $x \leq w$ and $w-x$ is defined for all $x \neq w$. Hence $x \leq y$ if and only if $y-x$ is defined. This establishes compatibility from which the lemma follows.

2.2 An Algebraic Setting for Petri Nets

In this section we formulate the fundamental object of this investigation, the Petri Net in an algebraic setting.

Definition 2.3 Let $(S, +, -, 0, \leq)$ denote a p.o.a.s. and X a nonempty countable set of objects called places. By the marking algebra $(M(X, S), +, -, 0, \leq)$ we mean:

- (i) $M(X, S)$ is the set of all mappings $M: X \rightarrow S$ such that $\{x \mid M(x) \neq 0\}$ is finite. The elements of $M(X, S)$ are called markings.
- (ii) For $M_1, M_2 \in M(X, S)$, $(M_1 + M_2)(x) = M_1(x) + M_2(x)$.
- (iii) Also $(M_1 - M_2)(x) = M_1(x) - M_2(x)$ provided $M_1(x) - M_2(x)$ is defined for all $x \in X$. Otherwise $M_1 - M_2$ is undefined.
- (iv) $0(x) = 0$ for all $x \in X$.
- (v) $M_1 \leq M_2$ provided $M_1(x) \leq M_2(x)$ for all $x \in X$.

As an immediate consequence of Def. 2.3 we obtain

Lemma 2.3 Let $(S, +, -, 0, \leq)$ be a p.o.a.s. and X a nonempty countable set of places. Then the associated marking algebra $(M(X, S), +, -, 0, \leq)$ is a p.o.a.s..

We extend the notion of marking algebra to that of a graded marking algebra (GMA) such that:

- (i) $M(X_1, S)$ is the GMA at level 1.
- (ii) $M(X_k, M(X_{k-1}, \dots, M(X_1, S) \dots))$ is the GMA at level k consisting of the mappings from X_k to the GMA at level $k-1$.

We will denote the GMA at level k by $M(X_k, X_{k-1}, \dots, X_1, S)$.

Observe that any GMA is isomorphic to some level 1 marking algebra. (See Section 2.3).

It is through graded marking algebras that we intend to study hierarchies.

Let $(M(X,S), +, -, 0, \leq)$ be a marking algebra and let $\mathcal{T}(X,S)$ denote the Cartesian Product $M(X,S) \otimes M(X,S)$. Elements $(L,R) \in \mathcal{T}(X,S)$ are called productions over $M(X,S)$ and are denoted $L \rightarrow R$.

Definition 2.4 A Petri Net P is a triple $(M(X,S), T, M_0)$ where

- (i) $(M(X,S), +, -, 0, \leq)$ is a marking algebra.
- (ii) T is a countable set of objects called transitions, associated with each $t_i \in T$ is a production $L_i \rightarrow R_i \in \mathcal{T}(X,S)$.

- (iii) $M_0 \in M(X,S)$ is called the initial marking of P .

The underlying pair $\mathcal{T}(P) = (M(X,S), T)$ in Def. 2.4 above is called a transition system. Indeed a Petri Net may be viewed as an "initialized" transition system. We will say that a Petri Net or its underlying transition system is finitely generated, finitely related or finitely presented whenever X or T or both are finite.

Let $\mathcal{T} = (M(X,S), T)$ be a transition system, $t_i \in T$, a transition denoting $L_i \rightarrow R_i \in \mathcal{T}(X,S)$ and $M \in M(X,S)$. We say t_i is enabled at M provided $L_i \leq M$. M directly derives M' by a firing of t_i if t_i is enabled at M and $M' = (M - L_i) + R_i$.

We denote this by $M \xrightarrow{t_i} M'$ and in general write $M \Rightarrow M'$ provided $M \xrightarrow{t_i} M'$ for some $t_i \in T$.

Let T^* denote the free monoid over the set T and $\epsilon \in T^*$, the empty word. The elements of T^* are called, in the context of Petri Net theory, firing sequences. We write $M \Rightarrow^* M'$ if and only if $M = M'$ or there is a sequence of markings

$M = M_0, M_1, \dots, M_k = M'$ where $M_{i-1} \xrightarrow{t_i} M_i$ for some $t_i \in T$, $i=1, \dots, k$.
 We say M derives M' by the firing of $t_1 \dots t_k$. Also we say
 M derives M by the firing of the null firing sequence ϵ , for
 all $M \in M(X, S)$. Note $\Longrightarrow^*$ is the reflexive transitive clo-
 sure of $\Longrightarrow$. We call the set of all sequences in T^* which
 are enabled at the initial marking M_0 of a Petri Net P , the
firing sequences of P and denote it by $S(P)$.

If $M \Longrightarrow^* M'$ in $\mathcal{T}$ we say M' is reachable from M . We call
 the set of markings reachable from the initial marking M_0 of
 a Petri Net P , the reachability set of P and denote it by
 $R(P)$.

Let $P = (M(X, S), T, M_0)$ denote a Petri Net where S is the
 set of nonnegative integers.

We present the following net graphically and alge-
 braically.

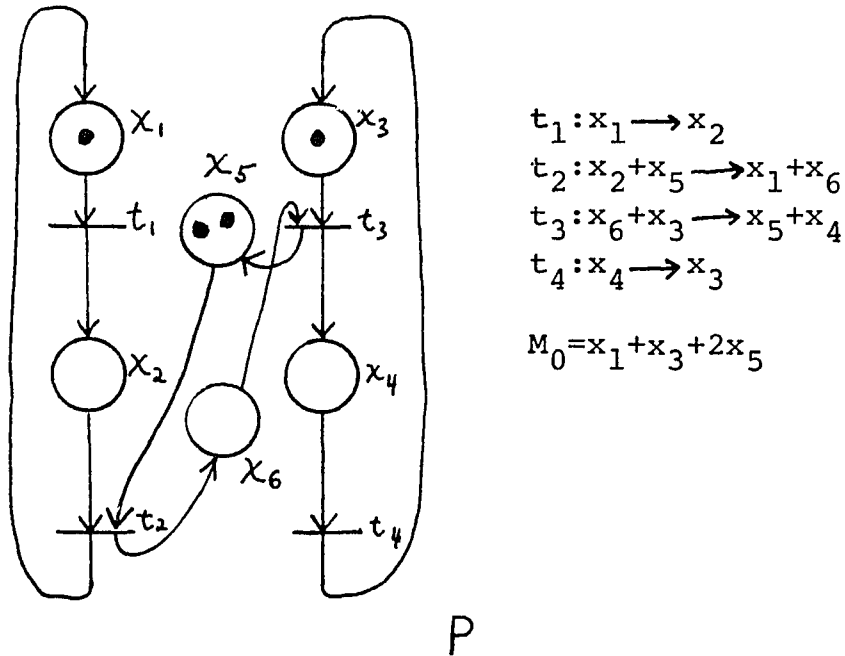


Figure 2-1

This Petri Net can be viewed as a model of a producer-consumer system. The transitions t_1 and t_2 would be the producer and the transitions t_3 and t_4 the consumer. The places x_5 and x_6 can be viewed as semaphores. They act to restrict the producer from getting more than two items ahead of the consumer. This is the result of $M_{(x_5)}=2$. Notice that in the initial marking only transition t_1 is enabled. Its firing results in the marking $x_2+x_3+2x_5$. At this marking only t_2 is enabled. Its firing results in the marking $x_1+x_5+x_6+x_3$. At this marking transitions t_1 and t_3 are enabled.

2.3 Morphisms

Here we specify the morphisms between the various sorts of objects which we have introduced.

Definition 2.5 Let $(S, +, -, 0)$ and $(S', +', -', 0')$ denote addition systems. A mapping $h: S \rightarrow S'$ is called an addition system morphism or a.s.-morphism provided

- (i) For all $x, y \in S$, $h(x+y) = h(x) +' h(y)$
- (ii) If $x-y$ is defined, then $h(x-y) = h(x) -' h(y)$.

Definition 2.6 Let $(S, +, -, 0, \leq)$ and $(S', +', -', 0', \leq')$ denote partially ordered addition systems. A mapping $h: S \rightarrow S'$ is called a partially ordered addition system morphism or p.o. a.s.-morphism provided:

- (i) h is an a.s. morphism from $(S, +, -, 0)$ to $(S', +', -', 0')$
- (ii) If $x \leq y$ then $h(x) \leq' h(y)$.

By Lemma 2.3 we may view marking algebra's $(M(X,S), +, -, 0, \leq)$ and $(M(X',S'), +', -', 0', \leq')$ as partially ordered addition systems. When speaking of p.o.a.s.-morphisms $h: M(X,S) \rightarrow M(X',S')$ it will be convenient to refer to them as marking algebra morphisms or m.a.-morphisms.

Definition 2.7 Let $\mathcal{T} = (M(X,S), T)$, $\mathcal{T}' = (M(X',S'), T')$ be transition systems and denote the derives relation in $\mathcal{T}$ and $\mathcal{T}'$ by $\Longrightarrow^*$ and $\Longrightarrow'^*$ respectively. A mapping $h: M(X,S) \rightarrow M(X',S')$ is said to be a transition system morphism or t.s.-morphism provided.

(i) h is a m.a.-morphism from $(M(X,S), +, -, 0, \leq)$ to $(M(X',S'), +', -', 0', \leq')$.

(ii) If $M_1 \Longrightarrow^* M_2$ then $h(M_1) \Longrightarrow'^* h(M_2)$.

Definition 2.8 Let $P = (M(X,S), T, M_0)$ and $P' = (M(X',S'), T', M'_0)$ denote Petri Nets. A mapping $h: M(X,S) \rightarrow M(X',S')$ is called a Petri Net morphism or p.n.-morphism provided:

(i) h is a t.s.-morphism from $\mathcal{T}(P)$ to $\mathcal{T}(P')$, $h(M_0) = M'_0$

(ii) There exist mappings $h_1: X \rightarrow X'$, $h_2: S \rightarrow S'$ such that for all $M \in M(X,S)$ and $x \in X$, $h_2(M(x)) = (h(M))(h_1(x))$,

that is the following diagram is commutative:

$$\begin{array}{ccc} X & \xrightarrow{M} & S \\ \downarrow h_1 & & \downarrow h_2 \\ X' & \xrightarrow{h(M)} & S' \end{array}$$

(iii) There exists a mapping $h_3: T \rightarrow T'$ such that for all $t \in T$
if $M_1 \xrightarrow{t} M_2$ then $h(M_1) \xrightarrow{h_3(t)} h(M_2)$.

We speak of a morphism being injective, surjective or bijective depending on whether the mapping of the carrier is respectively injective, surjective or bijective. An isomorphism is a bijective morphism whose inverse mapping also denotes a morphism.

Example of a p.n.-morphism

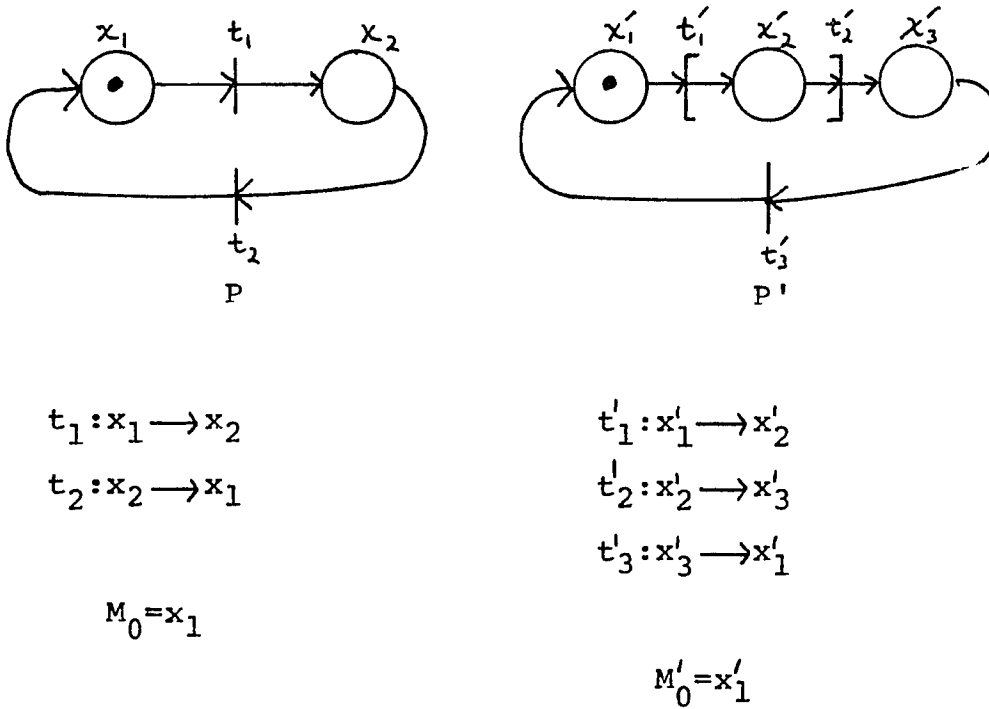


Figure 2-2

Observe that Figure 2-2 shows two Petri Nets such that P is an injective p.n.-morphism on P' .

2.4 Subsystems

Here we make precise the notion of 'subsystem' for those systems we have thus far introduced.

A submonoid of a commutative monoid $(S, +, 0)$ is a commutative monoid $(S', +', 0')$ where S' is a subset of S , $+' is the restriction of $+$ to S' and $0' = 0$. We say $(S', +', 0')$ is the submonoid induced by $(S, +, 0)$ on S' .$

By an addition subsystem of an addition system $(S, +, -, 0)$ we mean an addition system $(S', +', -', 0')$ where $(S', +', 0')$ is induced by $(S, +, 0)$ on S' and $x -' y = x - y$ whenever $x - y$ is defined. Also if $x -' y = x - y$ precisely when $x - y$ is defined we say $(S', +', -', 0')$ is induced by $(S, +, -, 0)$ on S' .

By a subposet of a poset $(S, \leq)$ we mean a poset $(S', \leq')$ where S' is a subset of S , and $x \leq' y$ implies $x \leq y$. Also if $x \leq y$ precisely when $x \leq' y$ we say $(S', \leq')$ is induced by $(S, \leq)$ on S' .

A partially ordered commutative submonoid of a partially ordered commutative monoid $(S, +, 0, \leq)$ is a partially ordered commutative monoid $(S', +', 0', \leq')$ where $(S', +', 0')$ is induced by $(S, +, 0)$ and $(S', \leq')$ is a subposet of $(S, \leq)$. Also if $(S', \leq')$ is induced by $(S, \leq)$ then we say that $(S', +', 0', \leq')$ is induced by $(S, +, 0, \leq)$ on S' .

A partially ordered addition subsystem of a partially ordered addition system $(S, +, -, 0, \leq)$ is a partially ordered addition system $(S', +', -', 0', \leq')$ where $(S', +', -', 0')$ is an addition subsystem of $(S, +, -, 0)$ and $(S', +', 0', \leq')$ is a partially ordered commutative submonoid of $(S, +, 0, \leq)$. Also

if $(S', +', -, 0')$ and $(S', +', 0', \leq')$ are induced by $(S, +, -, 0)$ and $(S, +, 0, \leq)$ respectively then we say that $(S', +', -, 0', \leq')$ is induced by $(S, +, -, 0, \leq)$ in S' .

By a subalgebra of a marking algebra $(M(X, S), +, -, 0, \leq)$ we mean a partially ordered addition subsystem of the marking algebra. Let X' be a nonempty subset of X . Let $A(X')$ consist of those markings M in $M(X, S)$ such that $M(x) \neq 0$ implies x in X' (the support of M lies in X'). $A(X')$ induces a partially ordered addition system or subalgebra, which we call the subalgebra induced by X' . The subalgebra induced by X' is p.o.a.s.-isomorphic to $(M(X', S), +, -, 0, \leq)$ under the mapping $h: A(X') \rightarrow M(X', S)$ given by: $h(M) = M'$, $M(x) = M'(x)$ for all x in X' . It will be natural to speak of the elements of $M(X', S)$ as submarkings of $M(X, S)$ and to identify the subalgebra of $M(X, S)$ induced by X' with $(M(X', S), +, -, 0, \leq)$.

By a subsystem of a transition system $(M(X, S), T)$ we mean a pair (A, T') where A is a subalgebra of $M(X, S)$ and each t_i in T' denotes a production $L_i \rightarrow R_i$ such that L_i, R_i are in A . We call such t_i , A-transitions. If all A -transitions in T are also in T' , we call (A, T) a full subsystem. Let X' be a nonempty subset of X . We say a subsystem is defined in X' provided it is of the form $(A(X'), T')$ where $A(X')$ is the subalgebra induced by X' . We call the full system $(A(X'), T')$ defined in X' , the subsystem induced by X' . We identify the subsystem induced by X' with the transition system $(M(X', S), T(X'))$ by employing $h: A(X') \rightarrow M(X', S)$ of the preceding paragraph. The markings M in $A(X')$ are iden-

tified with $h(M')$ and t_i in T , $t_i = L_i \rightarrow R_i$ is identified with t'_i in $T'(X)$ such that $t'_i = h(L_i) \rightarrow h(R_i)$. The identification permits us to discuss various manipulations with the subsystem induced by X' (e.g., derivations in the subsystem). We therefore explicitly refer to $(M(X',S), T(X'))$ as a transition subsystem of $(M(X,S), T)$ induced by X' . In general $(M(X',S), T')$, $T' \subseteq T$, is called a transition system of $(M(X,S), T)$.

Let $(M(X,S), T, M_0)$ be a Petri Net, and X' denote a non-empty subset of X , and M_0 in $A(X')$. The Petri Net induced by X' is given by $(M(X',S), T(X'), M'_0)$ where $(M(X',S), T(X'))$ is the transition subsystem induced by X' and M_0 is identified with M'_0 in $M(X',S)$ (i.e. $h(M_0) = M'_0$). A subnet of $(M(X,S), T, M_0)$ is an induced subset of some Petri Net $(M(X,S), T_1, M_0)$ where T_1 is a subset of T .

Subsystem Example

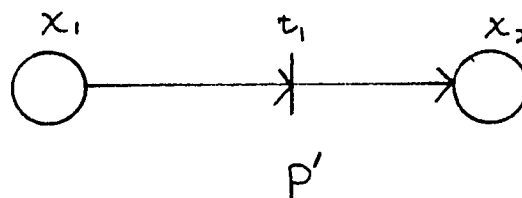


Figure 2-3

Observe that the Petri Net P' in Figure 2-3 is a subnet of the Petri Net P given in Figure 2-2.

2.5 Real Systems

In this section we characterize those systems whose underlying arithmetic operations and order relation are realizable in the real number system. We call such systems 'real systems.'

Following Fuchs [12] we say that a poset $(S, \leq)$ is a fully ordered set or foiset if and only if for all $x, y \in S$ either $x \leq y$ or $y \leq x$. We say that a p.o.a.s. $(S, +, -, 0, \leq)$ (resp. a p.o.c.m. $(S, +, 0, \leq)$) is a fully ordered addition system, f.o.a.s. (resp. a fully ordered commutative monoid, f.o.c.m.) if and only if $(S, \leq)$ is a foiset.

A commutative monoid $(S, +, 0)$ is said to be cancellative if and only if $x+z=x+z'$ implies $z=z'$.

We write $x < y$, for $x, y \in S$, $(S, \leq)$ is a poset provided $x \leq y$ and $x \neq y$. We write $y \geq x$ (resp. $y > x$) provided $x \leq y$ (resp. $x < y$).

Let $(S, +, 0)$ be a commutative monoid and let $x \in S$. We write $1 \cdot x = x, 2 \cdot x = x + x, \dots, (n+1) \cdot x = (n \cdot x) + x$.

Two distinct elements x, y of a f.o.c.m. $(S, +, 0, \leq)$ are said to be an anomalous pair if and only if $n x < (n+1) y$, $n y < (n+1) x$, for all positive integers n , or $n x > (n+1) y$, $n y > (n+1) x$ for all positive integers n .

By a real addition system r.a.s. we mean a p.o.a.s. $(S, +, -, 0, \leq)$ where $(S, +, 0)$ is a submonoid of the nonnegative reals and $-, \leq$ are ordinary subtraction and the standard ordering of the reals.

Lemma 2.4 Let $(S, +, -, 0)$ and $(S, +, -', 0)$ be addition systems with the same underlying commutative monoid $(S, +, 0)$. If $(S, +, 0)$ is cancellative then $(S, +, -, 0)$ and $(S, +, -', 0)$ coincide.

Proof. We need only show $-$, $-'$ define the same partial operation. Now $x - y = z$ and $x - y = z'$ exist precisely when $y \leq x$. When this is the case $y + z = x = y + z'$. If $(S, +, 0)$ is cancellative, $z = z'$.

Lemma 2.5 A f.o.a.s. $(S, +, -, 0, \leq)$ is isomorphic to a real addition system if and only if

- (i) $(S, +, 0)$ is cancellative
- (ii) S contains no anomalous pairs.

Proof. If $(S, +, -, 0, \leq)$ is a real addition system then (i) and (ii) are seen to hold.

Suppose $(S, +, -, 0, \leq)$ is a f.o.a.s. satisfying (i) and (ii). From Fuchs [Theorem 4, p. 167] we have that a f.o.c.m. is isomorphic to a submonoid of the addition group of reals if and only if (i) and (ii) hold. Since $x \geq 0$ for all $x \in S$, that submonoid may be seen to lie in the submonoid of non-negative reals. By (i) and Lemma 2.4 above, subtraction is seen to be uniquely determined and is in fact ordinary subtraction. Thus $(S, +, -, 0, \leq)$ is a real addition system.

A marking algebra $(M(X, S), +, -, 0, \leq)$ is said to be real provided S is a real addition system. A transition system or Petri Net whose underlying marking algebra is real is referred to as a real transition system or Petri Net.

2.6 Coverability Theorem

In the manner of Smullyan [28] we say:

Definition 2.9 An unordered tree U is a triple (V, l, R)

where:

- (i) V is a set of objects called vertices;
- (ii) $l: V \rightarrow \mathbb{Z}^+$, the level function assigns a positive integer $l(x)$ to each vertex $x \in V$ called the level of x ;
- (iii) R is a binary relation on V , xRy is read " x is a predecessor of y " or " y is a successor to x ";
- (iv) There is a unique vertex of level 1, a_1 , called the root of the tree;
- (v) Every vertex other than the root has a unique predecessor;
- (vi) For any $x, y \in V$ if y is a successor to x then
$$l(y) = l(x) + 1$$

We shall call $x \in V$ a leaf if it has no successors. By a path of U we mean a finite or denumerable sequence of vertices, beginning with the root, which is such that each term of the sequence of vertices (except the last, if there is one) is the predecessor of the next. By a maximal path or branch we shall mean a path whose last term is a leaf of the tree, or a path which is infinite.

We say that y is a descendant of x if x and y are on a path of U and y appears after x (denoted $x \ll y$).

Let $P = (M(X, S), T, M_0)$ be a Petri Net and $R(P)$ the reachability set of P .

Definition 2.10 We associate with P a coverability tree

CT(P), which we define as follows:

- (i) The root vertex is M_0 .
- (ii) Let x, y be vertices of the tree; then xRy iff there exists $t \in T$ such that $x \xrightarrow{t} y$.
- (iii) For distinct vertices x, y if $x \ll y$ and $x=y$ then y is a leaf.
- (iv) For vertex $y=M$ (denoted M^Y), we have $M^Y(r)=\omega$ iff there exists $x \ll y$ such that $M^X \leq M^Y$ and $M^X(r) < M^Y(r)$.

We now show that the coverability tree of a finitely presented Petri Net is finite. Our proof is similar to Karp & Miller [20].

Theorem 2.1 Let $P=(M(X,S),T,M_0)$ denote a finitely presented Petri Net such that S is a fuset. Then the coverability tree of P is finite.

Lemma 2.6 (König Infinity Lemma [20]) Let U be an unordered tree in which each vertex has only a finite number of successors and there is no infinite path. Then U is finite.

Lemma 2.7 In an infinite sequence of elements of $M(X,S)$ there exists an infinite non-decreasing chain.

Proof of Theorem 2.1

Since P is finitely presented and hence T is finite from lemma 2.6 the coverability tree will be infinite iff there exists an infinite path. Let vertices $v_1, v_2, \dots$

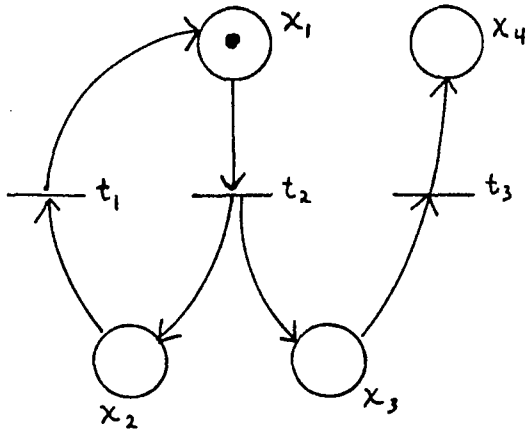
constitute such an infinite path. By lemma 2.7 there exist vertices in the path such that $v_{i_1} \leq v_{i_2} \leq \dots \leq v_{i_n} \leq \dots$.

From (iii) of definition 2.10 we know there does not exist k such that $v_{i_j} = v_{i_{j+k}}$ or the path wouldn't be infinite.

Hence $v_{i_1} < v_{i_2} < \dots$ is an increasing chain. From (iv) of definition 2.10 each $v_{i_{j+1}} > v_{i_j}$ has at least one more maximal marking. Since P is finitely presented, the path must be finite. QED.

Example of a Coverability Tree

Consider the following Petri Net.



The coverability tree follows:

(Note that we represent a marking as a vector of place markings.)

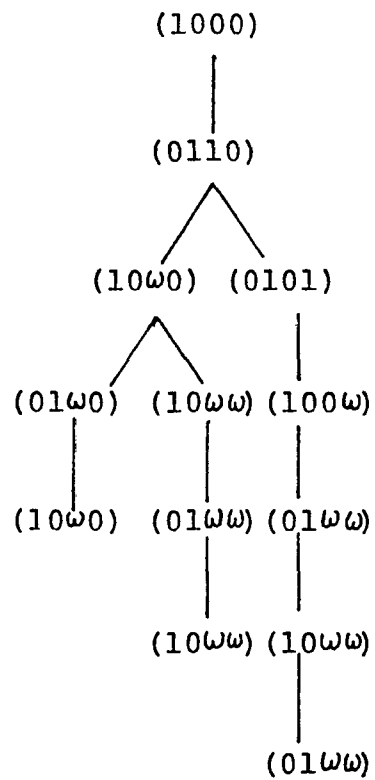


Figure 2-4

2.7 Concurrency

Here we will discuss concurrency and related notions such as persistence and permutability as they apply to Petri Nets. We will show two different definitions of concurrency to be equivalent. We first consider persistence as an important related notion.

For Petri Net $P=(M(X,S),T,M_0)$ transitions $t_1, t_2 \in T$, $t_1 \neq t_2$, are persistent iff for all $M \in R(P)$ $M \xrightarrow{t_1} M' \xrightarrow{t_2}$ implies $M \xrightarrow{t_1 t_2}$.

If for all $t_1, t_2 \in T$, t_1 and t_2 are persistent then we say that the Petri Net P is a persistent net.

Persistence is currently an open problem. Landweber and Robertson [22] have shown that the reachability sets of persistent nets are semilinear. Liveness is decidable for persistent nets [15]. Persistence is also known to be reducible to the reachability problem [25].

Persistence gives rise to the notion of permutability.

Definition 2.11 Let $PERM(s)$ denote the permutations of the word s . A firing sequence $s \in T^*$ is a permutable firing sequence at marking M iff for all $s' \in PERM(s)$, $M \xrightarrow{s'} \rightarrow$.

We will see that firing sequences which are concurrent also have the property of permutability. However, permutability does not imply concurrency, as in the case of mutual exclusion.

Definition 2.12 For Petri Net $P=(M(X,S),T,M_0)$ $t_1, t_2 \in T$, t_1 is concurrent with t_2 , denoted $t_1 // t_2$, iff there exists

$M \in R(P)$ such that $M \geq (L_1 + L_2)$. We extend this notion to a firing sequence.

Let $s \in S(P)$, $s = t_{i_1}^{n_1} t_{i_2}^{n_2} \dots t_{i_k}^{n_k}$.

The firing sequence s is a concurrent firing sequence in P denoted $s_{//}$ provided:

(i) $s \in S(P)$

(ii) there exists $M \in R(P)$ such that $M \geq \sum n_{i_j} L_{i_j}$.

We will now show that concurrency is decidable for finitely presented Petri Nets.

Definition 2.13 Let $P = (M(X, S), T, M_0)$ be a finitely presented Petri Net and $CT(P)$ its coverability tree. We define the extended coverability tree in the following manner:

(i) Let each node of $CT(P)$ be an extended vertex.

(ii) If x is an extended vertex and there exists $t_i = L_i \rightarrow R_i$ in T such that $L_i \leq M^x$ then the successor of x is the extended vertex $(M^x - L_i)$ and we label the pair $(M^x, M^x - L_i)$ with t_i . If $M^x = (M^x - L_i)$ then we label the pair t_i^* unless t_i^* already is a label on that path, in which case M^x is a leaf.

We define the commutative closure of a set S as the set S' such that for x in S , x and all of the permutations of x are in S' .

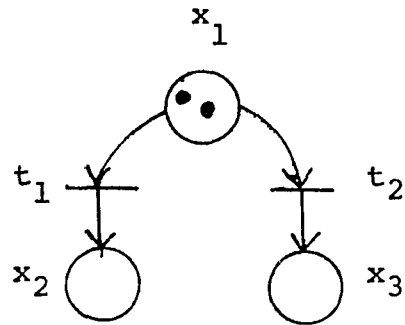
Definition 2.14 For a Petri Net P we define the concurrency set, $C(P)$, as the commutative closure of the labeled paths of the extended coverability tree.

Remark: For every Petri Net P the concurrency set $C(P)$ is finite.

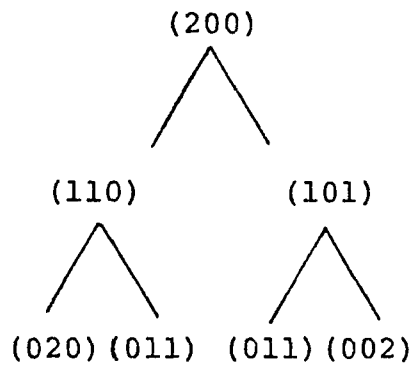
Proposition: For Petri Net P the following hold:

- (i) All concurrent firing sequences are permutable;
- (ii) For $s \in S(P)$, s is a concurrent firing sequence of P iff $s \in C(P)$.

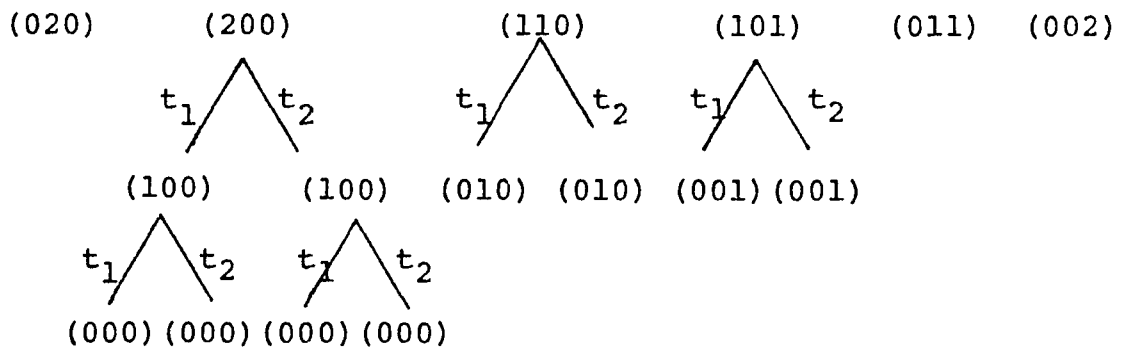
Petri Net



Coverability Tree



Extended Coverability Tree



$C(P)$ = commutative closure of $\{t_1 t_2, t_1^2, t_2^2, t_1, t_2\}$

Figure 2-5

We will now consider an alternative definition of concurrency.

Let $P=(M(X,S),T,M_0)$, $P'=(M(X',S'),T',M'_0)$ denote Petri Nets.

The mapping $SPLIT:M(X,S) \longrightarrow M(X',S')$ is an injective t.s.-morphism such that:

(i) h is an m.a.-morphism

$$h:(M(X,S),+,-,0,\leq) \longrightarrow (M(X',S'),+',-',0',\leq')$$

(ii) $M_1 \xrightarrow{*} M_2 \implies h(M_1) \xrightarrow{*'} h(M_2)$

(iii) For $t=L \rightarrow R \in T$, $M_1 \xrightarrow{t} M_2$ there exists

$t'=L' \rightarrow R'$, $t''=L'' \rightarrow R'' \in T'$ such that

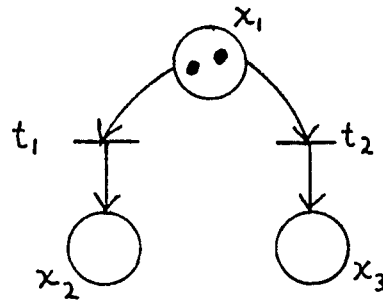
$$L'=h(L), R''=h(R), R'=L''=lx_t.$$

We call the split element of R' and L'' the transition place x_t .

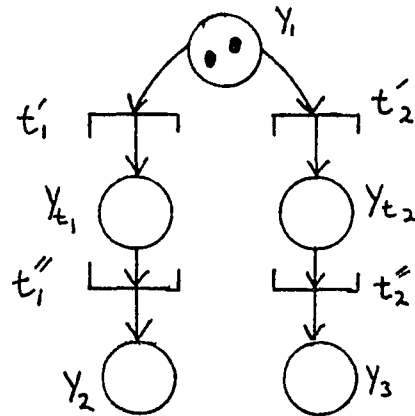
Definition 2.15 For $t_1, t_2 \in T$, $t_1 // t_2$ iff there exists $M \in R(P')$, $P' = SPLIT(P)$, such that $M(x_{t_1}) \neq 0$ and $M(x_{t_2}) \neq 0$.

Definition 2.16 We define the concurrency set $\overline{C}(P)$ as the commutative closure of the words $t_1^{M(x_{t_1})} \dots t_k^{M(x_{t_k})}$ for each M in $CT(P')$, $P' = SPLIT(P)$.

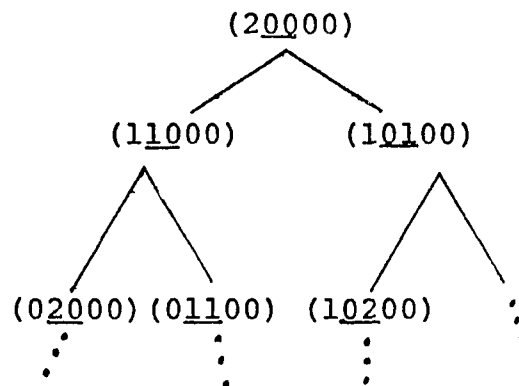
Petri Net P



$P' = \text{SPLIT}(P)$



Coverability Tree of P' ($M = (y_1 \ y_{t_1} \ y_{t_2} \ y_2 \ y_3)$)



$$\overline{C}(P) = \text{commutative closure of } \{t_1, t_2, t_1^2, t_2^2, t_1 t_2\}$$

Figure 2-6

Theorem 2.2

Let $P=(M(X,S),T,M_0)$ denote a finitely presented Petri Net. The concurrency sets $C(P)$ and $\bar{C}(P)$ are equivalent.

Observation: For $P' = \text{SPLIT}(P)$ there exists homomorphism h such that the coverability tree $CT(P) \subseteq h(CT(P'))$.

Proof of Theorem 2.2

We will show that $C(P) \subseteq \bar{C}(P)$ and $\bar{C}(P) \subseteq C(P)$. Recall that $C(P)$ is based on the labeled paths of the extended coverability tree and $\bar{C}(P)$ is based on the SPLIT of P .

(i) $x \in C(P) \Rightarrow x \in \bar{C}(P)$

If $x \in C(P)$ then x is of the form $t_1^{n_1} t_2^{n_2} \dots t_k^{n_k}$ and there exists marking M in $CT(P)$ such that $M \geq \sum n_i L_i$. By definition 2.15 we conclude that the split transition places x_{t_i} can achieve marking $n_i, i=1, \dots, k$. Hence there exists M' in $CT(P')$ such that $t_1^{M'(x_{t_1})} \dots t_k^{M'(x_{t_k})} = t_1^{n_1} t_2^{n_2} \dots t_k^{n_k}$ is in $\bar{C}(P)$.

(ii) $x \in \bar{C}(P) \Rightarrow x \in C(P)$

If $x \in \bar{C}(P)$ then x is of the form $t_1^{n_1} t_2^{n_2} \dots t_k^{n_k}$ and there exists marking M in $CT(P')$, $P' = \text{SPLIT}(P)$, such that $M(x_{t_i}) = n_i, i=1, \dots, k$. The existence of such an M in $CT(P')$ implies that there exists M' in $CT(P)$ such that $M' \geq \sum n_i L_i$. Such an M' in $CT(P)$ will give rise to a labeled path of the extended coverability tree which is equal to x up to commutativity. Thus x is in $C(P)$. QED.

2.8 Exclusion Systems

Here we sketch a model of systems which share a set of resources in a particular way. We call such systems Exclusion Systems.

We say that a Petri Net P is an R-exclusion system, denoted P^R , iff for the underlying transition system there exists transition subsystems such that:

- (i) P is covered by the subsystems.
- (ii) A subset of places is common to all subsystems. These are the resource places, X_R , shared by the subsystems.
- (iii) There is precisely one special transition in each subsystem with input places only in X_R . This transition, called the test-and-set transition, prevents the subsystem from starting until all of the resources it needs are available.
- (iv) There is precisely one special input place to the test-and-set transition with initial marking of one. This place is called a lock. It prevents the test-and-set transition from firing more than once.

Example of an R-exclusion system

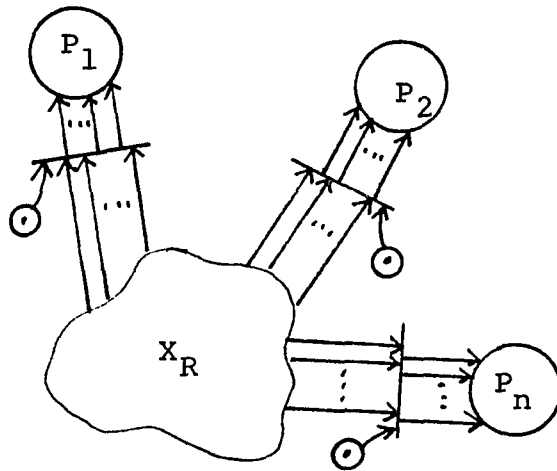


Figure 2-7

In combining R-exclusion systems we are interested in the resultant firing sequences. The next two definitions will be useful in describing these new sequences.

We say that a firing sequence of a Petri Net is an executable firing sequence iff it is a word containing each transition precisely once. We denote the set of executable firing sequences of P as $EX(P)$.

Let L_1 and L_2 be arbitrary languages; then the shuffle product [18] of L_1 and L_2 , denoted by $L_1 \sqcup L_2$, is the language

$$L_1 \sqcup L_2 = \{a_1 \dots a_k \mid a_{i_1} \dots a_{i_n} \in L_1, a_{j_1} \dots a_{j_m} \in L_2, \\ i_1 < i_2 < \dots < i_n, j_1 < j_2 < \dots < j_m, \text{ and } \{i_1, \dots, i_n\} \cup \\ \{j_1, \dots, j_m\} = \{1, 2, \dots, k\} \text{ and } \{i_1, \dots, i_n\} \cap \\ \{j_1, \dots, j_m\} = \emptyset.\}$$

Example of Shuffle Product

Let $L_1 = \{a, b\}$ and $L_2 = \{cd\}$; then $L_1 \sqcup L_2 = \{a, b, cd, acd, cad, cda, bcd, cbd, cdb\}$.

For R-exclusion systems P^R, Q^R we define the R-attachment of P^R and Q^R , denoted $P^R @ Q^R$, as a Petri Net formed by mapping resource places to resource places.

Example of R-attachment

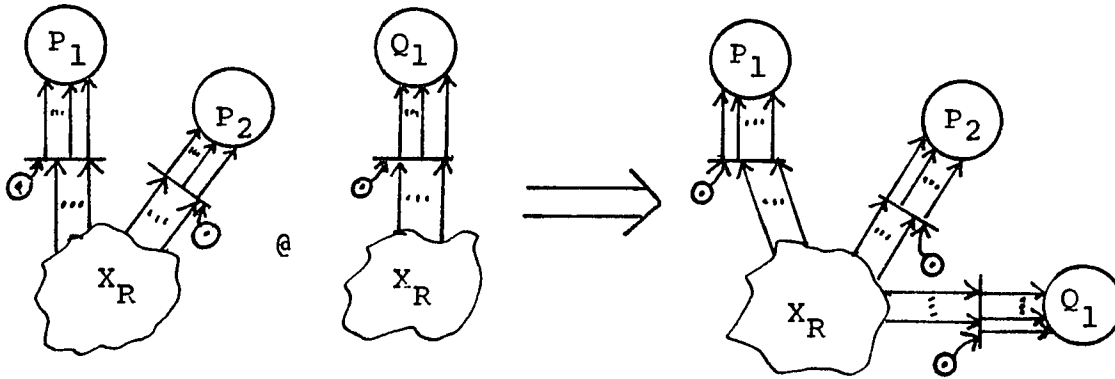


Figure 2-8

Observe that R-attachment results in an R-exclusion system and that $EX[P^R @ Q^R] \subseteq EX(P^R) \sqcup EX(Q^R)$.

Another important concern is whether or not an R-exclusion system is deadlock free. We will say that an R-exclusion system P^R is deadlock free iff all firing sequences of P^R are a prefix of some executable firing sequence in $EX(P^R)$.

We informally observe that given two deadlock free R-exclusion systems, their R-attachment results in a deadlock free R-exclusion system.

The usefulness of the concepts presented in this section are evident in Section 3.2 where we consider Data Base Systems.

CHAPTER 3

APPLICATIONS OF PETRI NETS

3.1 Protection Systems

The protection of one's information has always been a desirable, if not necessary, property of an information processing system. In a multiuser computer system that allows controlled sharing of information, protection mechanisms have come to be regarded as critical components. We intuitively define the protection afforded by a computer system to be all components of that system, both hardware and software, which make decisions regarding the rights of subjects (active entities of the system) to use objects (pieces of information in the system). As an example, we may think of a program as a subject when it is executing and a data file as an object which the subject might want to read.

In this section, we seek to formally model protection systems in order to answer questions about them, such as:

- i) Can subject s ever process object o in any way?
In a specific way?
- ii) Can object o ever be processed in some specific way?
- iii) Can subject s_1 transfer to another subject s_2 the right to process object o ?

In particular, we consider models of protection systems that seek to protect information by controlling access to that information.

We consider the work of Harrison, Ruzzo and Ullman [17].
They more formally define a protection system as follows:

Definition 4.1

A protection system consists of the following parts:

- i) a finite set R of generic rights, and
- ii) a finite set C of commands of the form:

$$\begin{array}{l}
 \text{command } \alpha (X_1, X_2, \dots, X_k) \\
 \left. \begin{array}{l}
 \text{if } r_1 \text{ in } (X_{s_1}, X_{o_1}) \text{ and} \\
 \vdots \\
 r_m \text{ in } (X_{s_m}, X_{o_m})
 \end{array} \right\} \text{ CONDITIONS} \\
 \\
 \text{then} \\
 \left. \begin{array}{l}
 op_i \\
 \vdots \\
 op_n
 \end{array} \right\} \text{ BODY} \\
 \\
 \text{end}
 \end{array}$$

They define each op_i to be one of the following primitive operations:

enter r into (x_s, x_o)	create object x_o
delete r from (x_s, x_o)	destroy subject x_s
create subject x_s	destroy object x_o

They go on to define the instantaneous description of a protection state:

Definition 4.2

A configuration of a protection system is a triple (S, O, P) where S is the set of current subjects, O is the set of current objects, $S \subseteq O$, and P is an access matrix with a row

for every subject in S and a column for every object in O .
 $P(s,o)$ is a subset of R giving the rights possessed by subject s to object o .

As examples, consider the commands `CREATE` and `CONFER_READ`.
READ:

<u>command</u> <code>CREATE(s,o)</code>	<u>command</u> <code>CONFER_READ(s₁,s₂,o)</code>
create object o	<u>if</u> own in $P(s_1,o)$
enter "own" into $P(s,o)$	<u>then</u> enter "read" into
<u>end</u>	$P(s_2,o)$
	<u>end</u>

Note that "own" and "read" are examples of generic rights.

The `CREATE` command allows a subject to create a file. By so doing, that subject acquires the ownership of that file. `CONFER_READ` allows the owner of a file to confer the generic right "read" to another subject. Notice that a subject having the "read" right but not the "own" right to some object could not confer that "read" right to another subject for that object.

It is this model of protection systems that motivates our use of Petri Nets in the investigation of problems in protection systems. The main advantages we see in using Petri Nets are that:

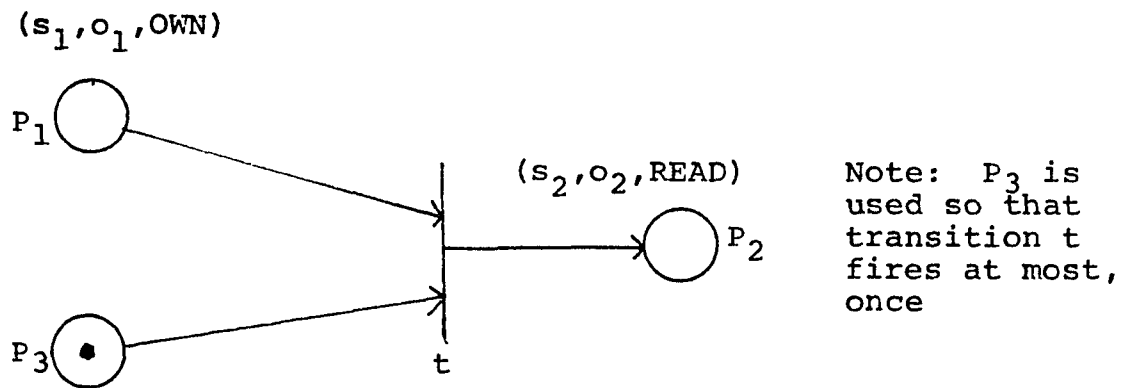
- 1) General proofs about correctness of protection systems need to be shown within a system framework allowing for concurrent mode of operation, and
- 2) Problems in protection systems can be related to reachability, liveness, and other Petri Net con-

cepts which are heavily under investigation and for which some results are already known. For instance, the concept of a subject holding a right to an object can be translated to a place holding a token. The question of whether a subject can ever access an object becomes a reachability question.

Our initial attempt to model the protection system of Harrison et al. using Petri Nets results in the "plug-in" model which follows.

We represent the holding of a generic right by the presence or absence of a token in a place $(s,o,r) \in S \times O \times R$. That is, subject s has right r to object o iff place (s,o,r) has a token. Every command in C can be represented by a Command Net which can be plugged into the appropriate places in the Protection Net when that particular command net is invoked. (This "plugging in" corresponds to the replacement of formal parameters by actual parameters, as in the subroutine-like mode of operation in the Harrison et al. model.)

Consider the `CONFER_READ` command discussed previously. The following Petri Net can implement it by "plugging" into places (s_1,o_1,own) and $(s_2,o_2,read)$.



In general, we define a Petri Net Protection System (PNPS) to consist of:

- i) a finite set of generic right places, and
- ii) a finite set of Command Nets of the form:

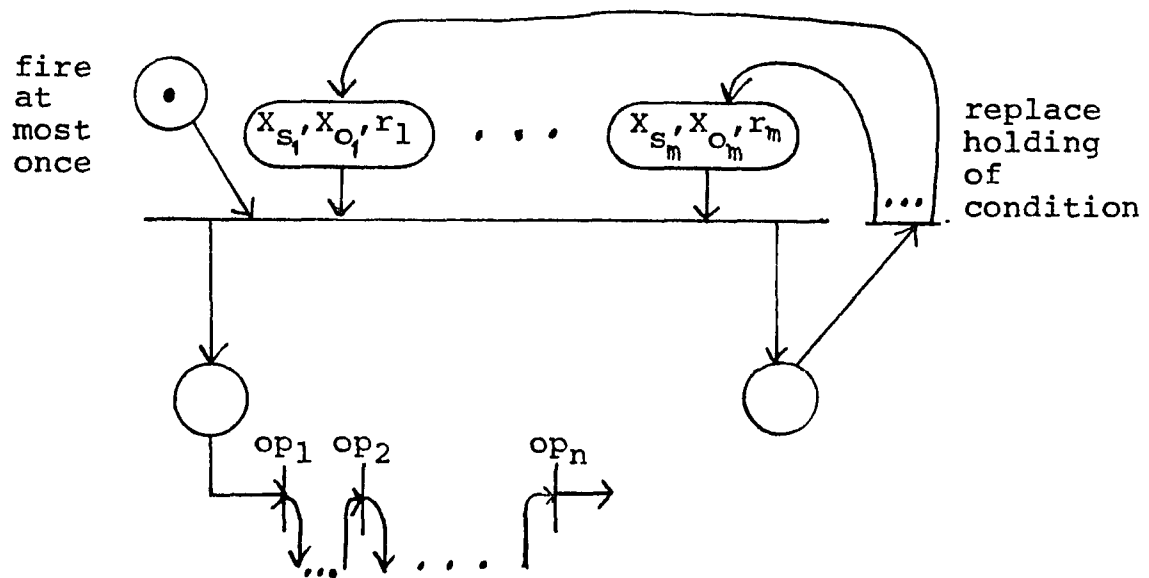
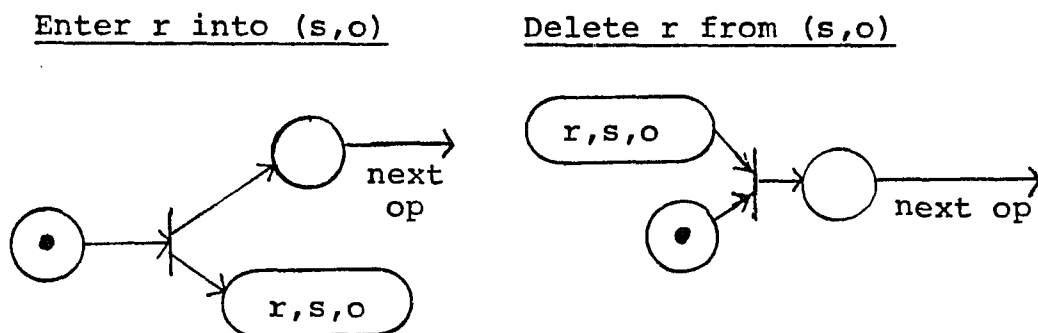


Figure 3-1

The operations enter and delete are handled by the following subnets:



Consider the following command which allows a subject, s_2 , with the read right to object o and the i-read right (indirect read) to subject s_1 , to allow s_1 to read object o without letting s_1 keep the read right.

```

command IREAD( $s_1, s_2, o$ )
  if
    read in ( $s_2, o$ ) and
    iread in ( $s_1, s_2$ )
  then
    enter read into ( $s_1, o$ )
    delete read from ( $s_1, o$ )
  end

```

Then the command net for IREAD is as follows:

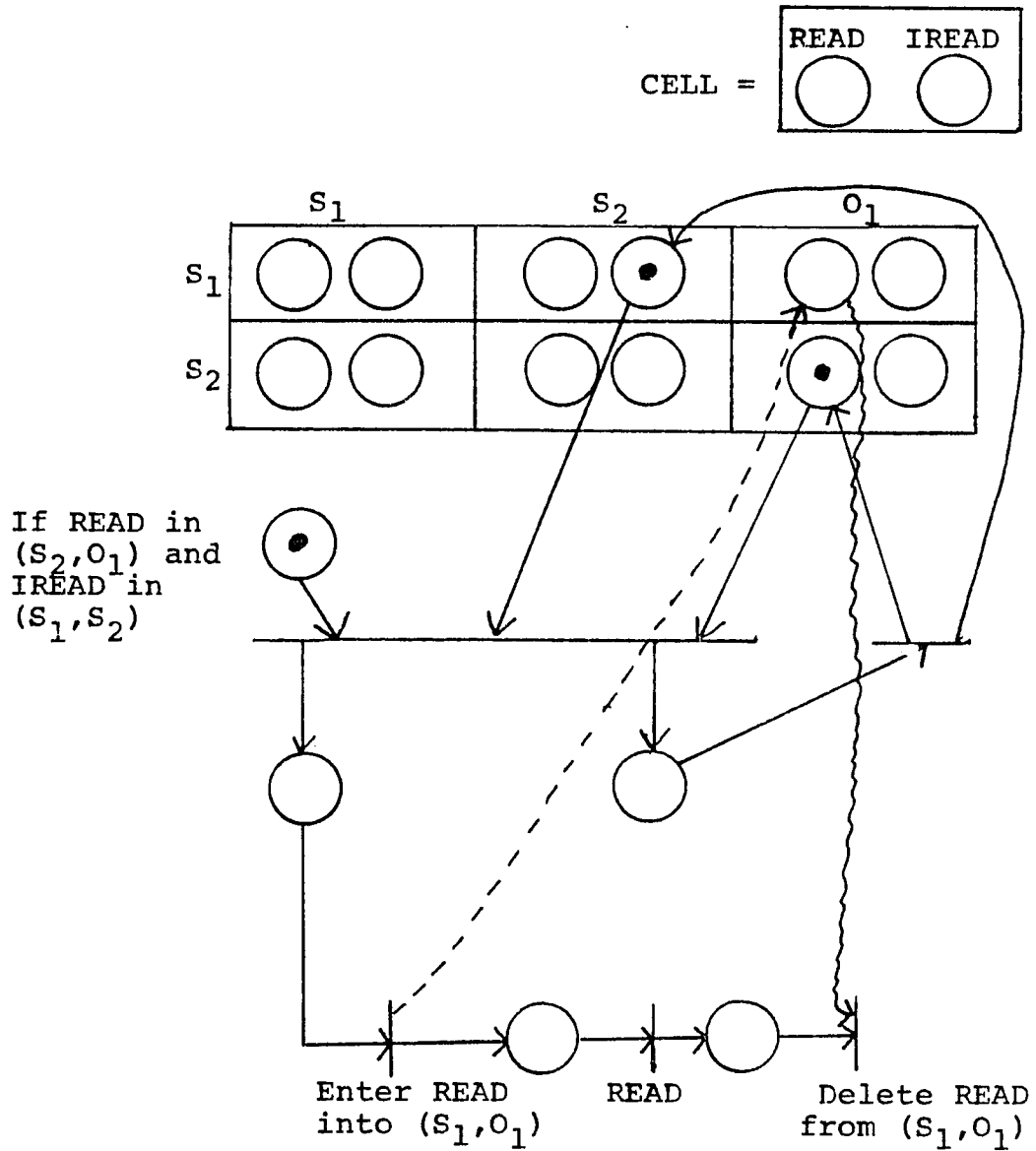


Figure 3-2

The create and destroy operations present a problem in implementation. These operations have to be handled either by dynamically altering the net (adding or deleting places and transitions) or by placing a bound on the number of subjects and objects. The latter would not allow for investigat-

ing general cases; therefore, it is not acceptable. Our extension of Petri Net Theory allows us to investigate these cases. We call the Petri Net models arising from these investigations "plug-in" models.

We want to be able to decide questions such as: for a given protection system and access matrix, can a subject s ever have a certain right r to some object o (i.e., can the place (s,r,o) ever be nonzero)?

We will illustrate our ideas using a well-known protection system. Consider the following Take-Grant System investigated by Lipton and Snyder [23]. They define a protection graph as a finite directed loop free graph with each arc labeled by a nonempty subset of $\{r,w,c\}$. They point out that while the graph will be a formal model one can intuitively think of a node as representing some system entity and then interpret r,w,c to represent read, write and call activities. The particular protection model studied is given by a protection graph and a set of "rewriting rules" specifying how the protection graph may change states. The Take-Grant System is now completed by five rewriting rules given in Figure 3-3.

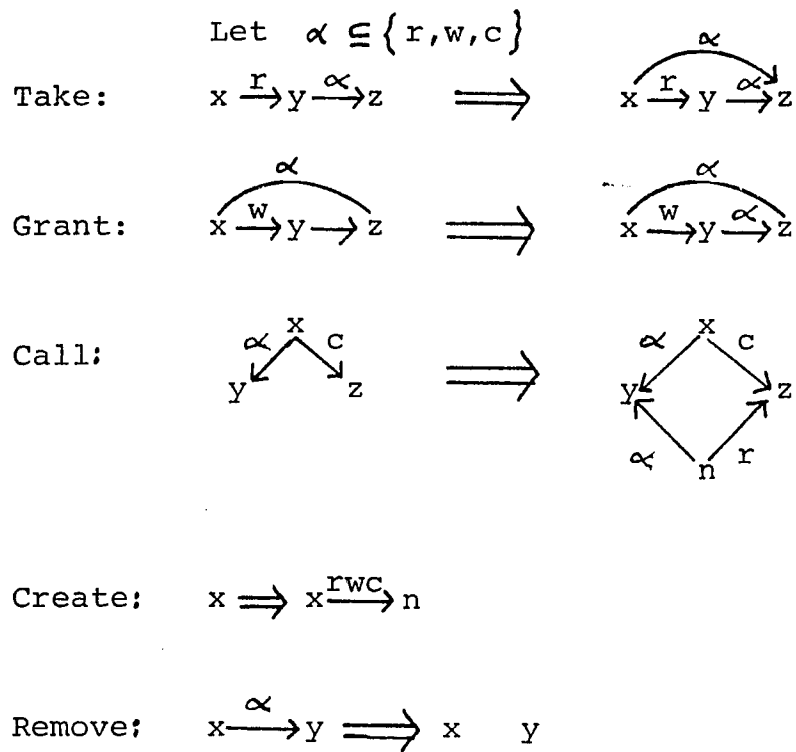
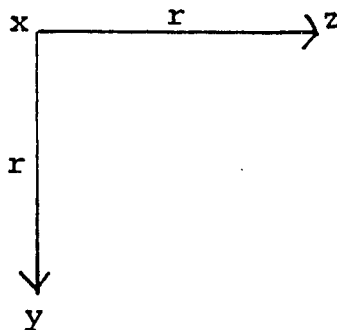


Figure 3-3

Using this protection model we are able to answer questions concerning the possible rights a node may acquire to another node by application of the rewriting rules. In particular Lipton and Snyder consider the following graph:



They pose the question, "Is there a sequence of rule applications that leads to a graph with a read arc from y to z?" They then show that in fact such a sequence exists. It is given below.

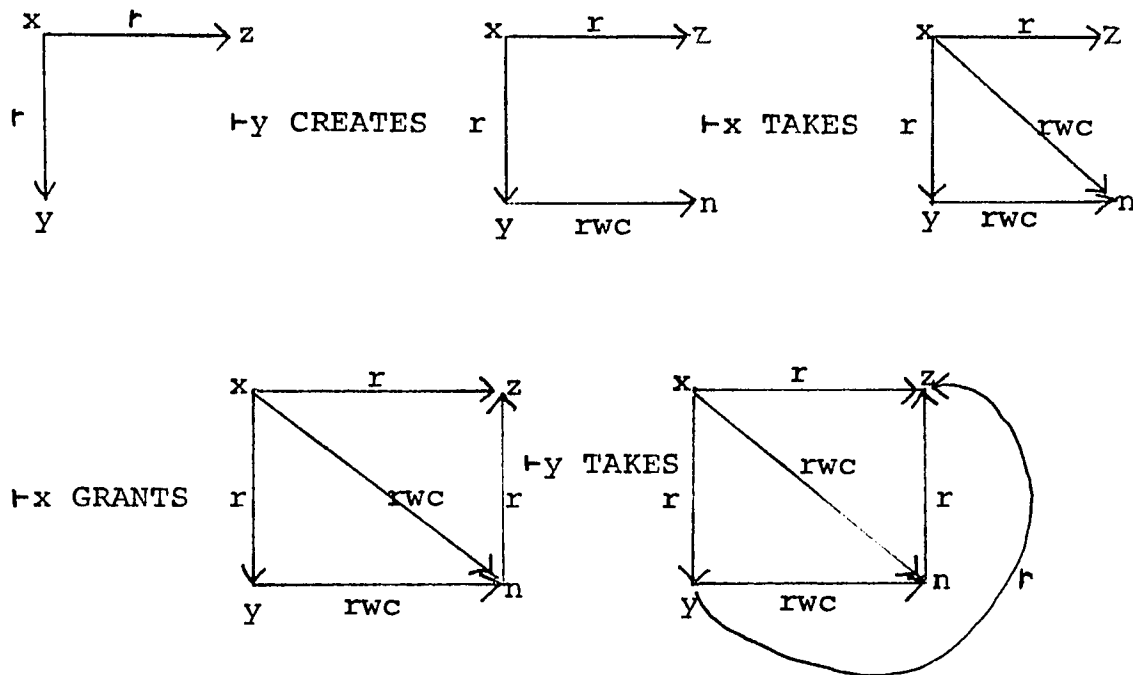
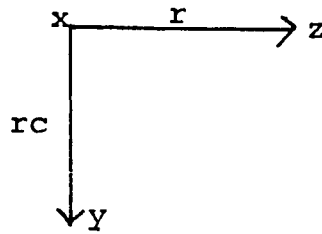


Figure 3-4

In considering the above sequence we note that it is a type of reachability problem. We ask the question: Given the initial graph (marking), is there some (firing) sequence which leads to a new graph (marking) in which y can read z? In translating the concepts embodied in the Take-Grant System to Petri Net Theory we proceed as follows: for each pair of nodes x,y such that there is an arc from x to y we denote the rights by the coefficient of xy. Each node is

also a term in the marking. For example, the labelled directed graph



is represented as $x+y+z+(r)xz+(rc)xy$.

We now present the Take-Grant System rules by means of Petri Net Theory.

Let $\alpha \in \{r, w, c\}$

Take: $x+y+z+(r)xy+(\alpha)yz \Rightarrow x+y+z+(r)xy+(\alpha)yz+(\alpha)xz$

Grant: $x+y+z+(w)xy+(\alpha)xz \Rightarrow x+y+z+(w)xy+(\alpha)xz+(\alpha)yz$

Call: $x+y+z+(\alpha)xy+(c)xz \Rightarrow x+y+z+n+(\alpha)xy+(c)xz+(\alpha)ny + (r)nz$

Create: $x \Rightarrow x+(rwc)xn$

Remove: $x+y+(\alpha)xy \Rightarrow x+y$

Figure 3-5

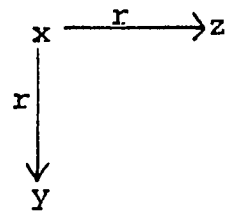
For the graph

we can translate the

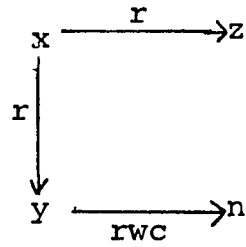
question "can y read z?" as the following reachability

problem: "Can we reach a marking such that the coefficient of yz contains r ?"

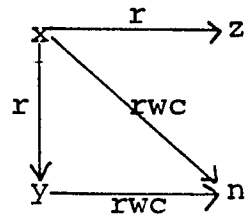
Figure 3-6 shows how the sequence given in Figure 3-4 can be viewed as a reachability problem.



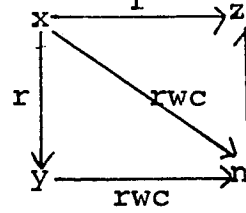
⊢ y Creates



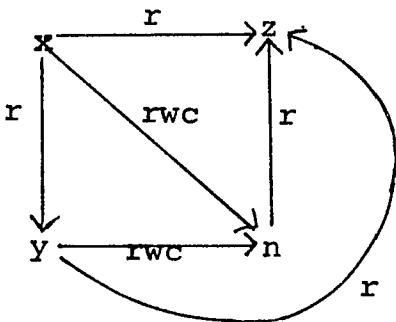
⊢ x Takes



⊢ x Grants



⊢ y Takes



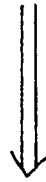
$$x+y+z+(r)xz+(r)xy$$



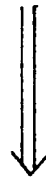
$$x+y+z+n+(r)xz+(r)xy+(rwc)yn$$



$$x+y+z+n+(r)xz+(r)xy+(rwc)yn+(rwc)xn$$



$$x+y+z+n+(r)xz+(r)xy+(rwc)yn+(rwc)xn \\ +(r)nz$$



$$x+y+z+n+(r)xz+(r)xy+(rwc)yn+(rwc)xn \\ +(r)nz+(r)yz$$

Figure 3-6

3.2 Data Base Systems

In attempting to resolve various problems which arise in data base systems, Codd [4] proposed a relational model.

Given sets $D_1, D_2, \dots, D_n$ we can define a relation R on these n sets where $R \subseteq D_1 \times D_2 \times \dots \times D_n$. The sets $D_1, D_2, \dots, D_n$ are called the domains of R . The data base is a collection of these relations.

One method of querying the data base is through relational algebra [7]. In such a query system, a query is formulated as an algebraic expression where the operations are over relations.

Consider the sample data base in Figure 3-7.

EMP	NAME	SAL	MGR	DEPT
	JONES	8K	SMITH	HOUSEHOLD
	ANDERSON	6K	MURPHY	TOY
	MORGAN	10K	LEE	COSMETICS
	LEWIS	12K	LONG	STATIONARY
	NELSON	6K	MURPHY	TOY
	HOFFMAN	16K	MORGAN	COSMETICS
	LONG	7K	MORGAN	COSMETICS
	MURPHY	8K	SMITH	HOUSEHOLD
	SMITH	12K	HOFFMAN	STATIONARY
	HENRY	9K	SMITH	TOY

SALES	DEPARTMENT	ITEM	SUPPLY	ITEM	SUPPLIER
	STATIONARY	DISK		PEN	PARKER
	HOUSEHOLD	PEN		PENCIL	BIC
	STATIONARY	PENCIL		INK	PARKER
	COSMETICS	LIPSTICK		PERFUME	REVLON
	TOY	PEN		INK	BIC
	TOY	PENCIL		DISH	DUPONT
	TOY	INK		LIPSTICK	REVLON
	COSMETICS	PERFUME		DISH	BIC
	STATIONARY	INK		PEN	REVLON
	HOUSEHOLD	DISH		PENCIL	PARKER
	STATIONARY	PEN			
	HARDWARE	INK			

TYPE	ITEM	COLOR	SIZE
	DISH	WHITE	M
	PENCIL	RED	L
	LIPSTICK	RED	L
	PERFUME	WHITE	L
	PEN	GREEN	S
	PENCIL	BLUE	M
	INK	GREEN	L
	INK	BLUE	S
	PENCIL	BLUE	L

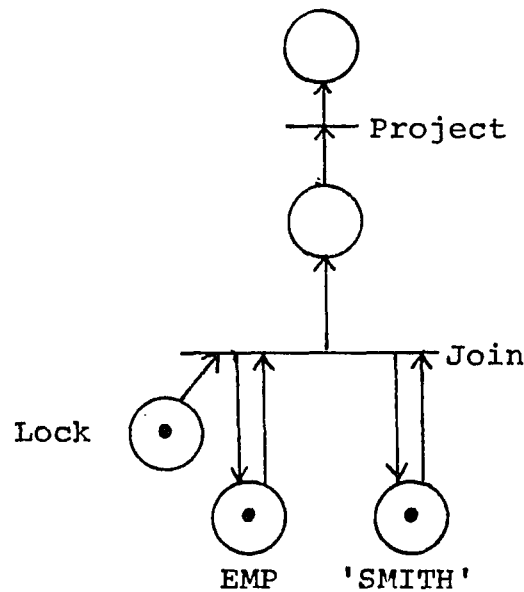
Sample Data Base

Figure 3-7

The user queries the data base using a query language. One such language uses relational algebra to express queries [7]. Consider the query "What are the names and salaries of the employees managed by Smith?" This can be expressed by the following expression:

Project [Join [EMP, 'SMITH'] over (MANAGER)] over (NAME;
SALARY)

We can model this query with the following Petri Net:



We associate with each transition a relational operation. The places labelled 'EMP' and 'SMITH' denote the relations 'EMP' and the constant relation 'SMITH'. The presence of a token denotes the availability of a relation. The input place labelled 'LOCK' ensures that the query doesn't continuously execute.

In modeling a data base system we are interested in:

- (i) Insuring that the query operations are performed in the correct order;
- (ii) Insuring that relations are locked and unlocked to prevent inconsistencies from occurring;
- (iii) Insuring that deadlock cannot occur;
- (iv) Allowing operations to be performed concurrently.

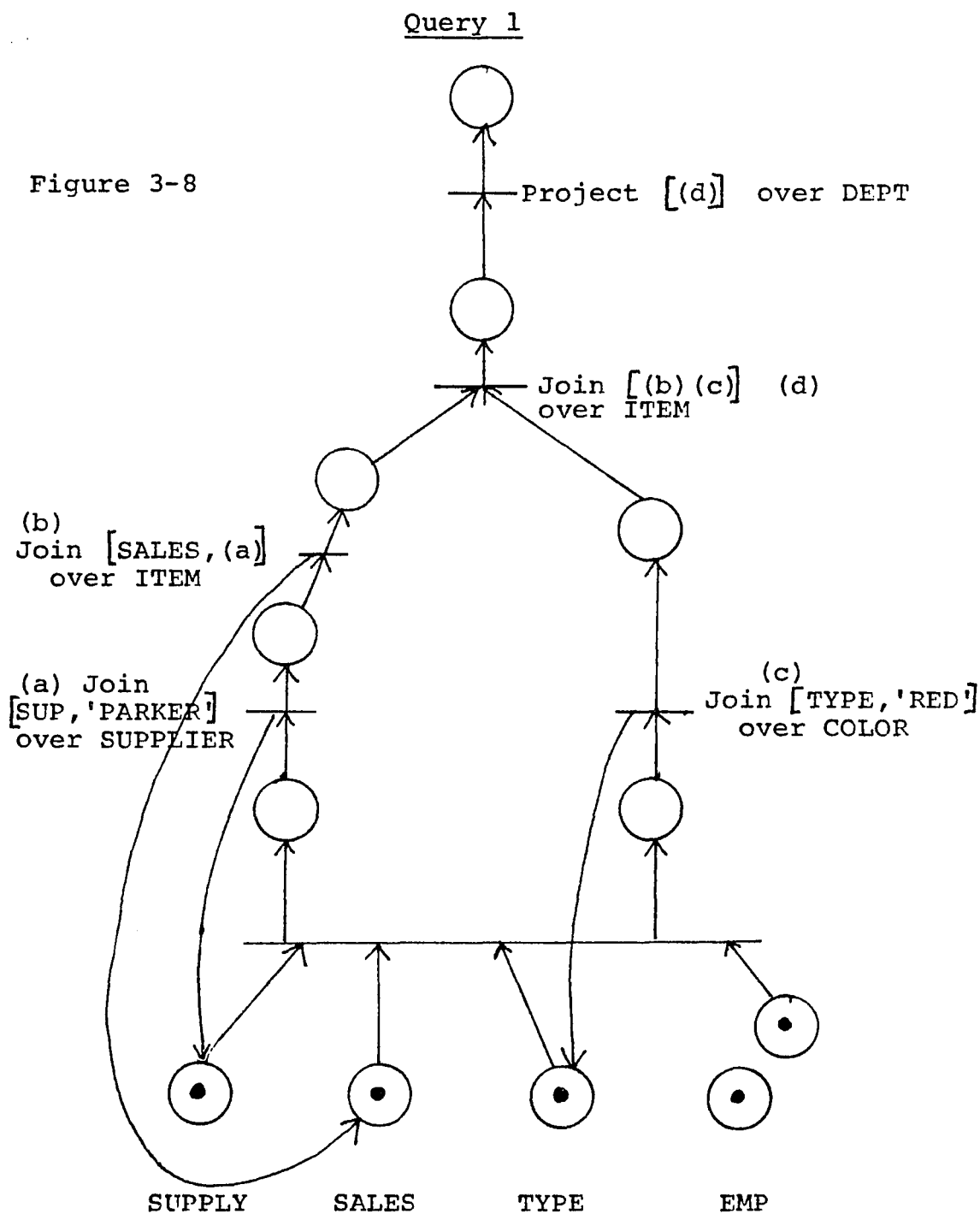
Consider the two queries given in Figures 3-8 and 3-9, and the resultant Petri Net model in Figure 3-10. Note that the firing rule insures that the operations of each query are performed in the proper order. The use of a single transition at the base of each query to acquire the necessary relations (and their subsequent release only after they are no longer needed by that query) insures consistent updates [11]. Note that each query is a P^R -Petri Net. Also note that the firing sequences that are possible are executable firing sequences. Hence each query is deadlock free and the attachment of the queries is deadlock free. (See 2.8).

Finally, we ask if concurrency among the relational operations is possible given all of the above. To answer this we use our SPLIT procedure on the Petri Net resulting from the attachment of the queries. The Petri Net in Figure 3-10 shows the SPLIT of the net resulting from the attachment of the queries given in Figure 3-8 and Figure 3-9.

By analyzing the net of Figure 3-10 we can answer questions concerning concurrency of relational operations. In particular, we compute $C(P)$ to get this information.

Query 1: What departments sell red items which are supplied by Parker?

Figure 3-8



Query 2: What items are sold by Jones?

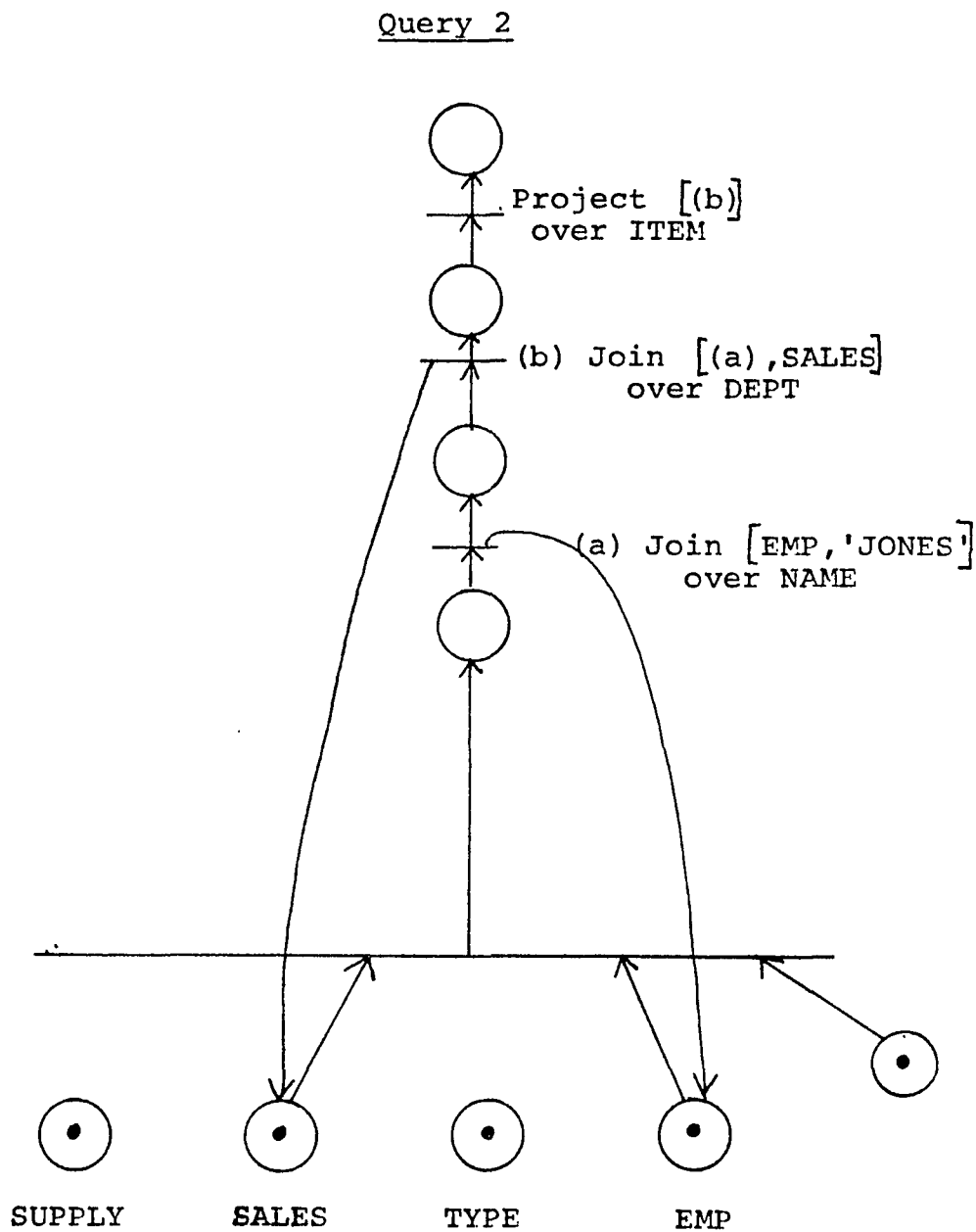
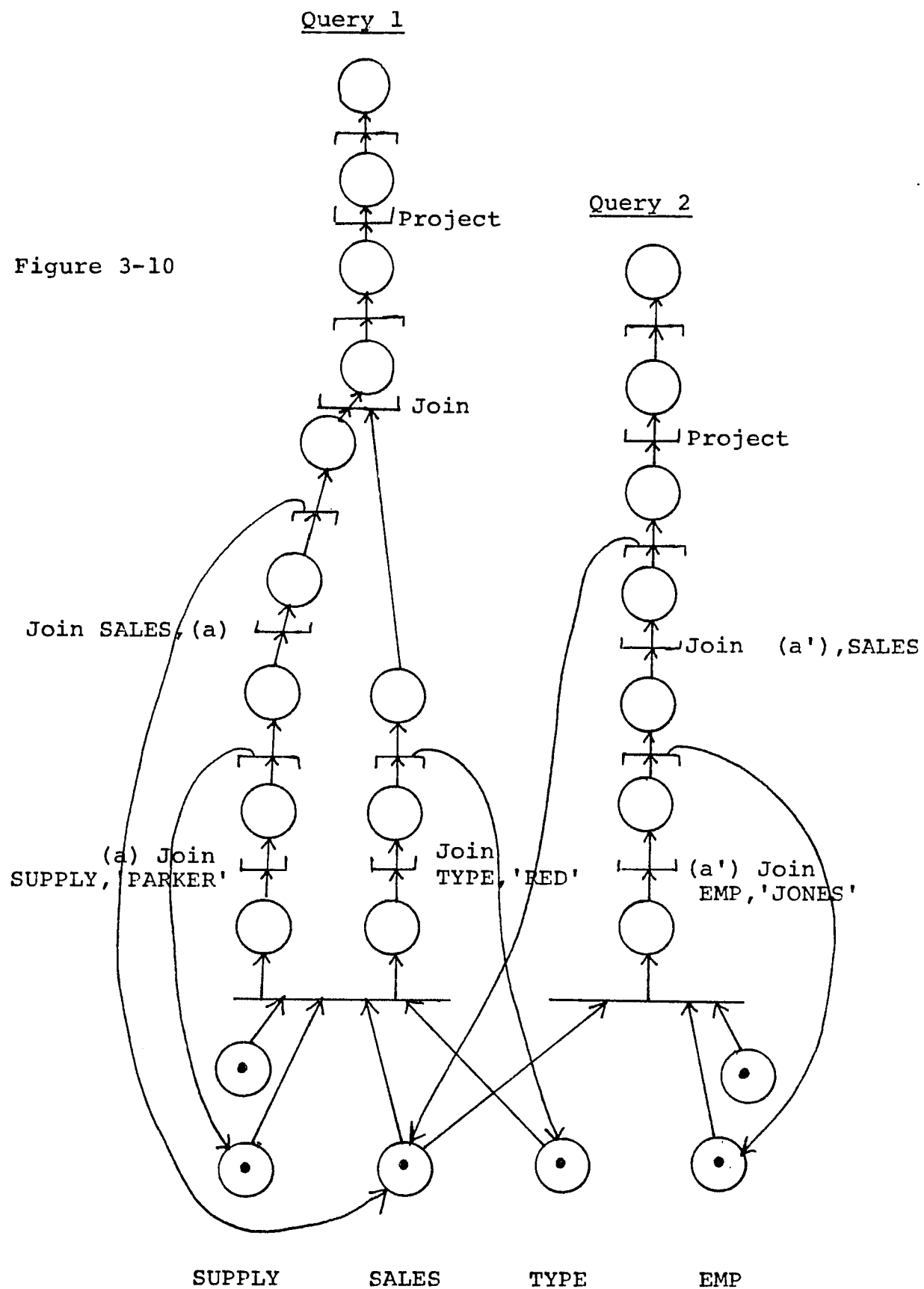


Figure 3-9

Figure 3-10



Informally, we suggest that the split net can be reduced to the following net (Figure 3-11) which can be examined more easily and provides the same information about concurrency. Notice that if there is a marking M such that $M(P_1) \neq 1$ and $M(P_6) \neq 0$, then the join given by P_6 can be concurrent with either the project or join given by P_1 .

Reduced Query Net System

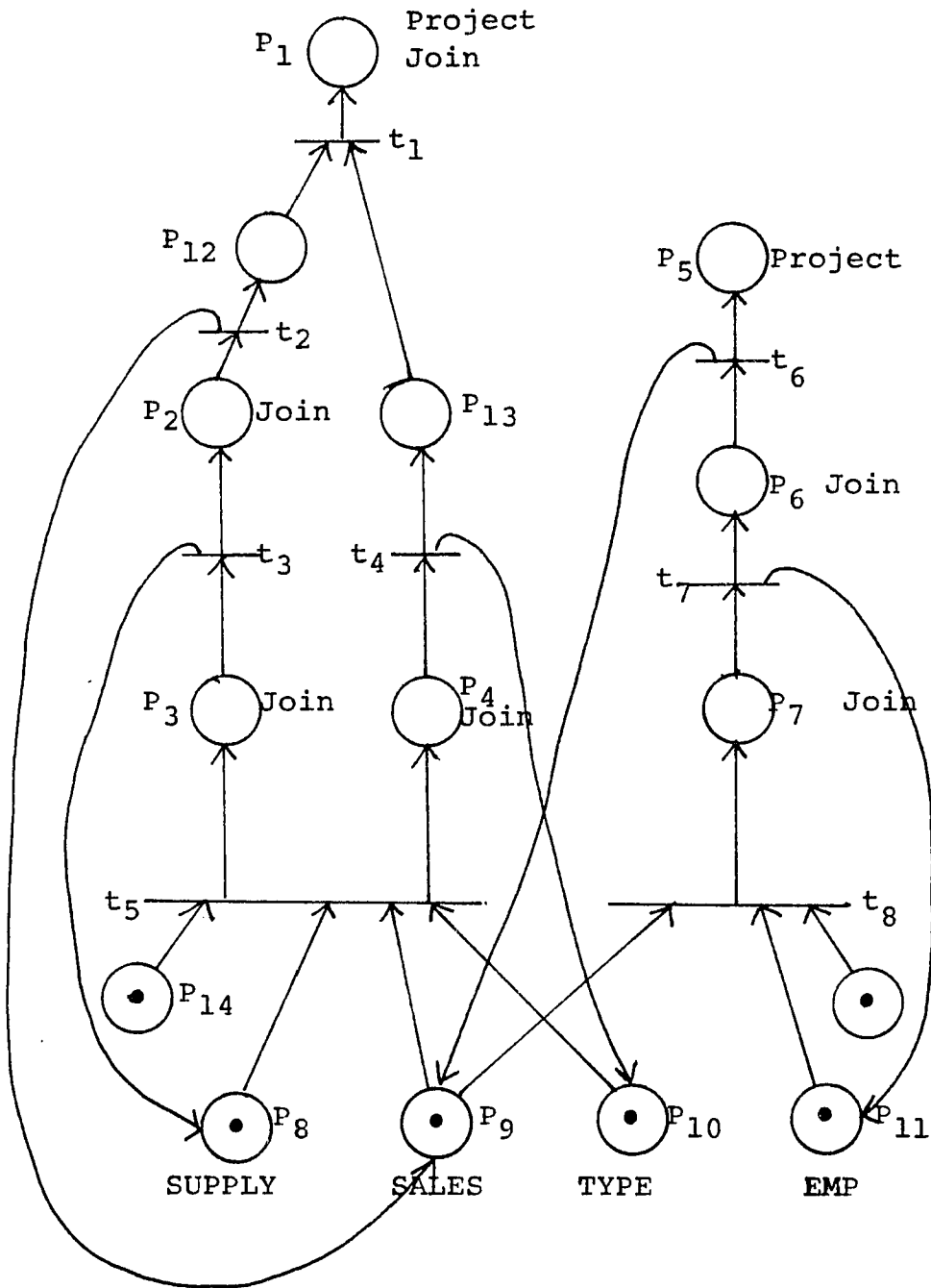


Figure 3-11

3.3 The Banker's Problem

The Banker's Problem [14] is concerned with the allocation of resources.

We consider a resource allocation system to consist of: 1 banker, n processes, and r resource types.

We represent a resource vector by an r -tuple of non-negative real numbers; the i^{th} component corresponding to the amount of the i^{th} resource.

The state of the resource system is given by the $n+1$ -tuple $S=(x, P_1, P_2, \dots, P_n)$ where x is the resource vector of the banker's status, i.e., the resources presently available for allocation. The status of the i^{th} process P_i consists of a termination request R_{i0} and any finite number of partial requests $R_{i1}, \dots, R_{im_i}$ (i.e., $P_i=(R_{i0}, \dots, R_{im_i})$, $m_i \geq 0$). Each request is given by a pair of resource vectors $R_{ij}=(y_{ij}, z_{ij})$, such that $y_{i0} \geq y_{i1} \geq \dots \geq y_{im_i}$ for each process P_i . We call y_{kj} the cost of R_{ij} , z_{ij} the return of R_{ij} , and $(z_{ij}-y_{ij})$ the profitability of R_{ij} .

The requests $R_{i0}, R_{i1}, \dots, R_{im_i}$ are interpreted as follows: The banker in order to complete its work must eventually allocate all of y_{i0} to P_i . When this occurs P_i can complete its work, return z_{i0} to the banker and terminate. The banker can allocate $y_{i1} \leq y_{i0}$ to P_i so that P_i can perform part of its work and return z_{i1} to the banker.

Subsequently, P_i will need to get back z_{i1} and

$(y_{i0}-y_{i1})$ which has not yet been allocated to P_i , in order to complete its work. If the banker allocates $y_{i2} < y_{i1}$ then P_i can perform a shorter part of its work, halt, and return z_{i2} to the banker. Note that $R_{i3}, \dots, R_{im_i}$ are requests to perform successively shorter initial segments of P_i 's work.

In summary, we can view a resource system's state S as follows:

$$S = \left\{ \begin{array}{l} x \\ (y_{10}, z_{10}) \dots (y_{1m_n}, z_{1m_n}) \\ \vdots \\ (y_{n0}, z_{n0}) \dots (y_{nm_n}, z_{nm_n}) \end{array} \right.$$

where x, y_{ij}, z_{ij} are R -tuples of real numbers such that

$$AX1: \quad x, y_{ij}, z_{ij} \geq 0,$$

$r, n, m_1, \dots, m_n$ are integers such that

$$AX2: \quad r \geq 1, \quad n, m_1, \dots, m_n \geq 0,$$

for each $i=1, \dots, n$,

$$AX3: \quad y_{i0} \geq y_{i1} \geq \dots \geq y_{im_i}$$

For a given resource system we are interested in determining if an allocation strategy exists such that all processes can terminate. Consider the following example [14]. There are 2 resources and the banker initially has $x=(20,0)$ units of the resource. Notice that S_2 is deadlocked.

	S_0		S_1		S_2	
banker's stockpile x	(20, 0)		(10, 1)		(0, 2)	
	cost	return	cost	return	cost	return
Process # 1 requests						
termination R_{10}	(20, 1) $\rightarrow$ (30, 2)		(10, 2) $\rightarrow$ (30, 2)		(10, 2) $\rightarrow$ (30, 2)	
partial R_{11}	(20, 0) $\rightarrow$ (10, 1)					
Process # 2 requests						
termination R_{20}	(30, 1) $\rightarrow$ (40, 2)		(30, 1) $\rightarrow$ (40, 2)		(20, 2) $\rightarrow$ (40, 2)	
partial R_{21}	(10, 1) $\rightarrow$ (0, 2)		(10, 1) $\rightarrow$ (0, 2)			

Note— S_0 is current system resource status. S_1, S_2 result from allocation of R_{11}, R_{21} .

Figure 3-12

The aspects of the Banker's Problem which interest us follow:

- i) Processes compete for resources;
- ii) Deadlock is possible; and
- iii) Resources can be described by r -tuples of non-negative real numbers

We consider a less general Banker's Problem as given in Gold [14]. What we show is how our algebraic Petri Nets can be used to study this problem.

In the restricted Banker's Problem each of the n processes states its maximum need. When this need is satisfied the process completes its work and returns (i.e., repays) its loan. The process can borrow from the banker one unit at a time. We say that the current state of the system is safe if it is possible for the banker to enable all of the processes to complete their work; otherwise, the state is called unsafe.

This version of the Banker's Problem can be viewed as an operating system lending resources to processes which return these resources when they finish.

Consider the example in Figure 3-13.

	$s_0=12$		$s_1=5$		$s_2=6$		$s_3=8$		$s_4=12$	
	<u>need</u>	<u>loan</u>	<u>need</u>	<u>loan</u>	<u>need</u>	<u>loan</u>	<u>need</u>	<u>loan</u>	<u>need</u>	<u>loan</u>
P_1	8	0	6	2	6	2	-	-	-	-
P_2	11	0	7	4	7	4	7	4	-	-
P_3	5	0	4	1	-	-	-	-	-	-
	(a)		(b)		(c)		(d)		(e)	

Figure 3-13

Notice that it is possible to go from a safe state to an unsafe state. Consider the state given by Figure 3-13(b). Had the banker lent two resource units to process P_1 or P_2 , an unsafe state would have resulted, which in this particular case is deadlock.

In Figure 3-14 we present a Petri Net model of the above Banker's Problem.

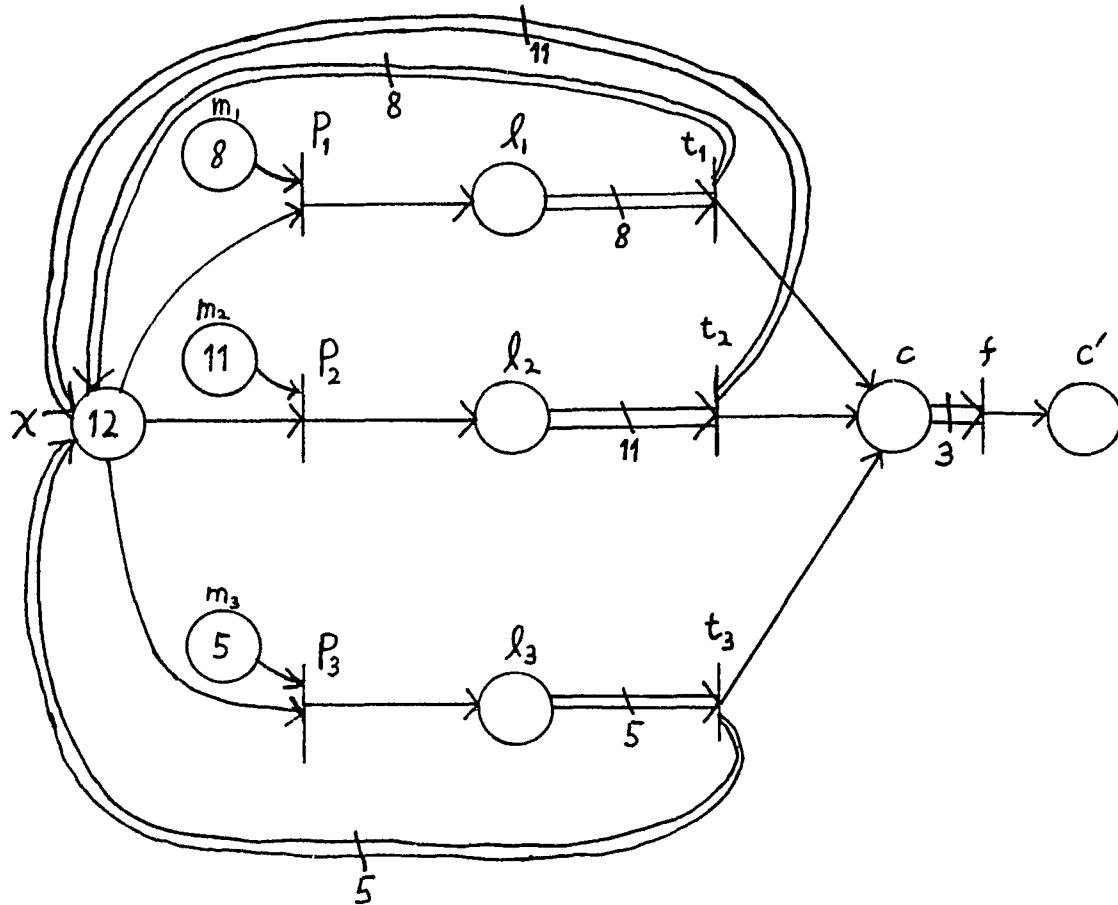


Figure 3-14

In this Petri Net model we represent process P_i as follows:

- i) m_i contains the number of resource units needed;
- ii) l_i contains the number of resource units lent;
- iii) t_i firing indicates that P_i completes. It returns the m_i units borrowed and increments place c (which holds the number of completed processes) by one.

Transition f can fire when all processes have terminated, placing a token in c' .

We make the following observations:

- i) A marking M (state) is safe iff there exists a firing sequence s such that $M \xrightarrow{s} M'$ and $M'(c')=1$.
- ii) For a "Banker's Net" P , an allocation strategy such that all processes terminate exists iff there exists $M \in R(P)$ such that $M(c')=1$.

We see how we can view safety as a reachability problem. If we consider that the resource units can be real numbers, the need for our algebraic Petri Nets becomes apparent. Furthermore we are interested in studying algebraic Petri Nets with markings of nonnegative real vectors.

CHAPTER 4

TOWARD A SYSTEM FOR THE DESIGN OF ASYNCHRONOUS CONCURRENT SYSTEMS

4.1 Introduction

In developing an asynchronous concurrent system a designer traditionally embeds control statements, such as Dijkstra's P and V commands, in the system modules. Using these synchronizing primitives the designer hopes to achieve a particular kind of concurrent relationship among the modules. For a large system the complexities can become prohibitive; debugging can be very lengthy and the outcome uncertain at best. As there is no general quantitative method for the designer to use in establishing the correctness of his solution, he must rely on program testing or simulation techniques.

Here we propose a system based on Petri Net Theory which will aid a designer in the following ways:

- Using a very high-level design language the user can specify how the system modules are to interact. The important issue here is that in using the design language the user only states what concurrencies may or may not take place and does not concern himself with the implementation of a suitable control mechanism. The concept here parallels the benefit of a query language where the user states to the system what is desired without specifying how to get the answer(s). Our design language essentially consists of statements which

establish constraints and other relationships between modules. One of the most difficult issues faced in concurrent programming is how to understand what concurrencies may be inadvertently inferred from the user's design. To help the user better understand his design, we built into our system a query facility with which the designer can get information about his design.

- The query facility the system provides is a very powerful tool available to the designer. Using this facility it is possible for the designer to ascertain the various possibilities for concurrency among the system modules. For instance, the user can learn that concurrency between two or more processes will never occur. With this facility the user can then "tune-up" the system for optimal performance. One can also determine if it is possible for some process to "die," i.e., if it is possible for a system module to become useless. It can also be determined if a system bottleneck can occur due to excessive demand of a system module.

- Once the user is satisfied with his design he can have the system compile his design, producing a control module. This control module will control the system's behavior according to the user's design statements. It can also give the user current "snapshots" of the system state, allowing him to modify his design accordingly. Other features for gathering information and statistics on system performance will be added to the control module.

It is also possible for the system to generate system

module "stubs" simulating the actual module execution times. The user can then simulate the system in order to get additional performance parameters, before actually committing the necessary resources to build it.

We will refer to our system by the acronym SYSDACS, System for the Design of Asynchronous Concurrent Systems.

4.2 The Design Language

Here we list a sample of the statements which the language will contain. These statements will be more formally described in the section on the compiler.

<processes> will represent a list of system module names.

- 1) PROCESSES <processes> ;
 - 2) <processes> PRECEDES [(n) OF] <processes> ;
 - 3) <processes> FOLLOWS [(n) OF] <processes'>;
 - 4) AT MOST (n) OF <processes> AT ANY TIME;
 - 5) MUTUAL EXCLUSION AMONG <processes> ;
 - 6) <processes> PRODUCER(S) FOR [(n) OF] CONSUMER(S) <processes'> [BUFFER (n)] ;
 - 7) ALLOW UP TO (n) CONCURRENT COPIES OF <processes> ;
 - 8) <processes> IN SEQUENCE;
-
- 1) PROCESSES <processes> ;

This statement declares the system processes. The system assumes that each process is asynchronous and independent of the others. It is the responsibility of the user to override this default through the use of other statements.

2) <processes> PRECEDES [(n) OF] <processes>;

This statement indicates precedence among processes. In the simplest case:

P₁ PRECEDES P₂;

indicates that an instance of P₁ must precede each instance of P₂. In a sense it chains them together: P₁ → ... → P₂. (In future implementation we may elect to have a statement DIRECTLY PRECEDES.)

3) <processes> FOLLOWS [(n) OF] <processes>;

This is similar to the PRECEDES statement. Consider:

P₁P₂ FOLLOWS P₃,P₄,P₅;

This indicates that only after an instance of P₃,P₄ and P₅ may P₁ and/or P₂ begin execution.

4) AT MOST (n) OF <processes> AT ANY TIME;

This statement is a form of mutual exclusion where we only allow n of the processes to be executing concurrently.

AT MOST 3 of P₁,P₂,P₃,P₄ AT ANY TIME;

will allow at most three of the processes to be executing, requiring the fourth to wait.

This statement establishes mutual exclusion for the processes.
It is equivalent to AT MOST 1 of $\langle \text{processes} \rangle$ AT ANY TIME;.

MUTUAL EXCLUSION AMONG P_1, P_2, P_3 ;

This statement establishes a producer-consumer relationship between processes₁ and processes₂. Consider:

This establishes P_1 as the producer and P_2 as the consumer.

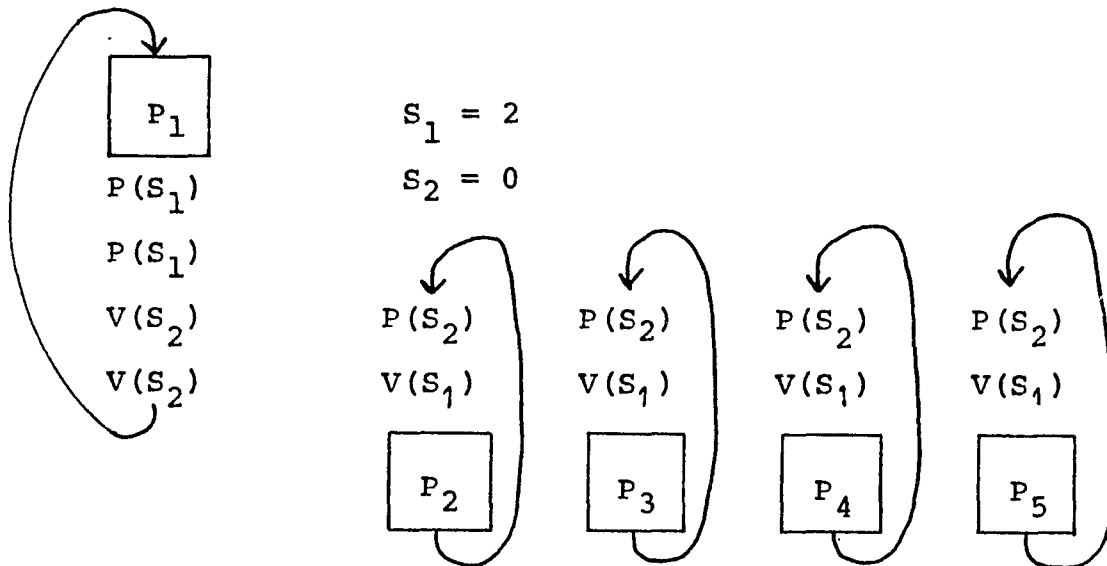
Using Dijkstra's semaphores we can implement this as follows:


$$V(S) \Rightarrow S \leftarrow S + 1$$

P_1 PRODUCER(S) FOR CONSUMER(S) P_2 BUFFER (3);
implies that P_1 can be up to 3 "instances" ahead of P_2 . The
same description as above applies except that S_1 is initial-
ized to 3.

Consider:

P_1 PRODUCER FOR 2 OF CONSUMERS P_2, P_3, P_4, P_5 ;
This implies that after P_1 produces, any of 2 of P_2, P_3, P_4
and P_5 will consume.



7) ALLOW (n) CONCURRENT COPIES OF <processes> ;

This statement will allow more than one concurrent version of <processes> . This assumes that <processes> are reentrant, or pure procedures.

Consider:

ALLOW 3 CONCURRENT COPIES OF P_1 ;

This statement will allow up to 3 copies of P_1 to be executing at the same time.

8) <processes> IN SEQUENCE;

This establishes a linear arrangement among the processes.

P_1, P_2, P_3 IN SEQUENCE;

will order the processes so that P_1 must occur before P_2 , etc. This statement can be built from a combination of PRECEDES statements.

4.3 The Compiler

The compiler for SYSDACS will translate an asynchronous concurrent system design program into a Petri Net. The transitions associated with the user's system modules will fire (i.e., execute) according to the user's design.

The compiler will be designed to formally translate each statement until a complete net for the system is built. Here we present some examples of how this will be done. However we have not yet developed a formal model.

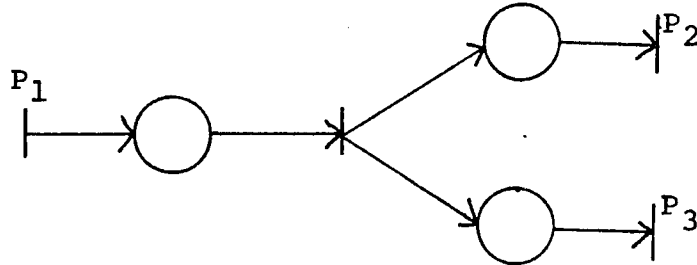
1) PROCESSES <processes> ;

This command informs the system about all processes. Each process will be represented as a transition.

2) <process> PRECEDES [(n) OF] <processes> ;

This statement indicates precedence among processes. Consider: P_1 PRECEDES P_2, P_3 ;.

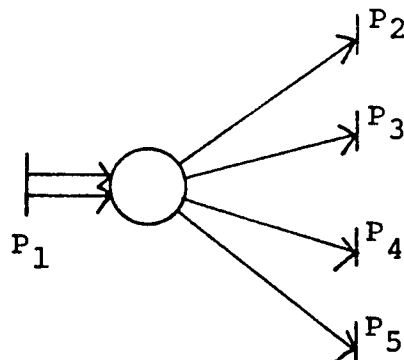
This is compiled as follows:



Note that both may fire after P_1 is finished. The use of [(n) OF] allows at most (n) of the processes to fire. This creates a conflict. The user may supply a special conflict task to decide which process(es) may fire. Consider:

P_1 PRECEDES 2 of P_2, P_3, P_4, P_5 ;.

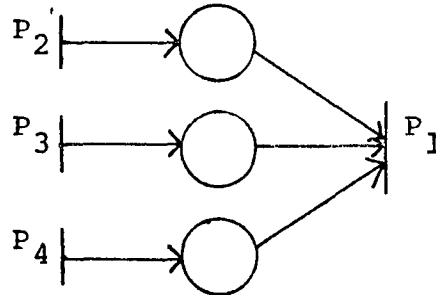
This would be compiled as follows:



3) <process> FOLLOWS [(n) OF] <processes>;

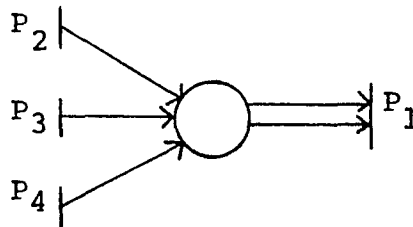
This command is similar to PRECEDES. Consider

P_1 FOLLOWS P_2, P_3, P_4 ;



The following example illustrates the similarity between the FOLLOWS statement and the PL/1 "WAIT" statement.

P_1 FOLLOWS 2 of P_2, P_3, P_4 ;



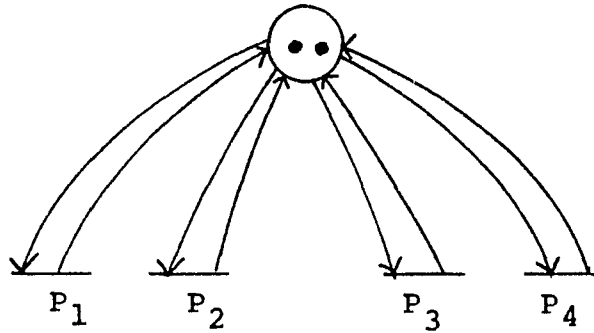
Note that if all are required to precede P_1 then one can use the following:

P_1 FOLLOWS ALL OF P_2, P_3, P_4 ;

4) AT MOST (n) OF <processes> AT ANY TIME;

This statement allows a user to limit the number of processes that can be active (executing) at any time. This can be used to "fine tune" a design if it is found that certain processes inefficiently execute together.

AT MOST 2 of P_1, P_2, P_3, P_4 AT ANY TIME;



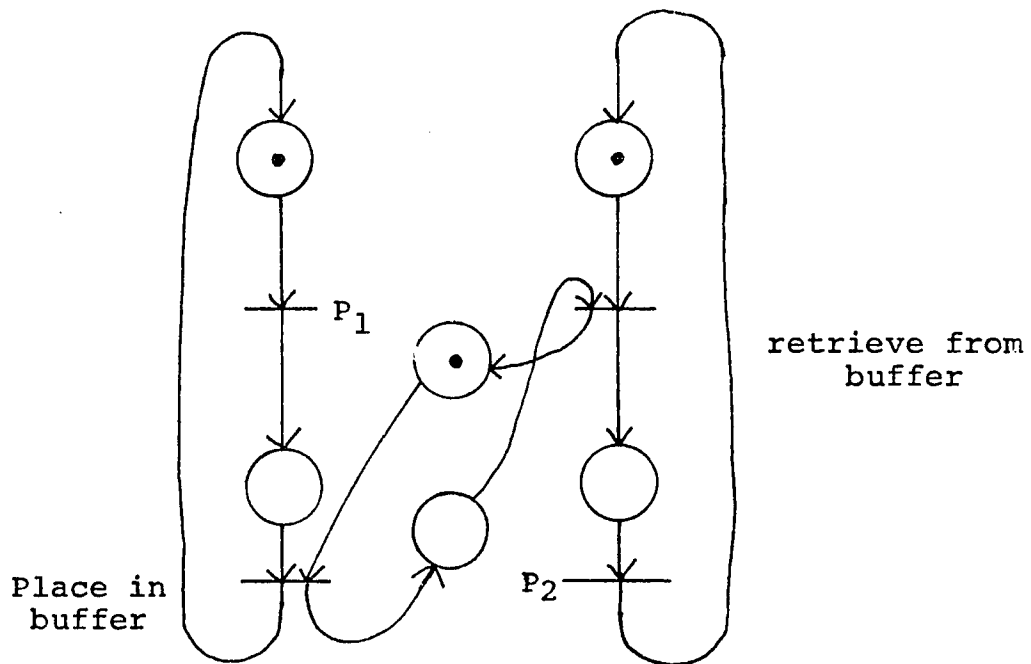
5) MUTUAL EXCLUSION AMONG $\langle \text{processes} \rangle$;

This translates to:

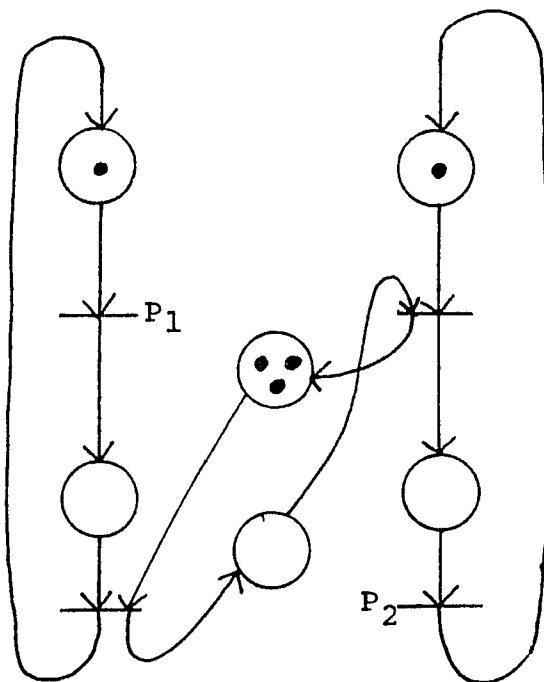
AT MOST 1 OF $\langle \text{processes} \rangle$ AT ANY TIME; (see (4) above)

6) $\langle \text{processes} \rangle$ PRODUCER(S) FOR $[(n) \text{ OF}]$ CONSUMER(S)
 $\langle \text{processes}' \rangle$ [BUFFER (n)];

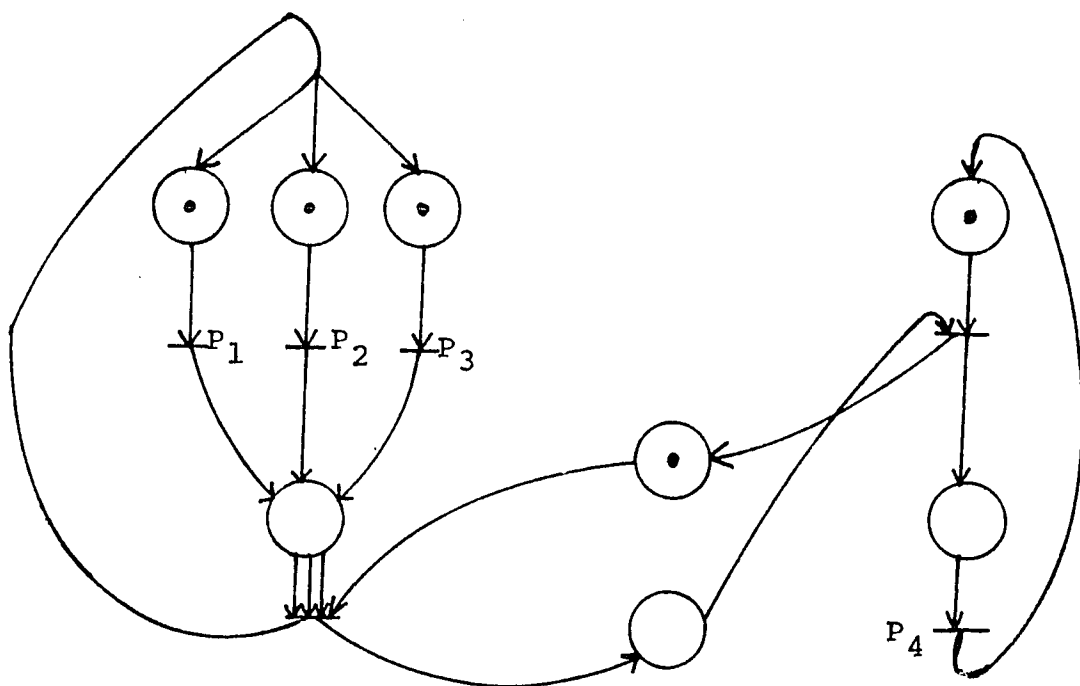
This sets up a classical producer-consumer relationship between $\langle \text{processes} \rangle$ and $\langle \text{processes}' \rangle$. The "BUFFER" option allows for a backlog of n items to build up waiting to be consumed. The "OF" option allows for the situation where an item produced can be consumed by any of the consumers, not all of them.



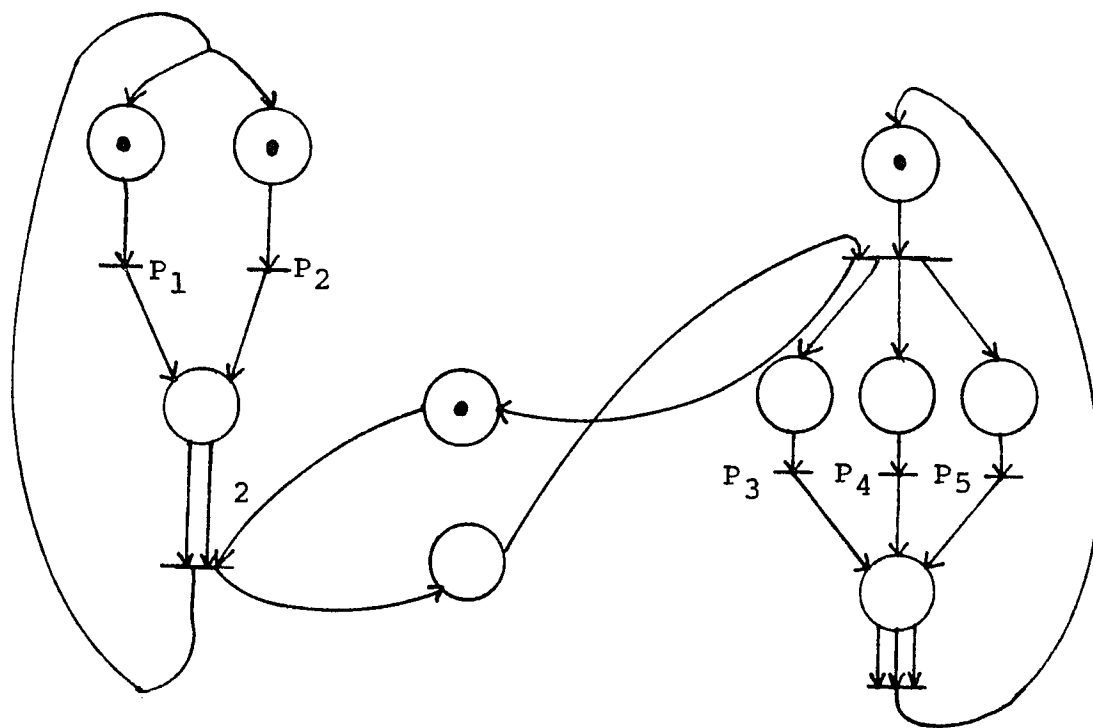
P_1 PRODUCER FOR CONSUMER P_2 ;



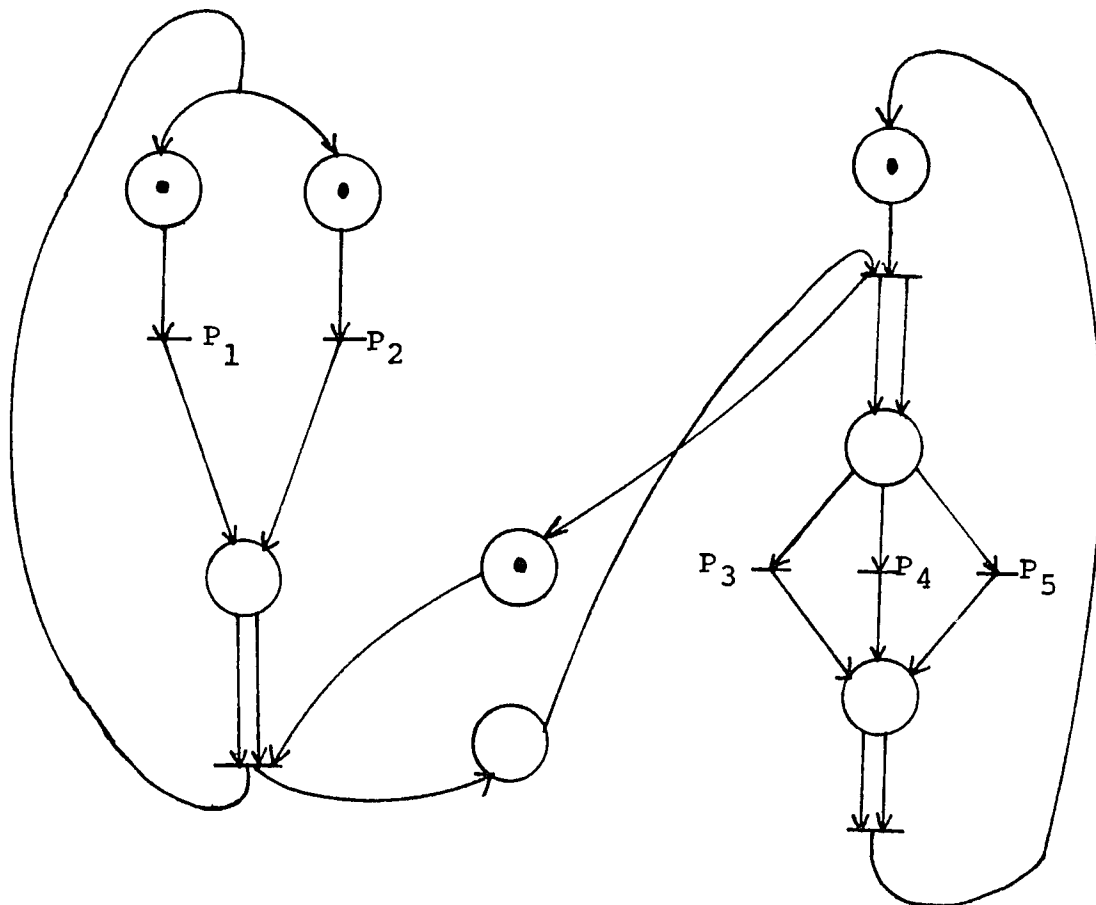
P_1 PRODUCER FOR CONSUMER P_2 BUFFER 3;



P_1, P_2, P_3 PRODUCERS FOR CONSUMERS P_4 ;



P_1, P_2 PRODUCERS FOR CONSUMERS P_3, P_4, P_5 ;



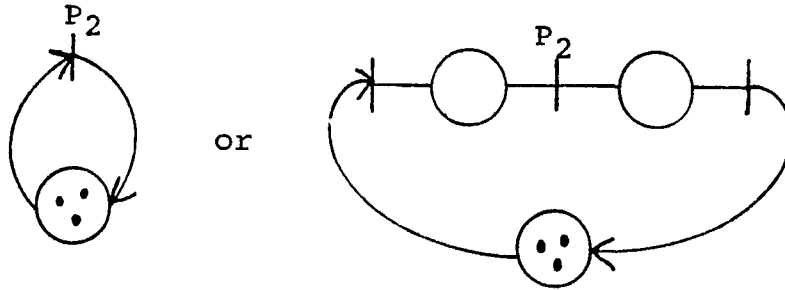
P_1, P_2 PRODUCERS FOR 2 OF P_3, P_4, P_5 ;

7) ALLOW UP TO (n) CONCURRENT COPIES OF processes ;

This command enables the user to allow more than one copy of a task to execute concurrently. (The task executing concurrently might be compiled as a pure procedure.)

ALLOW UP TO 3 CONCURRENT COPIES OF P_2 ;

P_2 would then be compiled as either



Note that in either case there can be up to 3 copies of P firing (executing) concurrently.

8) <processes> IN SEQUENCE;

This is compiled as a number of PRECEDE commands.

P_1, P_2, P_3 IN SEQUENCE

is treated as

P_1 PRECEDES P_2 ;

P_2 PRECEDES P_3 .

We will also provide the user with a facility for defining MACROS and/or SUBROUTINES. As an example of a MACRO consider:

PIPELINE P_1, P_2, P_3 ;

which translates into:

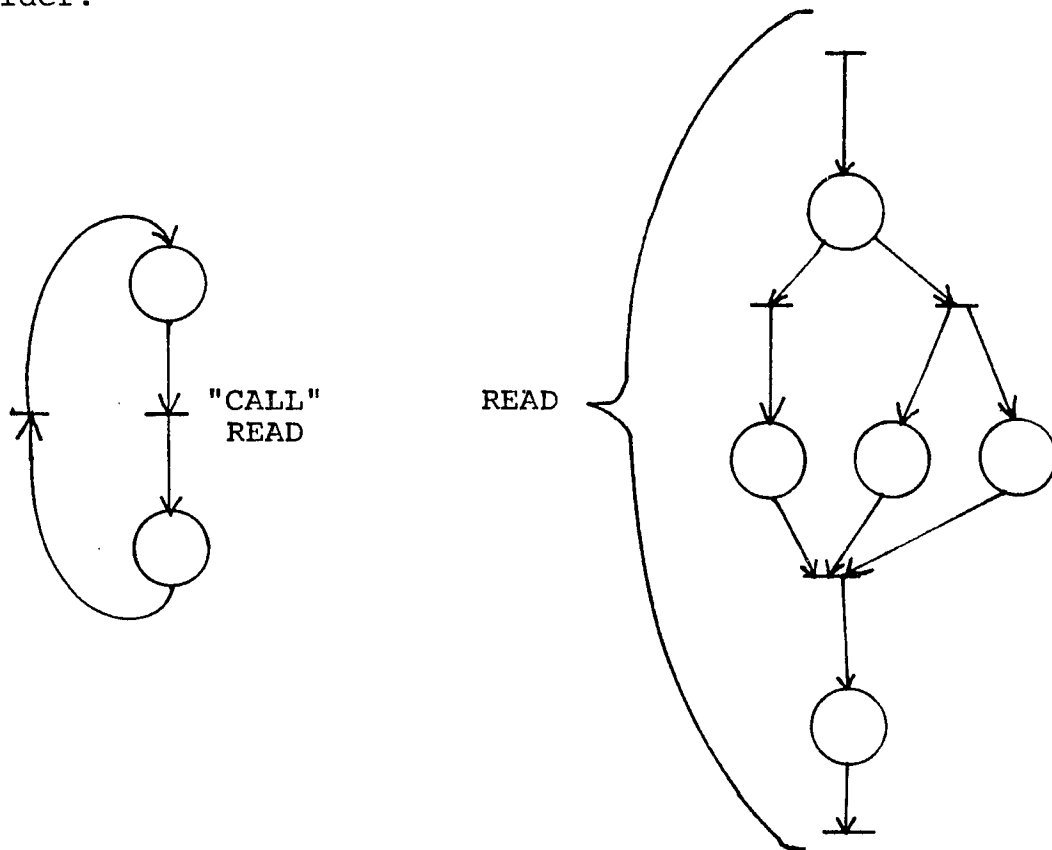
P_1 PRODUCER FOR CONSUMER P_2 ;

P_2 PRODUCER FOR CONSUMER P_3 .

As a subroutine facility we can allow any transition to represent a Petri Net subroutine if that subroutine obeys the requirement that it has a single entry transition and a

single exit transition.

Consider:



This feature may be useful in reducing the size of the coverability trees that may need to be searched.

4.4 The Control Module

The control module can be thought of as working similarly to the LISP APPLY function. Given a set of system modules and the Petri Net generated from the user's design statements, it will execute the system modules in accordance with the constraints of the user's design (i.e., the Petri Net).

We make use of the PL/1 preprocessing facilities to generate the control program. We construct a control pro-

gram as a conditional MACRO which reconfigures itself to run according to the Petri Net which is input to the compiler. Each transition firing of the net corresponds to either a call to a system module or a "dummy" firing.

The following example does not use the design language to generate a Petri Net. Instead, we have designed the Petri Net ourselves. Nonetheless, the example will be useful in illustrating how the system control module will function.

We consider four processes T1, T2, T3, and T4 (Figure 4-1). T1 will produce a datum. T2 will store the datum in the next available slot of a circular queue (buffer). T3 will retrieve the next datum from the queue. T4 will consume (i.e., process) the datum.

```

T1:PROCEDURE;
  /* PRODUCER */
  DECLARE
    (COUNT,TEMP) FIXED BIN (15) EXTERNAL INIT (0);
    K=SUBSTR(TIME,7,1) * 10;
    TEMP=COUNT;
    COUNT=COUNT+1;
    IF COUNT = 40 THEN STOP;
    PUT SKIP LIST('T1 DELAYS ',K);
    DELAY(K);
    RETURN;
  END T1;

T2:PROCEDURE;
  DECLARE
    (TEMP,BUFFER(4),BACKQ INIT (1)) FIXED BIN (15)
    EXTERNAL,
    BUFSIZE FIXED BIN (15) INIT (4) EXTERNAL,
    ;
  BUFFER (BACKQ)=TEMP;
  BACKQ=BACKQ+1;
  IF BACKQ BUFSIZE THEN BACKQ=1;
  RETURN;
  END T2;

T3:PROCEDURE;
  DECLARE
    (TEMP2,BUFFER(4)) FIXED BIN (15) EXTERNAL,
    FRONTQ FIXED BIN (15) EXTERNAL INIT (1),
    BUFSIZE FIXED BIN (15) EXTERNAL INIT (4);
  TEMP2=BUFFER(FRONTQ);
  FRONTQ=FRONTQ+1;
  IF FRONTQ BUFSIZE THEN FRONTQ=1;
  RETURN;
  END T3;

T4:PROCEDURE;
  /* CONSUMER */
  DECLARE
    (TEMP2) FIXED BIN (15) EXTERNAL;
    K2=SUBSTR(TIME,6,2);
    PUT SKIP LIST('T4 DELAYS ',K2,'CONSUMES ',TEMP2);
    DELAY(K2);
    RETURN;
  END T4;
END NET;

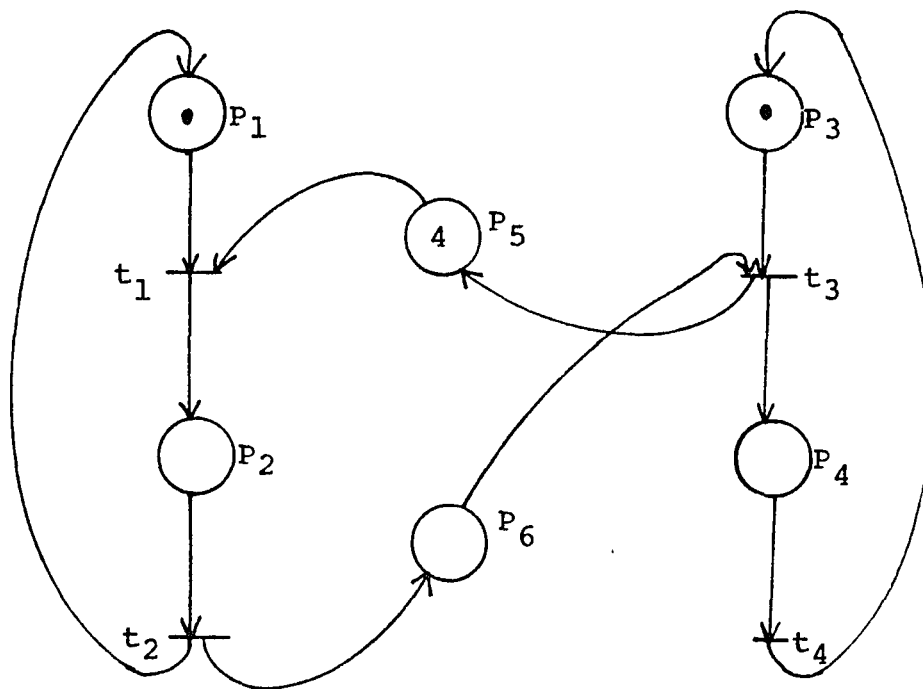
```

Figure 4-1

PL/1 Procedures T_1 , T_2 , T_3 and T_4 .

The Petri Net we design establishes a producer-consumer relationship between T1 and T4. The four tokens initially placed in place P_5 will ensure that no more than four data items can be held in the circular queue. Note that this is not tested for in processes T2 or T3.

The Petri Net follows:



The output of the control module includes the following (Figure 4-2):

Lines 1-7 show the input to the control module. This includes

- the number of places (6)
- the initial marking (1 0 1 0 4 0)
- the number of transitions (4) and their names

(T1, T2, T3, T4).

- the description of the Petri Net.

The rest of the page is output. The first line of output:

```
1 103909330 0 0 1 0 4 0 FIRING
```

is produced by the control module and indicates that transition 1 started firing at time 103909330, resulting in marking 0 0 1 0 4 0.

The next line of output:

```
T1 DELAYS 30
```

is produced by process T1 which gives the amount of time (30) that it will delay to simulate its production time. (Recall that T1 is the producer of data.)

Lower down is another type of output line:

```
T4 DELAYS 94 CONSUMES 0
```

This is produced by process T4 and gives the amount of time (94) that it will delay to simulate "consuming." It also prints the datum consumed (0). If the synchronization is correct between T1, the producer, and T4, the consumer, the datum will be consumed in order, i.e., 0, 1, 2, 3, ...

6 PLACES WITH INITIAL MARKING						1	0	1	0	4	0
4 TRANSITIONS T1,T2,T3,T4											
F						B					
-1	0	0	0	0	0	0	1	0	0	0	0
0	-1	0	0	-1	0	1	0	0	0	0	1
0	0	-1	0	0	-1	0	0	0	1	1	0
0	0	0	-1	0	0	0	0	1	0	0	0
1 103909330						0	0	1	0	4	0
T1 DELAYS	30										FIRING
	1	103909390	0	1	1	0	4	0			COMPLETED
	2	103909400	0	0	1	0	3	0			FIRING
	2	103909410	1	0	1	0	3	1			COMPLETED
	1	103909410	0	0	1	0	3	1			FIRING
	3	103909430	0	0	0	0	3	0			FIRING
T1 DELAYS	40										
	3	103909440	0	0	0	1	4	0			COMPLETED
	4	103909450	0	0	0	0	4	0			FIRING
T4 DELAYS	94					CONSUMES					0
	1	103909480	0	1	0	0	4	0			COMPLETED
	2	103909490	0	0	0	0	3	0			FIRING
	2	103909500	1	0	0	0	3	1			COMPLETED
	1	103909500	0	0	0	0	3	1			FIRING
T1 DELAYS	50										
	4	103909560	0	0	1	0	3	1			COMPLETED
	3	103909560	0	0	0	0	3	0			FIRING
	1	103909580	0	1	0	0	3	0			COMPLETED
	2	103909580	0	0	0	0	2	0			FIRING
	3	103909590	0	0	0	1	3	0			COMPLETED
	4	103909600	0	0	0	0	3	0			FIRING
	2	103909610	1	0	0	0	3	1			COMPLETED
	1	103909610	0	0	0	0	3	1			FIRING
T4 DELAYS	96					CONSUMES					1
T1 DELAYS	60										
	1	103909690	0	1	0	0	3	1			COMPLETED
	2	103909700	0	0	0	0	2	1			FIRING
	2	103909710	1	0	0	0	2	2			COMPLETED
	1	103909710	0	0	0	0	2	2			FIRING
	4	103909730	0	0	1	0	2	2			COMPLETED
	3	103909730	0	0	0	0	2	1			FIRING
T1 DELAYS	70										
	3	103909750	0	0	0	1	3	1			COMPLETED
	4	103909750	0	0	0	0	3	1			FIRING
T4 DELAYS	97					CONSUMES					2

Output of Control Module

Figure 4-2

T₁ T₂ T₃ T₄



The program also produces a time graph (Figure 4-3) which shows which processes are firing (i.e., executing) at discrete times. The top row labels the transitions. Time increases as one goes down the page.

Time Graph Output By the Control
Module

Figure 4-3

4.5 The Query System

The query system of SYSDACS allows the user to interrogate his design in order to determine information about concurrency, demands, etc. The query system is one of the most important tools available to the designer. In designing a sophisticated concurrency scheme there can be many inherent concurrencies, or lack of them, that are not obvious from the initial design. Using the query system the designer can determine if there are any conditions arising unexpectedly from his design. He thus has the information necessary to modify his design accordingly.

For a given design program with processes $P_1, P_2, \dots, P_n$ we consider a process to be 1 if it is executing or 0 if it is not. The user can determine for any subset of processes if a particular combination of executing and nonexecuting processes may exist. This is essentially done by determining the value of an expression we may think of as a boolean expression. For example, the user can ask if at any time processes P_1 and P_3 are executing while P_2 is not. This query could take the form of $P_1 \wedge \neg P_2 \wedge P_3$?

To determine the truth or falsity of a given boolean expression involving processes the query system will construct the concurrency set for the Petri Net compiled from the design program. Clearly in general we cannot expect to construct the entire concurrency set. We intend to develop heuristics for use in searching such a tree in order to discover if a concurrency is possible or not.

Other uses of the query system include:

- Determining if a process is dead, i.e., never fireable. This is determined for process p by the simple reasoning that if $\neg p$ then p is never fireable.
- Determining if some process p ever dies.
- Determining possible bottlenecks or heavy demand.

We note those processes which can be engaged in multiple firings or those queues (places) that can reach omega.

- Determining if the reachability set is finite.

Note that the designer can determine why the reachability set is not finite, if it is not, and then take actions to force a finite reachability set.

4.6 Examples

Example 1

As an example consider the following:

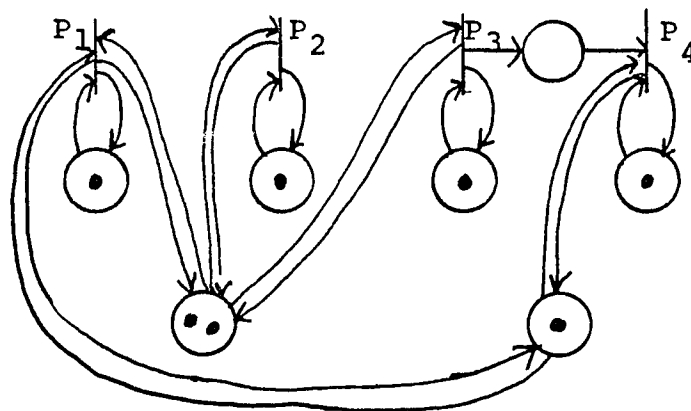
PROCESSES P_1, P_2, P_3, P_4 ;

P_3 PRECEDES P_4 ;

AT MOST 2 OF P_1, P_2, P_3 AT ANY TIME;

MUTUAL EXCLUSION AMONG P_1, P_4 ;

The corresponding Petri Net would be:



Example 2

As another example we assume a user wants processes P_1, P_2, P_3, P_4 , and P_5 to operate in the following manner:

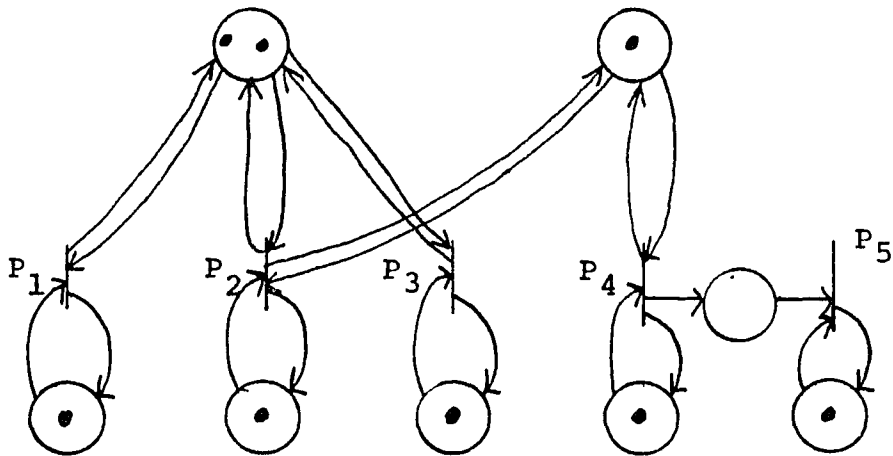
- i) among processes P_1, P_2 , and P_3 only two can be executing at any time;
- ii) if P_2 is executing then P_4 cannot be executing and vice versa;
- iii) P_5 can only execute after P_4 has executed, leaving the result queued for P_5 .

PROCESSES P_1, P_2, P_3, P_4, P_5 ;

AT MOST 2 OF P_1, P_2, P_3 AT ANY TIME;

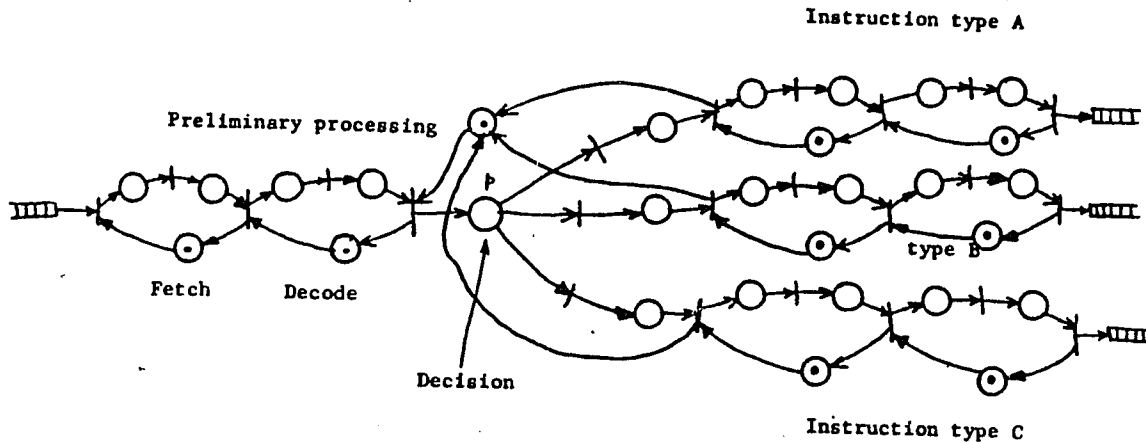
MUTUAL EXCLUSION AMONG P_2, P_4 ;

P_4 PRECEDES P_5 ;



Example 3

As our last example we consider the pipeline processor as given in Ramchandani's [27] work. His original Petri Net design follows:



This we describe as follows:

PROCESSES fetch, decode, decide, type A, type B, type C,
type C' ;

fetch PRODUCER FOR CONSUMER decode;

decode PRODUCER FOR CONSUMER decide;

decide PRODUCER FOR 1 OF CONSUMERS type A, type B, type C;

type C PRODUCER FOR CONSUMER type C' ;

The Petri Net is given in Figure 4-4 on the following page.

Note that in our description we added a Process C' which
is in a producer-consumer relationship with C.

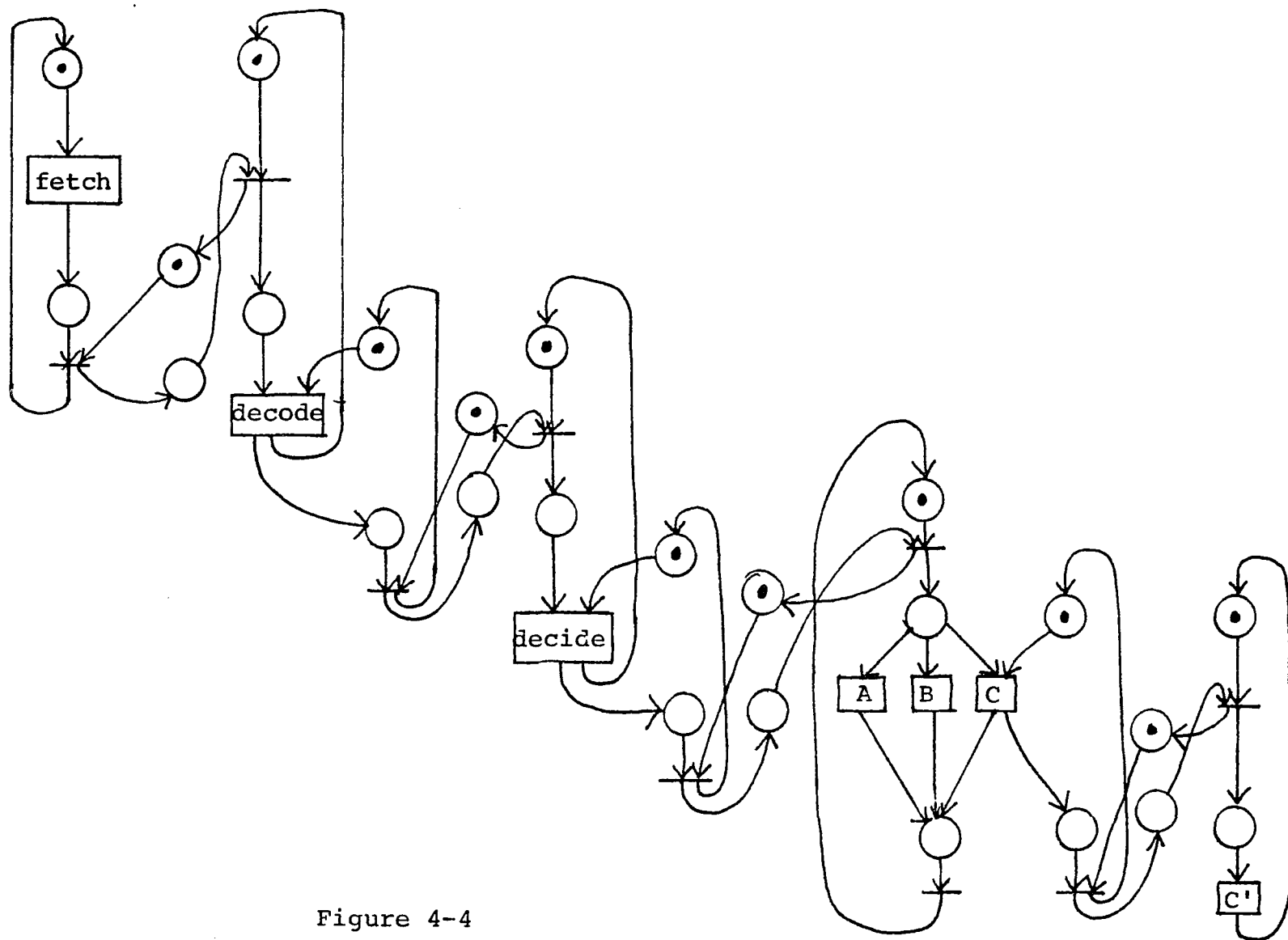


Figure 4-4

CHAPTER 5

A SURVEY OF PETRI NETS AND THEIR APPLICATIONS

5.1 History

In 1962 Carl Adam Petri [26] presented his thesis which was concerned with the conceptual foundation of a theory of communication. "It shall be the task of this theory to describe in a consistent and exact manner as many as possible of the phenomena that occur in the transmission and transformation of information." Among the results of this work he noted that:

"for complicated organization of processes of any given sort the theory yields a means of representation that with equal rigor and simplicity accomplishes more than the theory of synchronous automata."

Petri proposed using nets (graphs) to describe the asynchronous nature of information flow.

In particular he suggested using a switching element (Figure 5-1) with inputs a , b , and output c , with the property

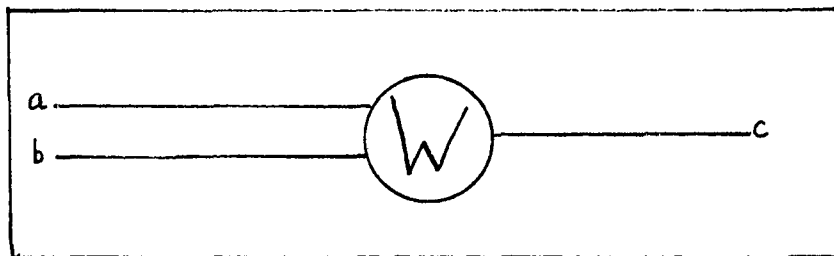


Figure 5-1

"W waits until it observes the object 1a and 1b in its neighborhood and produces from these a new object 1c. This waiting of the switching element W should, however, by no means be thought of as an activity, say of the type such that the element repeatedly interrogates the states of a, b, and c and then on the occasion of some particular result of the interrogation changes the states of a, b, and c; rather it merely represents a connection of a, b, and c that will occasion a state change only if 1a, 1b, and 0c are all present, and the state change consists only in the fact that after its entrance (occurrence) the objects 0a, 0b, and 1c are present.

The process of the substitution of 0a, 0b, 1c for 1a, 1b, 0c we view as a single elementary process that can no longer be divided into separate parts."

He continues, "On the other hand there shall be required of a connection (switching element) W nothing more than the described behavior; we require only that 0a, 0b, 1c be a (temporal) successor of 1a, 1b, 0c."

The work of Petri came to the attention of Anatol Holt [19] who led a group interested in developing methods for the description and analysis of dynamic systems. Holt's group adapted Petri's Nets and developed the Petri Net model as we now know it.

Their intent was to "build a theoretical foundation which will make it possible to ask questions such as:

- A.1 Does the system perform the functions which are expected of it?
- A.2 Might the system "die" or deadlock?
- A.3 Where are system bottlenecks?"

They went on to note that, "Our starting points are the notions of condition, event and their instances. An instance of an event is called an occurrence of that event. An in-

stance of a condition is called a holding of that condition. Out of them we hope to build the concepts with which a wide class of systems can be usefully described, categorized, and analyzed."

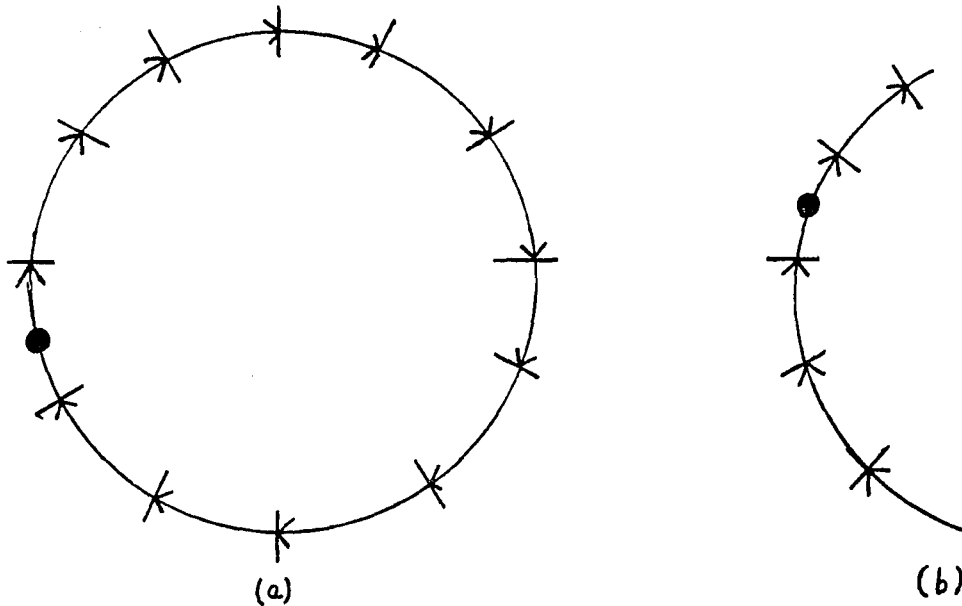
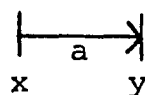


Figure 5-2

Considering Figure 5-2a, "each directed interval $i \longrightarrow i+1$ represents a condition which we can express in English: It is i o'clock. Each dividing mark $\longrightarrow_i$ represents an event, i o'clock. The picture also establishes two relations between conditions and events:



All holdings of condition 'a' begin with occurrences of event x and all holdings of condition 'a' end with occurrences of event y."

A small circle called a token can be used to represent a holding of a particular condition. Figure 5-2(a) represents the holding of the condition: It is 8 o'clock.

By moving the token across an event, we can exhibit the effect of that event. Figure 5-2(b) shows the effect of the event 9 o'clock. An occurrence of 9 o'clock ends the holding of the condition: It is 8 o'clock.

They go on to consider concurrent holdings. "An important aspect of describing complex systems is the ability to represent states of affairs defined by the concurrent holding of many conditions." In Figure 5-3, the occurrence of the event b ends the holding of two conditions. Each begins the holding of two different conditions.

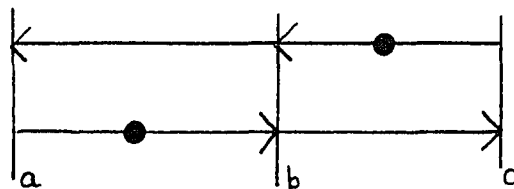


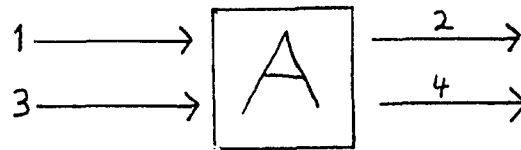
Figure 5-3

The question of deciding for a given net which transitions may become simultaneously enabled is one of interest to us.

Jack B. Dennis, part of Project MAC at MIT, looked at Petri Nets for describing asynchronous control structures of a computer processing unit and for describing concur-

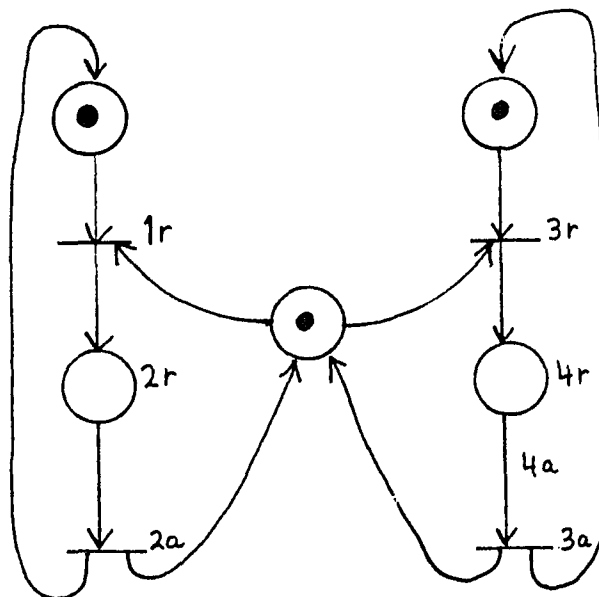
rency in software systems.

For example in [8] he suggests how a Petri Net can model an arbiter module.



"The arbiter causes a ready-acknowledge cycle on link 2 or link 4 for each ready-acknowledge cycle on link 1 or link 3, respectively, subject to the restriction that link 2 and 4 may not be active at the same time."

He uses the arbiter in building a control structure that allocates shared units among competing requests for service. The Petri Net model follows:



Note:

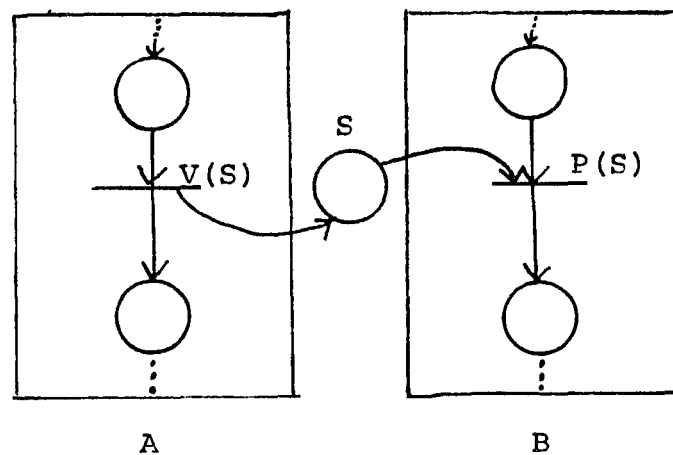
r - ready
a - acknow-
ledge

We will later see another example of how Petri Nets can model a pipeline processor.

In his work he points out how the synchronizing primitives P, V^1 of Dijkstra [9] can be realized in Petri Nets. Consider the following two processes A, B.



This can be represented by a Petri Net as follows:



¹ $P(s)$ behaves in the following manner
 L: If $S > 0$ then $S \leftarrow S - 1$; else go to L;
 $V(s)$ behaves in the following manner: $S \leftarrow S+1$;

In 1968 Karp and Miller [20] published a report on a parallel program schemata. In this report they introduced a mathematical model they called Vector Addition Systems. This was used to study decidability issues about their parallel program schemata.

"An r -dimensional vector addition system is a pair $\mathcal{W} = (d, W)$ in which d is an r -dimensional vector of non-negative integers and W is a finite set of r -dimensional integer vectors." The elements of W we call transitions. For example consider $\mathcal{W} = (d, W)$ where:

$$(i) \quad d = (1, 1, 0, 0)$$

$$(ii) \quad W = \{(-1, -1, 1, 1) (-1, 1, -1, 1) (1, 0, 0, -1)\}$$

. Note how the vector of d describes the initial marking of the Petri Net's places.

Of particular interest from this report of Karp and Miller is an algorithm they present for computing a finite set of vectors (markings) $\mathcal{T}(\mathcal{W})$ such that for any possible marking x achievable by the Vector Addition System (or Petri Net) there is $y \in \mathcal{T}(\mathcal{W})$ such that $y \geq x$. We call $\mathcal{T}(\mathcal{W})$ the coverability tree of $\mathcal{W}$.

According to Hack [15]: "The similarity of these two formalisms had been recognized early, but had not been exploited until about 1972.

In so doing, the results and open problems in each became evident. Of particular interest to us is that the coverability tree procedure can be used with Petri Nets.

We list some of the decidability issues which were raised. Given Petri Nets P_1 and P_2 can we decide if:

- (i) $R(P_1) \subseteq R(P_2)$? This was shown to be unsolvable by M. Baker [3] in 1972.
- (ii) $R(P_1) = R(P_2)$? This was shown to be unsolvable by M. Hack [16] in 1974.
- (iii) $x \in R(P_1)$ for a given x ?
- (iv) $R(P)$ is semilinear for any Petri Net P ? It was shown by Rabin [3] that there exists a Petri Net P such that $R(P)$ is not semilinear.

From Karp and Miller's [20] Coverability Tree procedure we can answer the following questions about Petri Net P .

- (1) For a given x does there exist $y \in R(P)$ such that $x \leq y$?
- (2) For all $y \in R(P)$ is $(y)_i \leq n$ for some $n \in \mathbb{N}$?
- (3) For all $y \in R(P)$ is a given set of elements of y simultaneously bounded?
- (4) Is $R(P)$ a finite set?

5.2 Computability

Before we present one of the important decision problems in Petri Net Theory, we define linear and semilinear sets. A set of k -tuples is linear iff there exists $I \in N^k$ and a set of k -tuples $P \subseteq N^k$ such that S consists of all k -tuples of the form

$$I + n_1 P_1 + n_2 P_2 + \dots + n_j P_j, n_i \in N, P_i \in P, 1 \leq i \leq j.$$

Then S can be denoted by $L(I;P)$.

A set of k -tuples is semilinear iff S is the finite union of linear sets of the same dimension. For the set

$$J \in N^k, J = \{J_1, J_2, J_3, \dots, J_t\} \text{ we define}$$

$$L(J;P) = L(J_1;P) \cup L(J_2;P) \cup \dots \cup L(J_t;P).$$

Lemma 5.2.1 The uniform membership problem for semilinear sets is decidable.

$$\text{For } A, B \in N^k \quad \underline{A \leq B} \text{ iff } a_i \leq b_i, 1 \leq i \leq k.$$

$$\underline{A < B} \text{ iff } A \leq B \text{ and for some } j, a_j \neq b_j.$$

If $A \not\leq B$ and $B \not\leq A$ then A and B are incomparable.

Lemma 5.2.2 Any set of mutually incomparable elements in N^k is finite.

Proof: Consider that every infinite sequence of vectors in N^k contains an infinite non-decreasing subsequence. In a set of mutually incomparable vectors there can be no such subsequence. Hence, the set must be finite.

It had been conjectured by Miller [20] that the reachability sets of VAS's were strictly semilinear. However, Rabin [3] showed that this was not true by constructing a VAS for which the reachability set was not semilinear. We

present Rabin's proof that there exists a reachability set which is not semilinear. We will then present Rabin's proof of the undecidability of the Reachability Set Inclusion Problem as done in Hack[15].

We first consider counter machines (CM's) which have also been called program machines. The CM's are composed of a finite number of registers or counters each of which has an infinite capacity. Programs for CM's consist of the following five types of instructions:

- 1) $c_i \leftarrow c_i + 1$; where c_i is register or counter i
- 2) $c_i \leftarrow c_i - 1$;
- 3) if $c_i = 0$ then goto a_1 ; where a_1 is the label of an instruction
- 4) goto a_1 ;
- 5) stop;

By convention, the CM program is always started on the first instruction called the initial instruction.

Here we concern ourselves with restricted non-deterministic counter machines (RNCM's). RNCM's differ from CM's in that the former:

- i) cannot test for zero (restricted) and
- ii) has a multiway branch instruction (non-deterministic).

The instructions for a RNCM consist of:

- 1) $c_i \leftarrow c_i + 1$;
- 2) $c_i \leftarrow c_i - 1$;
- 3) goto $a_1, a_2, \dots, a_t$;
- 4) stop;

Figure 5-4

The multiway branch instruction is similar to a "fork" instruction in that with the execution of this instruction, the computation "forks" into t paths. Each of these paths become independent computations from this point on. This instruction is sometimes interpreted as splitting the program into t copies where in copy j , goto $\bar{a}_j$.

A simulation of an RNCM will produce a (possibly infinite) tree of instructions which were executed, the root of the tree being the initial instruction. Any finite path of this tree from the root is called a computation. All computations fall into one of three categories:

- i) a computation is accepting if its path ends by stopping.
- ii) a computation is failing if an instruction in its path 'drives' a counter negative.
- iii) a computation is incomplete if it's not accepting or failing.

As an example of an RNCM, consider the following program which copies a number from c_1 to c_2 :

```
test:  goto loop,end;
loop:   $c_1 \leftarrow c_1 - 1$ ;
        $c_2 \leftarrow c_2 + 1$ ;
       goto test;
end:   stop;
```

We now consider what it means for a function to be weakly computable.

A counter configuration of an m -counter machine is an

m -tuple $x=(c_1, c_2, \dots, c_m)$. A function $f:N^n \longrightarrow N$ is said to be weakly computable on an m -counter ($m \geq n$) RNCM F if for every non-negative integer $k \leq f(x_1, x_2, \dots, x_n)$ there exists a computation by F which leads from the counter configuration $X=(x_1, x_2, \dots, x_n, 0, \dots, 0)$ to $U=(u_1, u_2, \dots, u_n, k, u_{n+2}, \dots, u_m)$ and conversely a computation from X to U implies that $k \leq f(x_1, x_2, \dots, x_n)$.

We now give an RNCM which weakly computes x^2 . The following RNCM actually computes multiplication.

$$\underline{c_1 \times c_2 \longrightarrow c_4}$$

- 1) loopback: goto end, decx, *+1;
- 2) $c_3 \longleftarrow c_3 - 1;$
- 3) $c_2 \longleftarrow c_2 + 1;$
- 4) goto loopback; } pump c_3 back into c_2
- 5) decx: $c_1 \longleftarrow c_1 - 1;$ c_1 controls the number of iterations of loopforward
- 6) loopfor: goto loopback, *+1;
- 7) $c_2 \longleftarrow c_2 - 1;$
- 8) $c_3 \longleftarrow c_3 + 1;$
- 9) $c_4 \longleftarrow c_4 + 1;$ } pump c_2 into c_3 and c_4
- 10) goto loopfor;
- 11) end: stop;

Figure 5-5

The maximum in c_4 is reached as follows:

- 1) c_2 is transferred to c_3 and c_4 .
- 2) c_3 is transferred back to c_2 .

3) steps (1) and (2) are repeated c_1 times.

Note that c_4 is non-decreasing.

We will now demonstrate how a VAS can completely mimic the calculation of an RNCM.

Theorem 5.1: Let C be an RNCM with m counters and p instructions. Then there exists a VAS $\mathcal{C} = (I, W)$, $I \in N^{m+p}$, $W \in N^{m+p}$ such that for every input $\bar{k} \in N^m$ to $\mathcal{C}$ the reachability set of $\mathcal{C}$ with $I = (k_1, k_2, \dots, k_m, 1, 0, \dots, 0)$ and W described below is $\{ (b_1, b_2, \dots, b_m, -, \dots, -) \in N^{m+p} \mid (b_1, \dots, b_m) \text{ is a counter configuration of } \mathcal{C} \}$.

Proof: We interpret the $m+p$ -tuple vectors as representing the contents of the m counters in the first m components and use the remaining p components as a finite state control. Each of the components $m+1$ through $m+p$ is one-to-one with the p instructions in the RNCM program. By placing a 1 in one of the components, we enable the instruction corresponding to that component. This vector then actually represents an instantaneous description in the computation of a function.

We proceed to the description of W . For each instruction $c_i \leftarrow c_i + 1; (c_i \leftarrow c_i - 1;)$ it's clear that there will be one corresponding vector in W with $+1$ (-1) in its i^{th} component, 0 in the other $m-1$ components. If we let this instruction be the j^{th} instruction in the program, then there will be a -1 in its $m+j^{\text{th}}$ component and $+1$ in the $m+(j+1)$ component. The former testing of the j^{th} instruction is enabled, the latter enabling the $j+1^{\text{st}}$ instruction. So, if we have as

the j^{th} instruction $j) c_i \leftarrow c_i + 1$; we have in W the following $m+p$ -tuple $(0, 0, \dots, 0, 1, 0, \dots, 0, \overset{j}{-1}, 1, 0, \dots, 0)$. Notice that this vector will only be "enabled" by the instantaneous description $(x_1, x_2, \dots, x_m, 0, \dots, 0, \overset{j}{1}, 0, \dots, 0)$.

If the j^{th} instruction in the program is stop; then the corresponding vector in W would be $(0, 0, \dots, 0, -1, 0, \dots, 0)$ where the -1 is in the j^{th} location. This would have the effect of disabling all instructions from changing the ID since the "stop vector" zeroes out the finite state control part and all of the other vectors in W contain a -1 in exactly one of the last l components of each vector. Finally, if the j^{th} instruction was goto $a_1, a_2, \dots, a_t$; then we have in W t vectors.

The following VAS computes the product of two numbers and corresponds to the RNCM given in Figure 5-5.

RNCM $c_1 \times c_2 \rightarrow c_4$	W										
	$(c_1 c_2 c_3 c_4$	$I_1 I_2 I_3 I_4$	$I_5 I_6 I_7 I_8$	$I_9 I_{10} I_{11})$							
1) loopback: goto end, decx, *+1;	(0 0 0 0	-1 1 0 0	0 0 0 0	0 0 0)							
	(0 0 0 0	-1 0 0 0	1 0 0 0	0 0 0)							
	(0 0 0 0	-1 0 0 0	0 0 0 0	0 0 1)							
2) $c_3 \leftarrow c_3 - 1$;	(0 0 -1 0	0 -1 1 0	0 0 0 0	0 0 0)							
3) $c_2 \leftarrow c_2 + 1$;	(0 +1 0 0	0 0 -1 1	0 0 0 0	0 0 0)							
4) goto loopback;	(0 0 0 0	1 0 0 -1	0 0 0 0	0 0 0)							
5) decx: $c_1 \leftarrow c_1 - 1$;	(-1 0 0 0	0 0 0 0	-1 1 0 0	0 0 0)							
6) loopfor: goto loop- back, *+1;	(0 0 0 0	1 0 0 0	0 -1 0 0	0 0 0)							
	(0 0 0 0	0 0 0 0	0 -1 1 0	0 0 0)							
7) $c_2 \leftarrow c_2 - 1$;	(0 -1 0 0	0 0 0 0	0 0 -1 1	0 0 0)							
8) $c_3 \leftarrow c_3 + 1$;	(0 0 +1 0	0 0 0 0	0 0 0 -1	1 0 0)							
9) $c_4 \leftarrow c_4 + 1$;	(0 0 0 +1	0 0 0 0	0 0 0 0	-1 1 0)							
10) goto loopfor;	(0 0 0 0	0 0 0 0	0 1 0 0	0 -1 0)							
11) end: stop;	(0 0 0 0	0 0 0 0	0 0 0 0	0 0 -1)							

Figure 5-6

We can now conclude:

Corollary: There exists a VAS $\mathcal{W}$ such that $R(\mathcal{W})$ is not semi-linear.

Proof: Construct the VAS corresponding to the RNCM which weakly computes multiplication.

We now turn our attention to weakly computing functions with Petri Nets.

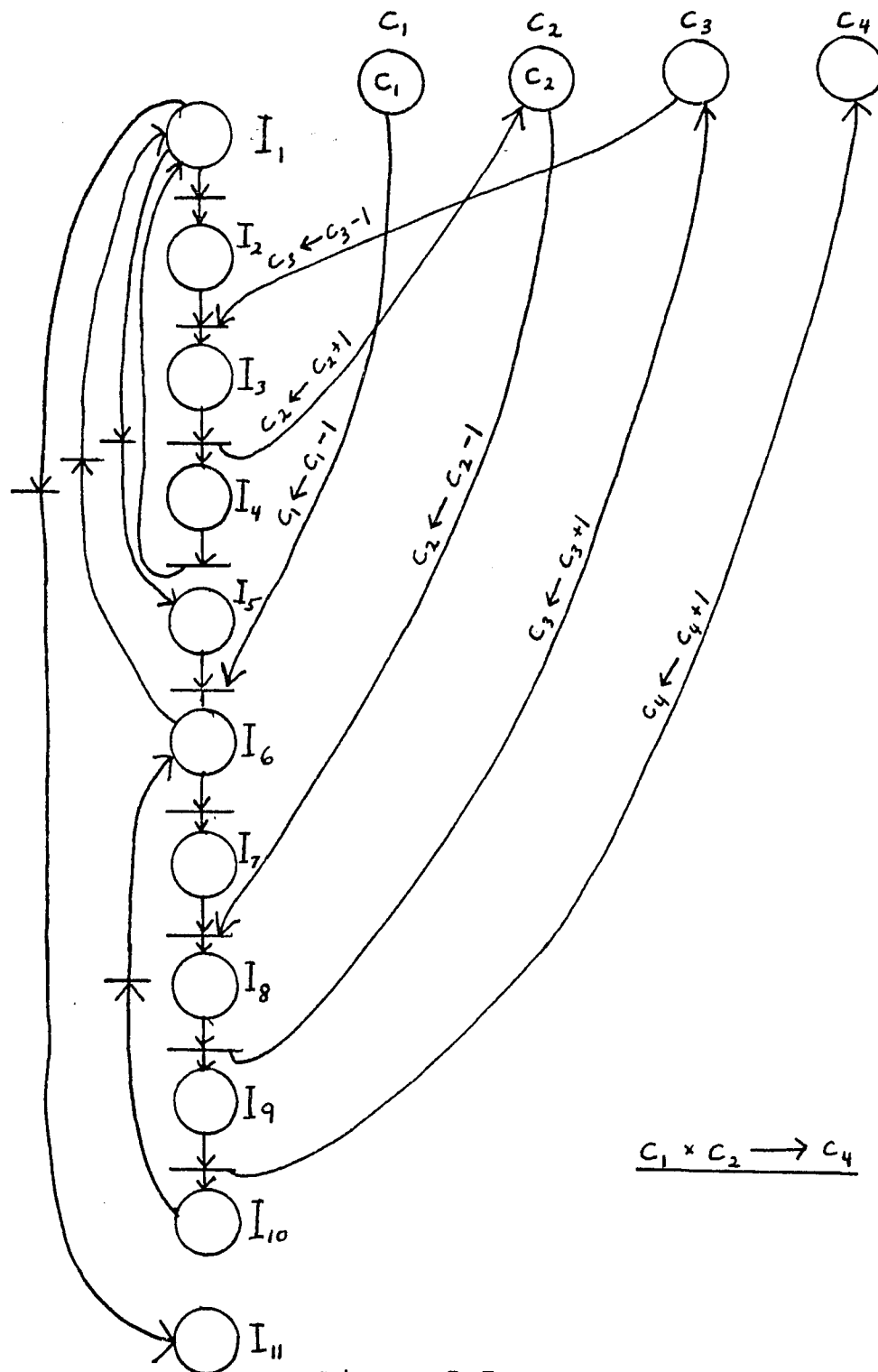


Figure 5-7

The above is the Petri Net corresponding to the RNCM and VAS given in Figure 5-6..

Hack [15] used Petri Nets to demonstrate Rabin's proof of the undecidability of the reachability set inclusion problem for VAS's. We present Hack's proof.

Theorem 5.2: Given two Petri Nets having the same number of places, each with a given initial marking, it is undecidable in general whether every marking reachable in one net is also reachable in the other.

Proof: In this proof we use the undecidability of Hilbert's 10th problem which asks if one can decide whether an arbitrary polynomial with integer coefficients has an integral root.

The following lemma is equivalent¹ to Hilbert's 10th problem.

Lemma 5.2.3: Given two polynomials of r variables and non-negative integer coefficients $P(\bar{x})$ and $Q(\bar{x})$ such that $(P(\bar{x}) \geq Q(\bar{x}))$, it is undecidable if there exists a solution $\bar{x} \in \mathbb{N}^r$ to $P(\bar{x}) = Q(\bar{x})$.

To prove the theorem, we show how we can construct Petri Nets A and B which weakly compute P and Q respectively such that the reachability set of B is included in the reachability set of A except if there exists a marking such that $P(\bar{x}) = Q(\bar{x})$ i.e., $R(B) \subseteq R(A)$ iff $\exists \bar{x} \in \mathbb{N}^r$ such that $P(\bar{x}) = Q(\bar{x})$.

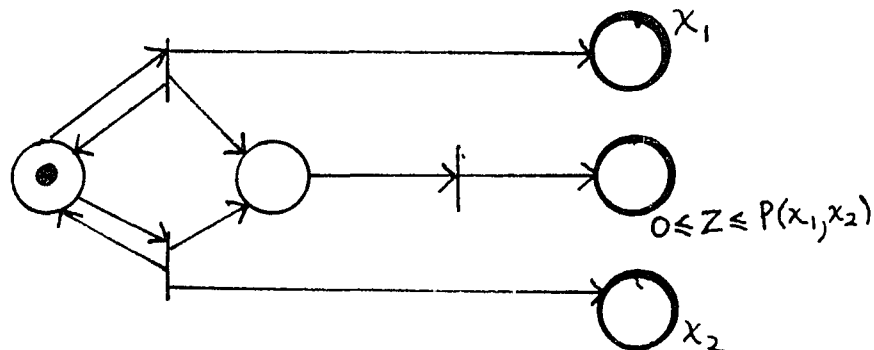
We first prove lemma 5.2.3.

¹Problem A is reducible to B iff solving B solves A .
Problem A is equivalent to B iff A is reducible to B and B is reducible to A .

Proof of lemma 5.2.3: Let $R(\bar{x})$ be an arbitrary polynomial with r variables. Then we can find $R_1(\bar{x})$ such that $R(\bar{x})=0$ has a solution in $\mathbb{Z}^r$ iff $R_1(\bar{x})=0$ has a solution in $\mathbb{N}^r$. Letting $R_2(\bar{x})=(R_1(\bar{x}))^2$ we then have $(\forall \bar{x} \in \mathbb{N}^r) (R_2(\bar{x}) \geq 0)$. We now separate the positive and negative coefficients of R_2 giving us $R_2(\bar{x}) = P(\bar{x}) - Q(\bar{x}) \geq 0$ where P and Q are polynomials with non-negative integer coefficients. Clearly, if it was decidable if there exists $\bar{x} \in \mathbb{N}^r$ such that $P(\bar{x}) = Q(\bar{x})$ we could decide Hilbert's 10th problem. We now show how a Petri Net can weakly compute a polynomial with non-negative integer coefficients.

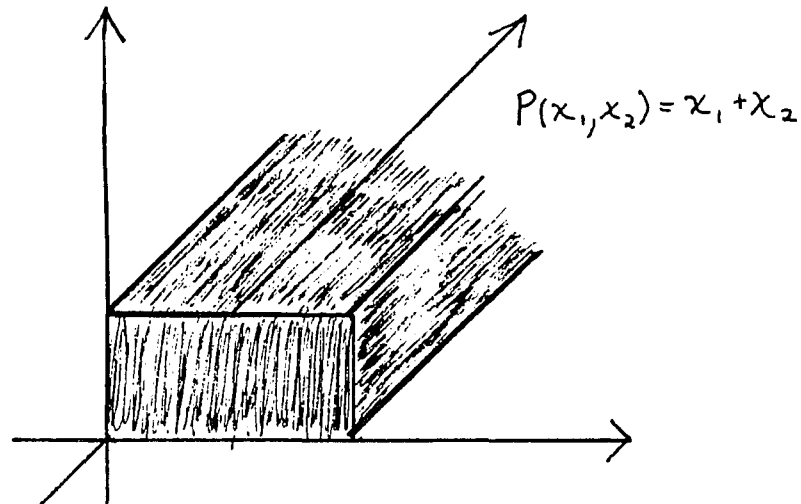
Lemma 5.2.4: Given a polynomial with non-negative integer coefficients of r variables $P(x_1, x_2, \dots, x_r)$ there exists a PN with $r+1$ distinguished places such that the set of reachable markings in these distinguished places is

$\{ \langle x_1, x_2, \dots, x_r, z \rangle \mid x_i \in \mathbb{N} \text{ and } 0 \leq z \leq P(x_1, x_2, \dots, x_r) \}$,
the weak graph of $P(x_1, x_2, \dots, x_r)$. By distinguished places we simply mean some designated subset of places in the net. As an example, consider the following net with distinguished places x_1, x_2, z which weakly computes the polynomial $P(x_1, x_2) = x_1 + x_2$.



The reachable markings of the distinguished places correspond to all the integral points in the rectangular solid

$$P(x_1, x_2) = x_1 + x_2.$$



We can build the polynomial from PN's which weakly compute addition, multiplication, and one which allows us to reproduce a copy of an argument. Given a polynomial P with r variables $x_1, x_2, \dots, x_r$ the net will be configured as shown in Figure 5-8.

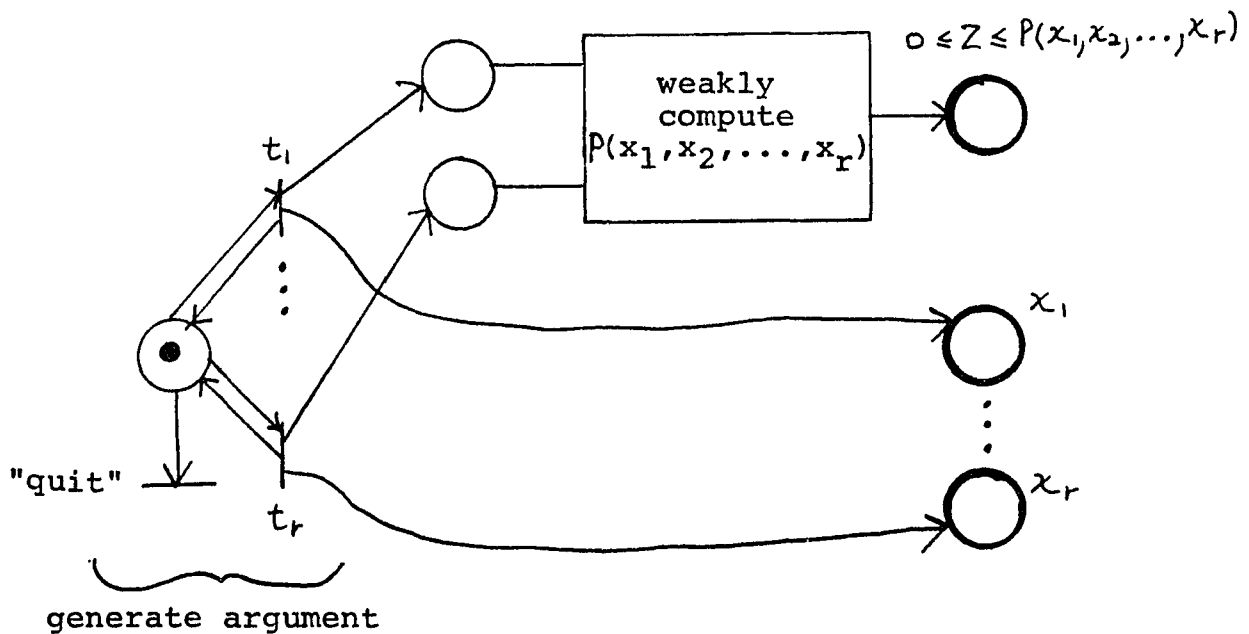
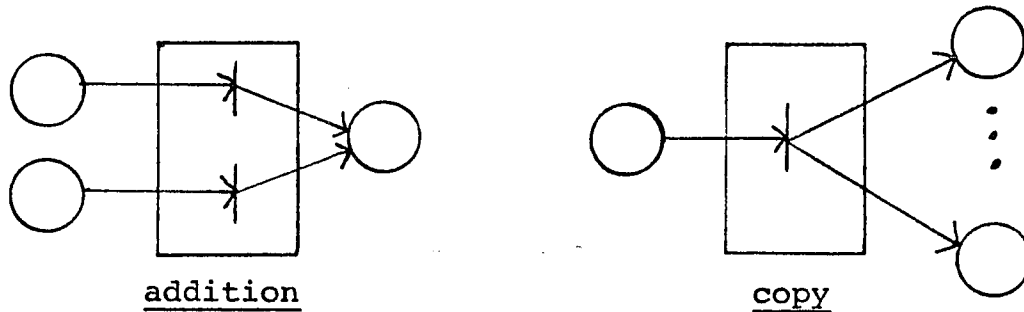


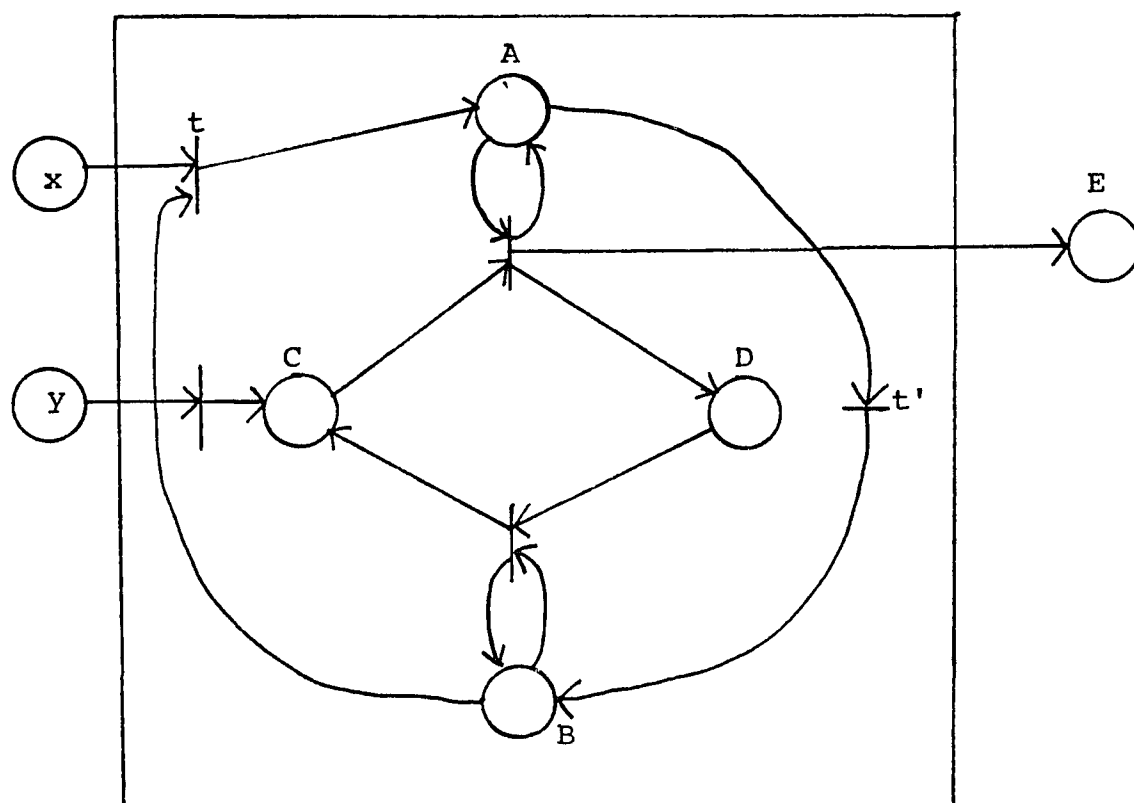
Figure 5-8

It is clear from Figure 5-8 that the firings of transitions t_1 through t_r will produce the arguments $x'_1, \dots, x'_r$ and that the reachable set of markings of distinguished places $x_1, \dots, x_r$ include all arguments. What we need only show now is that the distinguished place z can achieve all markings $0 \leq z \leq P(x_1, x_2, \dots, x_r)$.

Consider the following two nets which weakly compute addition and copy respectively:



In Figure 5-9 we give the PN from Hack which weakly computes multiplication. We could have used the Petri Net derived from the VAS of Figure 5-6 which corresponds to the RNCM of Figure 5-5. This is shown in Figure 5-7.



multiplication

Figure 5-9

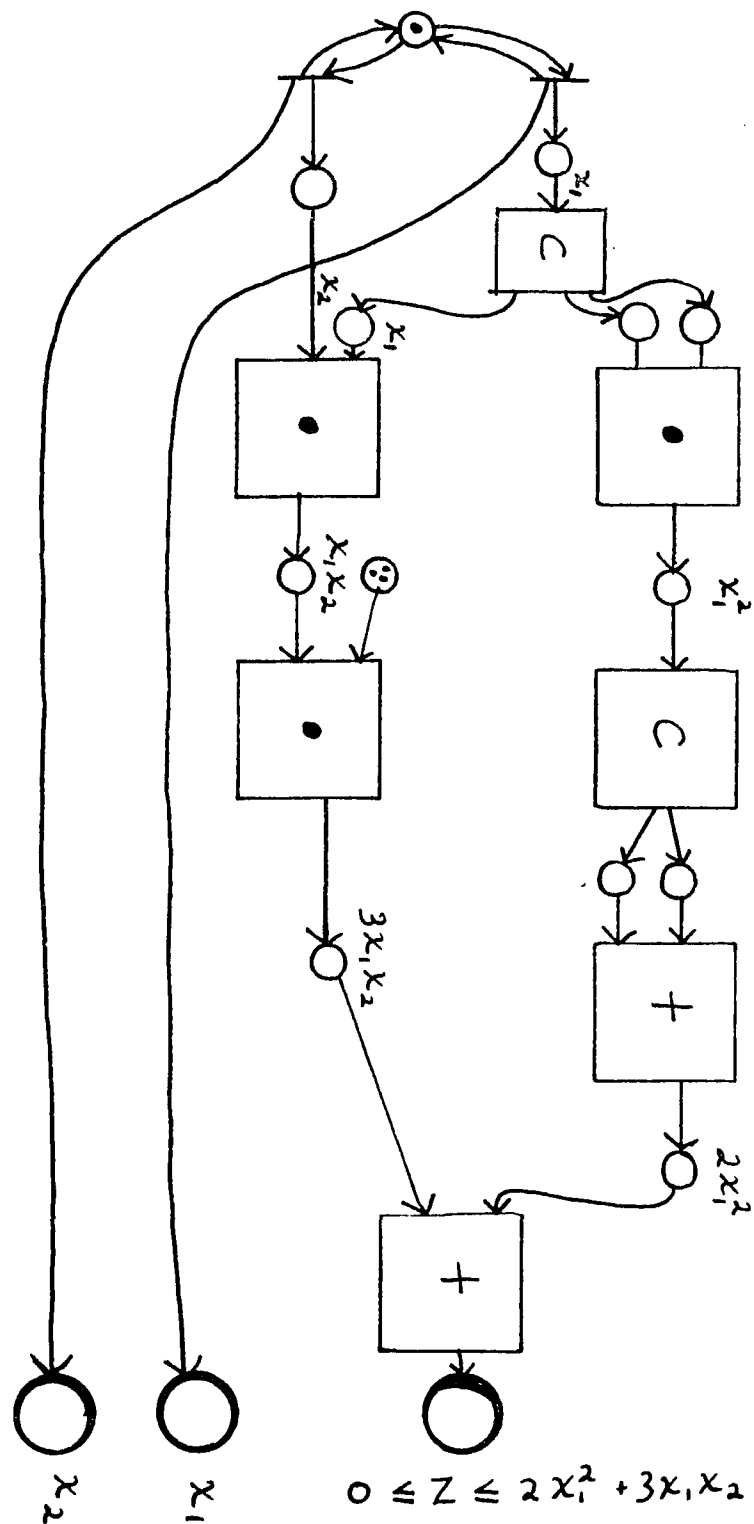
To achieve xy in E the following firing strategy is employed:

- | | x | y | A | B | C | D | E |
|--|-------|-----|-----|-----|-----|-----|------|
| 1) transfer all y tokens into C | x | 0 | 0 | 1 | y | 0 | 0 |
| 2) fire t , then transfer all tokens back to C | $x-1$ | 0 | 1 | 0 | 0 | y | y |
| 3) fire t' to bring y tokens back to C | $x-1$ | 0 | 0 | 1 | y | 0 | y |
| 4) repeat for remaining $x-y$ tokens | 0 | 0 | 0 | 1 | y | 0 | xy |

Remember that we only need to show that xy is achievable.

We now give an example of a Petri Net which weakly computes

$$P(x_1, x_2) = 2x_1^2 + 3x_1x_2.$$



Previously, we mentioned that every marking of the $r+1$ distinguished places of PN B weakly computing $Q+1$ is in the marking class of the $r+1$ distinguished places of PN A except if there exists $\bar{x} \in N^r$ such that $P(\bar{x}) = Q(\bar{x})$. Here we show how to modify A and B to get two nets A' and B' of the same number of places n , such that $R(B') \not\subseteq R(A')$ iff there is a marking of the $r+1$ distinguished places of B that cannot be reached in the $r+1$ distinguished places of A. The following discussion will refer to Figure 5-10.

We make the assumption that B has fewer places than A so we first add k dummy places to B and then to each we add two more places called α and β . Now A' and B' each have n places. We want to show that $R(\text{dist } B) \subseteq R(\text{dist } A)$ iff $R(b') \subseteq R(a')$ and hence cannot decide in general if $R(N) \subseteq R(M)$ for two arbitrary PN's N and M of the same number of places.

In B' we let α and the k dummy places be blank and place a token in β . What we will now show is how we can get A' to match any marking in the P_{r+2} through P_n places of B'. Then all markings reachable in P_{r+2} through P_n of B' are reachable in A'. In A' insert a transition t from α to β and let every other transition in A be in a self-loop with α . Place a single token in α and leave β blank. As long as α has a token the net can operate (i.e. A' behaves as A). But notice that when the token is transferred to β , A is "frozen". In particular, the distinguished places of A are "frozen".

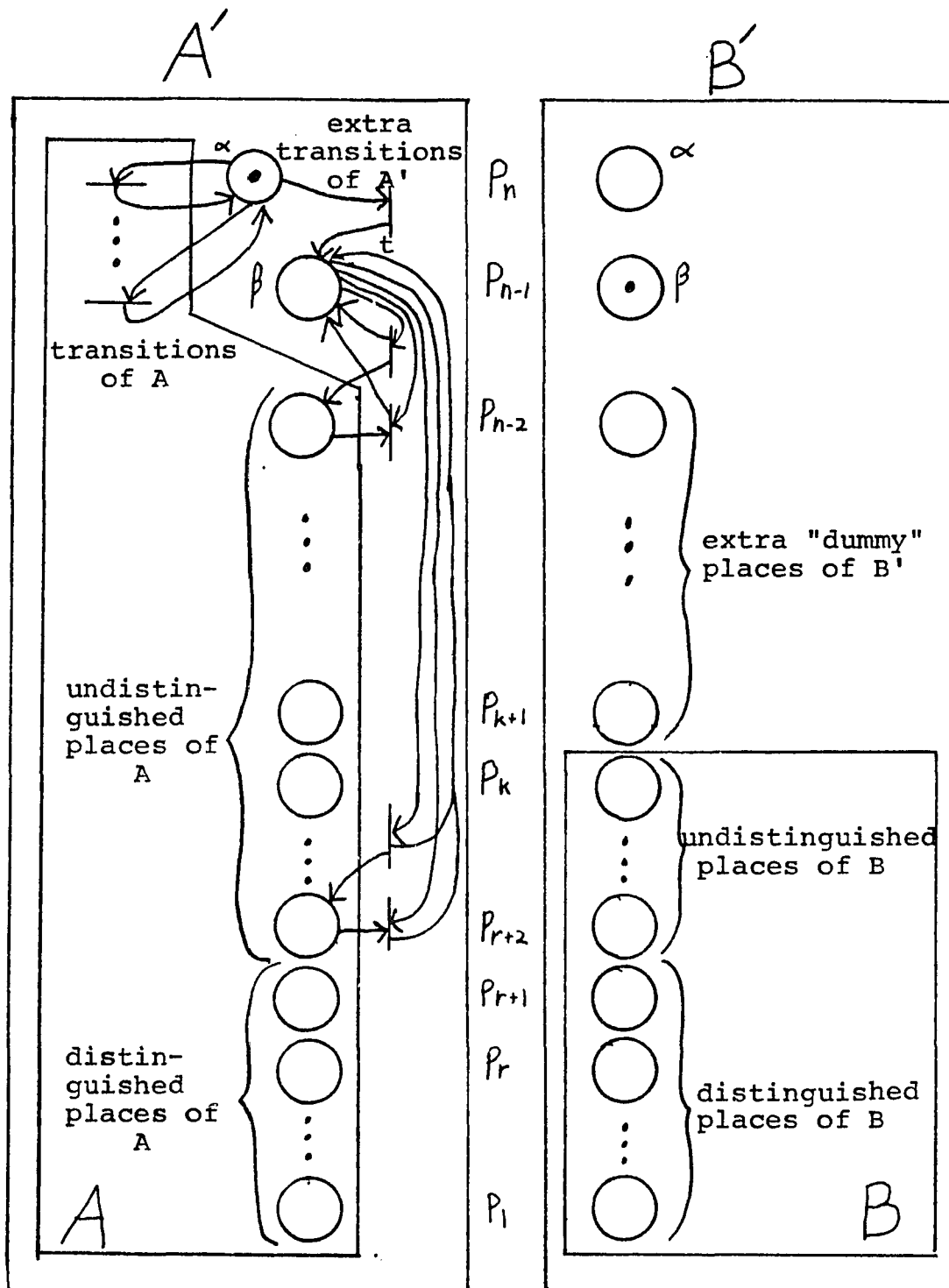


Figure 5-10

Once t fires "freezing" A , we want to show that the undistinguished places of A' , P_{n-2} through P_{r+2} , can achieve any marking and hence match the places P_{n-2} through P_{r+2} of B' . We do this by adding two transitions for each undistinguished place of A , one of the transitions for adding tokens and one for removing tokens to each undistinguished place of A . Notice, too, that after A is frozen, β of A' will have a single token in it matching β in B' .

What we've shown is that a marking of B' is not reachable in A' iff there is a marking in the distinguished places of B not reachable in the distinguished places of A . From the previous lemma we know that if one could decide this, then we could decide Hilbert's 10th problem. QED.

Theorem 5.3: The Inclusion Problem is reducible to the Equality Problem, i.e., if we could solve the equality problem, we could solve the inclusion problem [16].

Proof: We show how given an inclusion problem we can "reduce" it to the equality problem. Let A and B be two n place Petri Nets. We construct Petri Net C as follows: (see Figure 5-11) C will have $n+5$ places. The first n of these will be the places of A and B shared or overlapped. The transitions of A and B are connected to the places exactly as they were in A and B so that if the transitions of B were disabled, C would behave exactly as A , and vice versa. We then make place $P_{n+1}(P_{n+3})$ an input place to all transitions in $A(B)$ and $P_{n+2}(P_{n+4})$ an output place of all transitions in $A(B)$.

We connect these two places with $a''(b'')$ as in Figure 5-11. Notice that if P_{n+1} and P_{n+2} are blank, then no transition in A can fire; similarly, for B with P_{n+3}, P_{n+4} . We now add P_{n+5} as an input place to transition a' and b' . We construct a' so that if it fires, it places a single token in P_{n+2} and places in the n places shared by A and B the initial marking of A; b' is constructed in a similar fashion for B. In P_{n+5} we place a single token; C will then operate as A or B (that is the reachable markings in the n shared places will be the reachability set of A or B), depending on which transition fires first, a' or b' .

For $I \subseteq N^k$, $M \subseteq N^j$ let $L \times M =$

$$\{W \in N^{k+j} \mid n_1, \dots, n_k \in L \text{ and } n_{k+1}, \dots, n_{k+j} \in M\}.$$

Then $R(C) = 0^n \times \{ \langle 0, 0, 0, 0, 1 \rangle \}$

$$\cup R(A) \times \{ \langle 0, 1, 0, 0, 0 \rangle, \langle 1, 0, 0, 0, 0 \rangle \}$$

$$\cup R(B) \times \{ \langle 0, 0, 0, 1, 0 \rangle, \langle 0, 0, 1, 0, 0 \rangle \}.$$

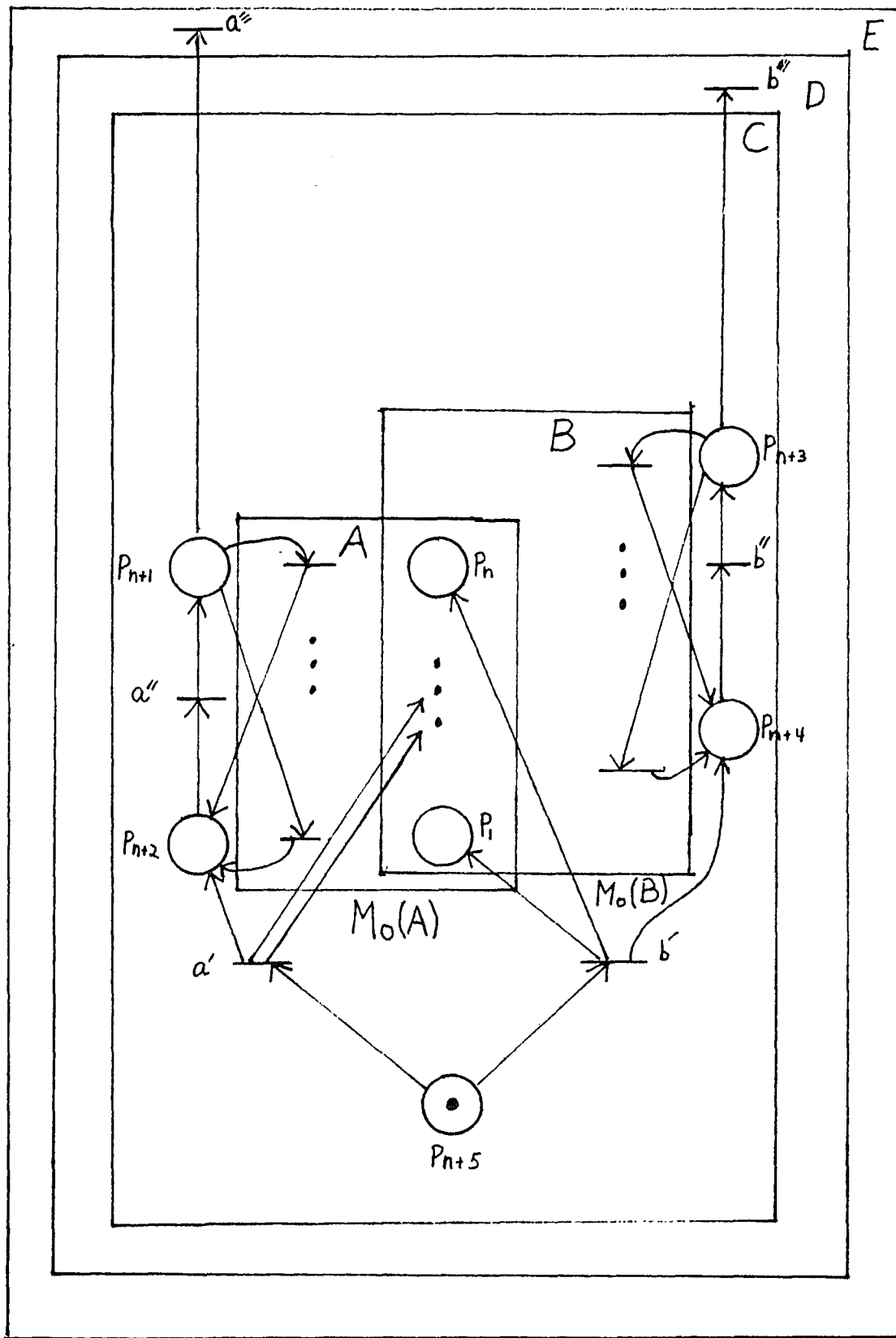


Figure 5-11

Now we construct Petri Net D from C with $n+5$ places by making place P_{n+3} an input place for transition b''' . We construct Petri Net E from C by making place P_{n+1} an input place to transition a''' in addition to P_{n+3} inputting to b''' . Now b''' can only fire if C was behaving as B so

$$R(D) = R(C) \cup R(B) \times \{ \langle 0, 0, 0, 0, 0 \rangle \}$$

In addition to b''' , Petri Net E also has a''' which can fire only if C was behaving as A so

$$\begin{aligned} R(E) &= R(D) \cup R(A) \times \{ \langle 0, 0, 0, 0, 0 \rangle \} \\ &= R(C) \cup R(B) \times \{ \langle 0, 0, 0, 0, 0 \rangle \} \cup R(A) \times \{ \langle 0, 0, 0, 0, 0 \rangle \} \\ &= R(C) \cup (R(A) \cup R(B)) \times \{ \langle 0, 0, 0, 0, 0 \rangle \} \end{aligned}$$

Recalling that

$$\begin{aligned} R(C) &= 0^n \times \{ \langle 0, 0, 0, 0, 0 \rangle \} \cup R(A) \times \{ \langle 0, 1, 0, 0, 0 \rangle, \langle 1, 0, 0, 0, 0 \rangle \} \\ &\quad R(B) \times \{ \langle 0, 0, 0, 1, 0 \rangle, \langle 0, 0, 1, 0, 0 \rangle \} \end{aligned}$$

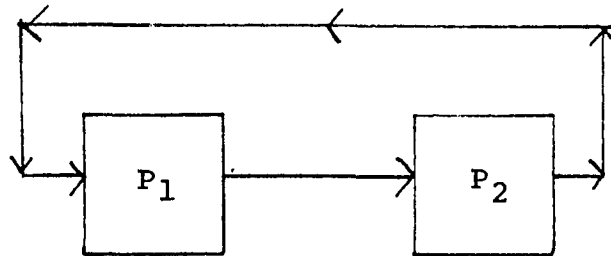
we see that no marking in $R(C)$ can end in $\langle 0, 0, 0, 0, 0 \rangle$.

Hence, $R(D) = R(E)$ iff $R(A) \subseteq R(B)$. But we know that $R(A) \subseteq R(B)$ can't be decided and, therefore, $R(D) = R(E)$ also cannot be decided.

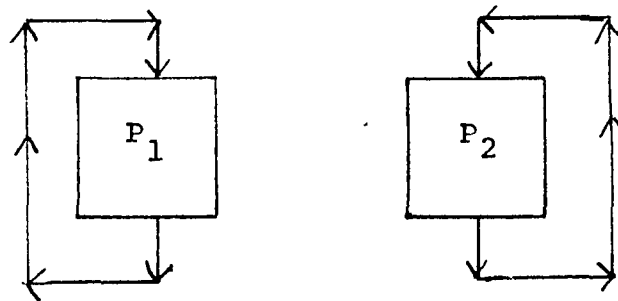
5.3 Process Synchronization

In this section we consider some commonly referenced problems occurring in concurrent systems. We first present a class of problems in which the relationship of the procedures to one another is called a Producer-Consumer relationship.

Consider two procedures P_1 and P_2 . Let P_1 be the "producer" of some datum, and let P_2 be the "consumer" of this datum. Programmed as a sequential procedure, P_1 would compute followed by P_2 , then P_1 again and so on.



Our interest is in examining how we might allow them to execute in parallel.



We first consider what might happen if P_1 and P_2 operate independently without any mechanism for coordination. Assume that P_1 produces $a_1, a_2, a_3, \dots$ so that for i, j , a_i is produced before a_j . We assume P_1 and P_2 share a buffer

which can hold one datum. Whenever P_1 produces a datum, it places it in the buffer. Whenever P_2 is ready for the next datum, it retrieves it from the buffer.

First, consider the situation in which P_1 produces a_2 , and a_3 before P_2 has consumed a_1 . In this case, when P_2 was finished consuming a_1 and went to get the next datum from the buffer, P_1 had already replaced a_2 with a_3 , so that P_2 missed processing a_2 . It is clear that without any coordinating mechanism we can never determine what datum P_2 will process and which will be overlooked.

Another problem arises in the case where P_2 finishes consuming a_1 before P_1 produces a_2 . In this case, P_2 might take a_1 from the buffer and mistakenly "consume" it again. To avoid these problems, we will introduce a simple mechanism whereby process P_1 will signal process P_2 that it has produced a datum and then wait for P_2 to signal that it has retrieved it. Process P_2 will wait for the signal from P_1 before retrieving a datum from the buffer. However, this scheme breaks down under the following circumstances:

- P_1 produces a_1 and signals P_2 to proceed.
- P_2 receiving the signal from P_1 retrieves the datum and signals P_1 that it picked up a_1 .
- P_1 begins to produce a_2 while P_2 begins to consume a_1 .
- Assume that P_1 finishes producing a_2 before P_2 finishes consuming a_1 .
- P_1 signals P_2 that it has produced a new datum and waits for P_2 to signal that it has retrieved it.

However, P_2 was still processing a_1 , and so it missed the signal from P_1 . Meanwhile, P_2 finishes processing a_1 and now waits for P_1 to signal it. But P_1 is stopped and waiting for P_2 to acknowledge its previous signal. This situation is called a dead-lock or deadly embrace.

We can have P_1 constantly signal P_2 until P_2 signals back that it retrieved the datum. But this won't suffice if we want to allow the buffer to hold more than one datum. What is called for is a way for a procedure to remember a signal until it is ready for it.

Dijkstra [9] has introduced what he calls semaphores for this purpose. We can think of a semaphore s as a non-negative integer. Let P and V be indivisible operations on s such that:

$P(s)$ will keep control until s is greater than zero and then decrement s by 1.

$V(s)$ will increment s by 1.

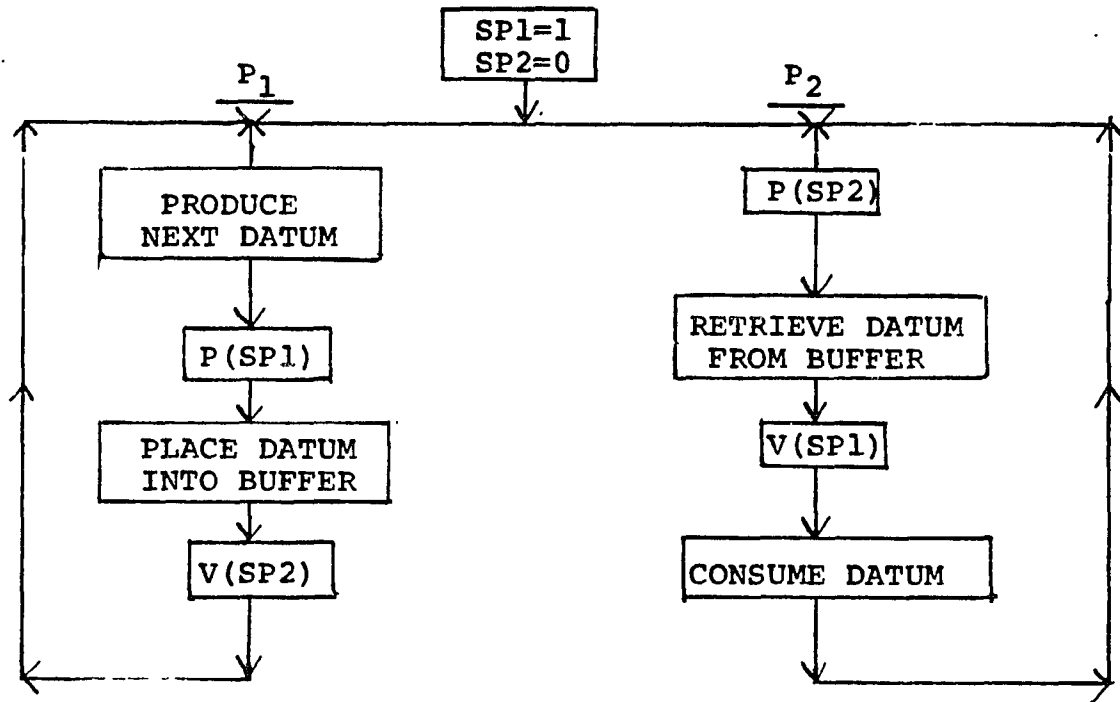
By indivisible we mean that the operation cannot be interrupted, and any words referenced by the operation cannot be read from or written to until the operation is finished. So if a semaphore has the value 1 and two P operations are executed, exactly one of the P operations would decrement the semaphore and proceed.

P and V are semantically equivalent to the following:

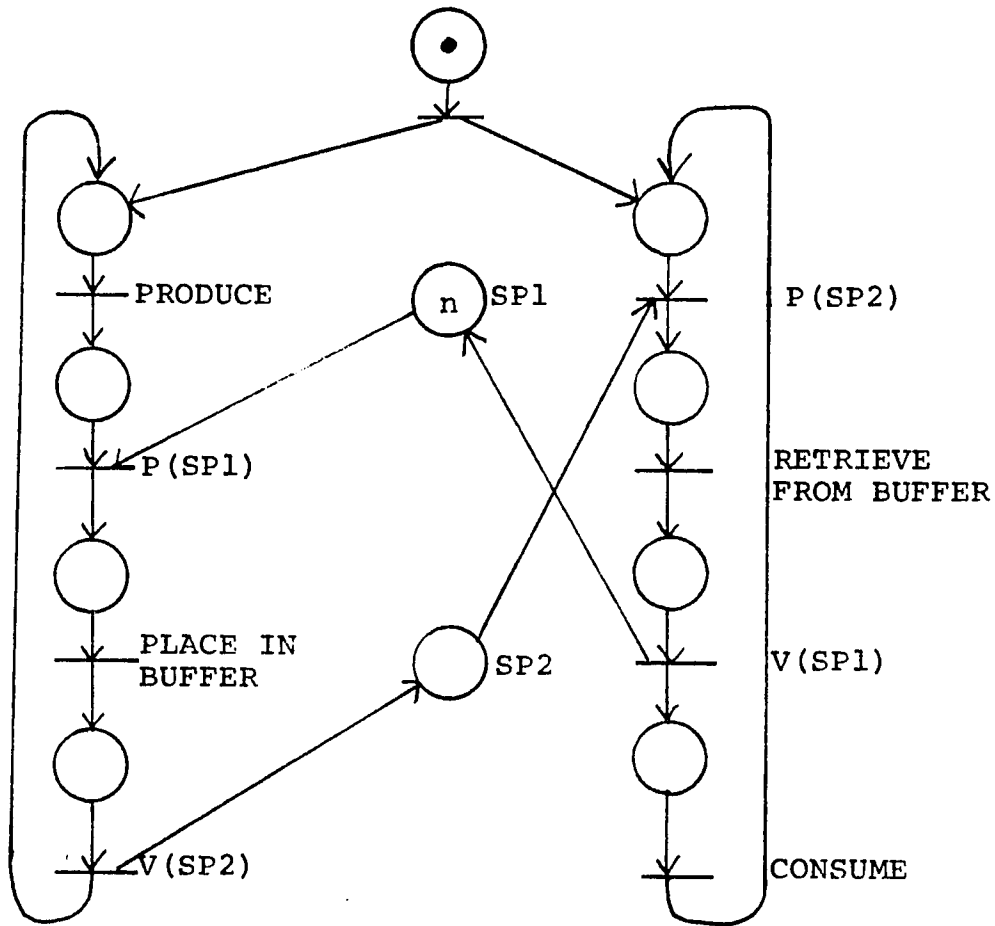
$P(s) - L$: if $s \neq 0$ then go to L ; else $s = s - 1$;

$V(s) - s = s + 1$.

Using the P and V operations and semaphores SP1 and SP2, we can specify the above producer-consumer relationship between procedures P_1 and P_2 as follows:



The following is a Petri Net model of the system previously shown using semaphores:



Notice that we can give an "equivalent" and "reduced" design of the system by collapsing "  " sequences and carrying over any transition operations to the collapsed transition.

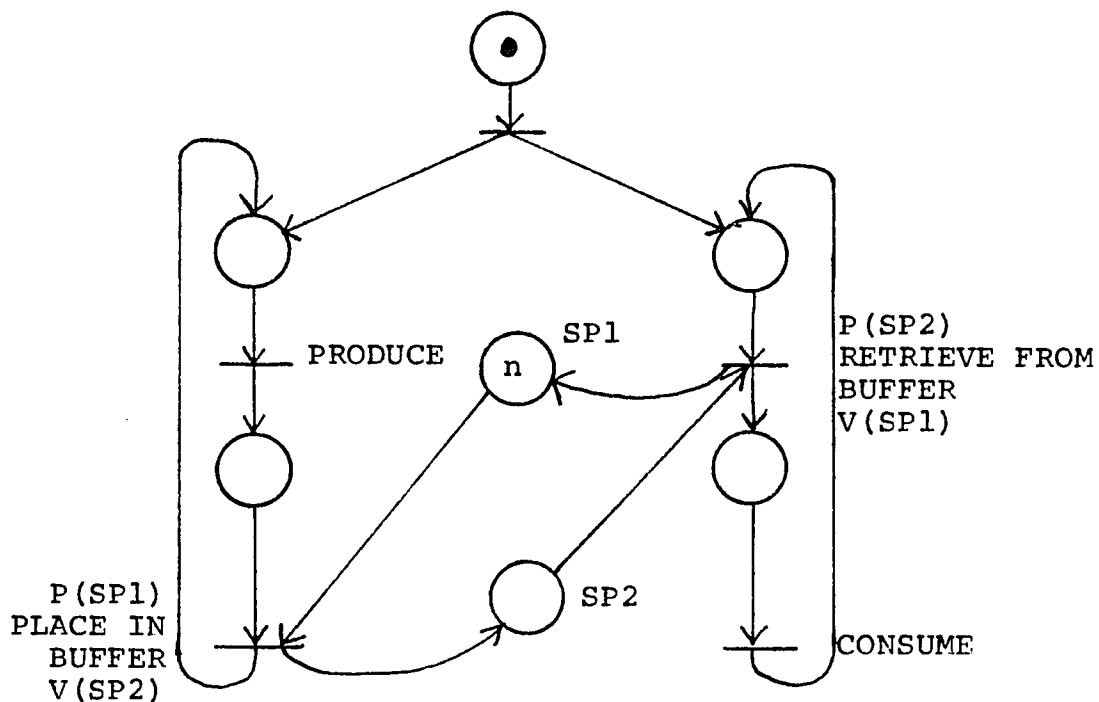


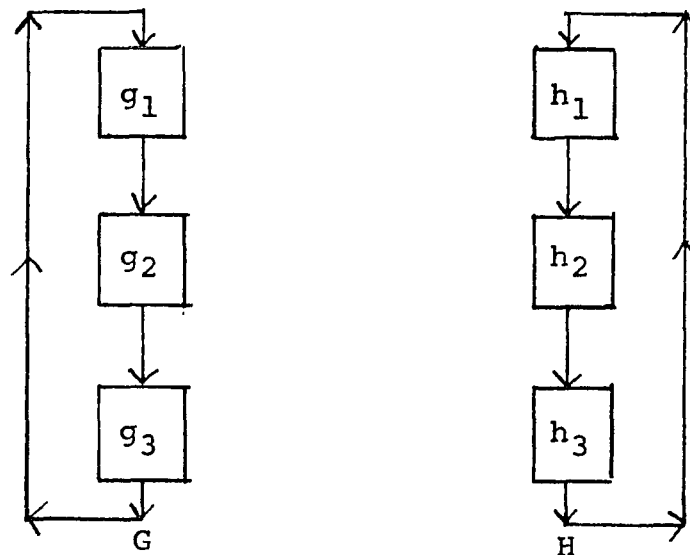
Figure 5-12

We next consider the problem of mutual exclusion. Let us define a section of a parallel procedure as some sequence of statements of that procedure.

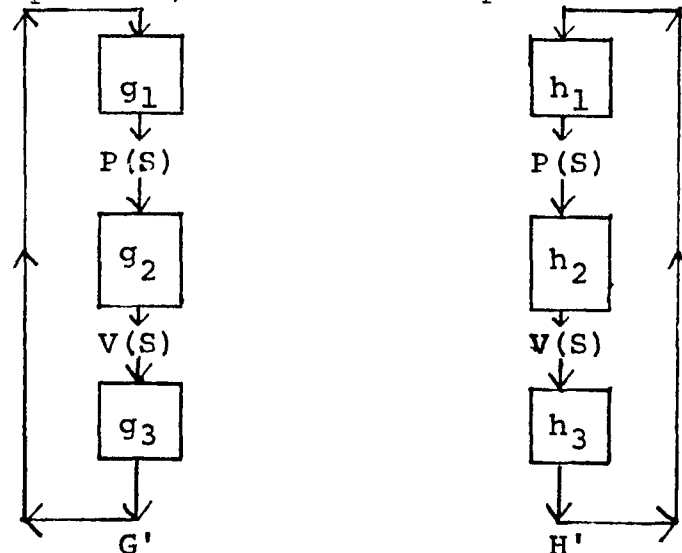
Another frequently occurring problem is that of mutual exclusion. The problem of mutual exclusion arises when we want to prevent the simultaneous or overlapping execution of different sections of a parallel procedure. These sections are called critical sections. For example, in a data base management system, if we had a section which updated some

information in the system and another section which interrogated ("reads") this information, we would want to prevent the two from overlapping since the information read during an update may be inconsistent.

Consider the following parallel procedure made up of sequential procedures G and H. Let the only constraint on their execution be that the execution of "subprocedures" g_2 and h_2 cannot overlap in any way (i.e., g_2 and h_2 are critical sections).



Using a semaphore S , this can be implemented as follows:



Notice we make no assumption about which subprocedure g_2 or h_2 is selected; the choice is completely arbitrary.

We can model the mutual exclusion problem using a Petri Net as follows:

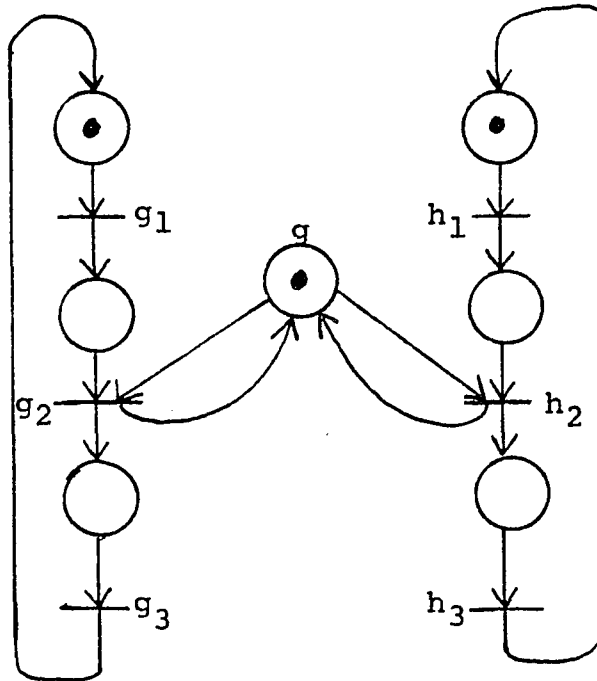


Figure 5-13

As a more interesting mutual exclusion problem, we consider the problem known as the "readers-writers" problem [25]. We allow any number of procedures to concurrently read from a file when that file is not being modified, i.e., written to. We give any "writers" priority so that after a writer requests to write, no subsequent reads are allowed until after all current writers are finished. There can be, at most, one writer writing at any time.

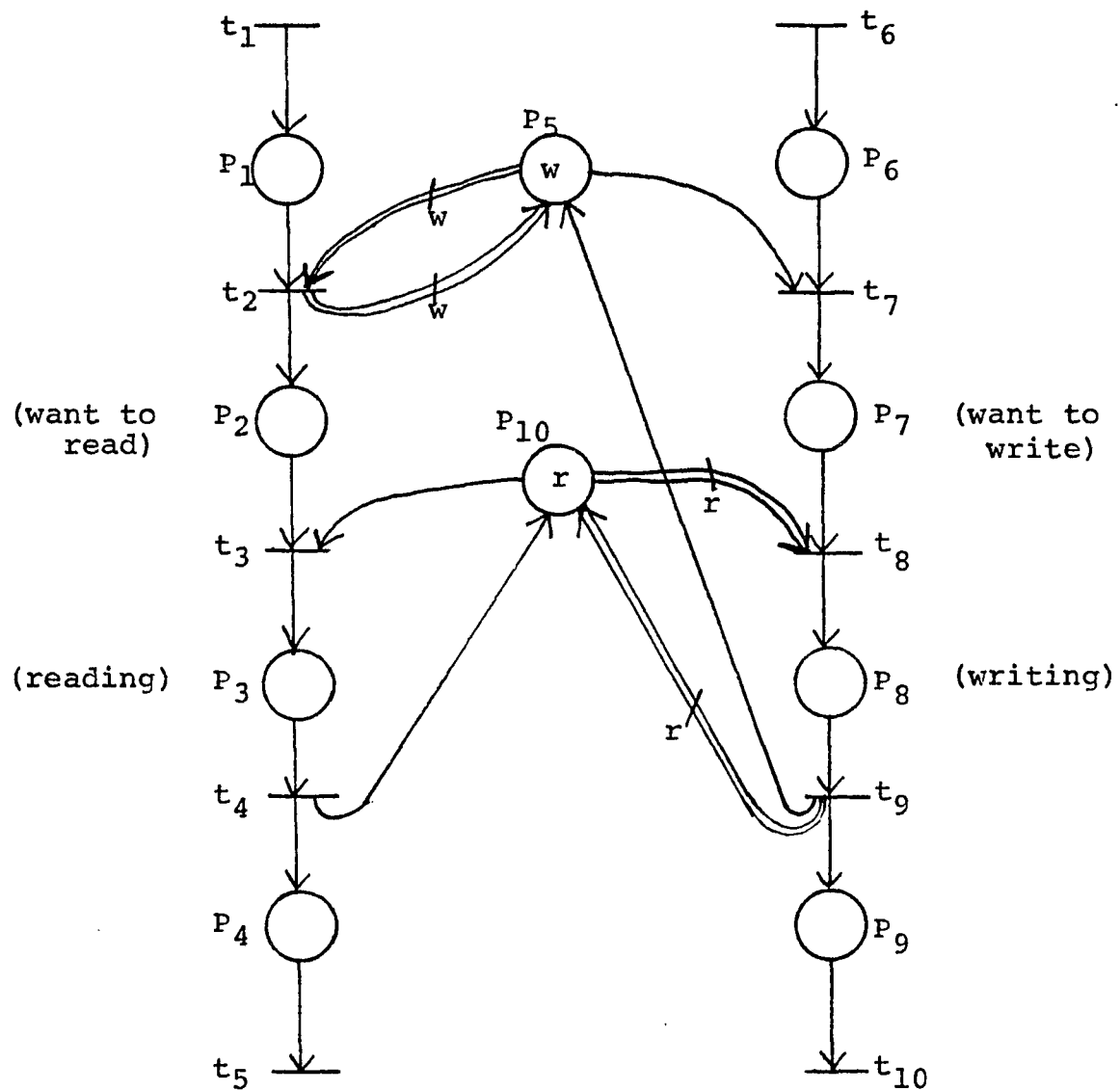


Figure 5-14

In the following diagram, we simulate how the markings might change in a readers-writers example with two readers and one writer.

	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	P ₇	P ₈	P ₉	P ₁₀	
Initial	0	0	0	0	1	0	0	0	0	2	
r ₁ arrives	1	0	0	0	1	0	0	0	0	2	
t ₂ fires	0	1	0	0	1	0	0	0	0	2	
t ₃ fires (read)	0	0	1	0	1	0	0	0	0	1	P ₁₀ indicates only 1 reader position remains
w ₁ arrives; t ₆ fires	0	0	1	0	1	1	0	0	0	1	
t ₇ fires	0	0	1	0	0	0	1	0	0	1	No other process moves through t ₇ or t ₂
r ₁ arrives; t ₁ fires	1	0	1	0	0	0	1	0	0	1	

Note: t₈ must wait for r₁ to finish reading

t ₄ fires	1	0	0	1	0	0	1	0	0	2
t ₅ fires	1	0	0	0	0	0	1	0	0	2
t ₈ fires	1	0	0	0	0	0	0	1	0	0
t ₉ fires	1	0	0	0	1	0	0	0	1	2
(r ₁ can read)										
t ₂ fires	0	1	0	0	1	0	0	0	1	2

Figure 5-15.1

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
t_{10} fires	0	1	0	0	1	0	0	0	0	0
t_3 fires	0	0	1	0	1	0	0	0	0	1
t_4 fires	0	0	0	1	1	0	0	0	0	2
t_5 fires	0	0	0	0	1	0	0	0	0	2

Figure 5-15.2

Note that P_5 and P_{10} act as semaphores. Further, note that at t_2 , all "write" tokens must be present in P_5 (i.e., no outstanding write requests) in order for a read request to proceed, thus, giving writers priority over readers. Also, note that t_8 requires all "read" tokens to be in P_{10} (i.e., no reading) while any writer is potentially active. Also note that r tokens are not returned to P_{10} until after write has finished so that no other write request can get past t_8 until current writing is finished.

5.4 A Pipeline Processor

Ramchandani [27] shows how a multi-stage pipeline processor can be modeled as a Petri Net. Notice how concurrency is possible among all the different stages.

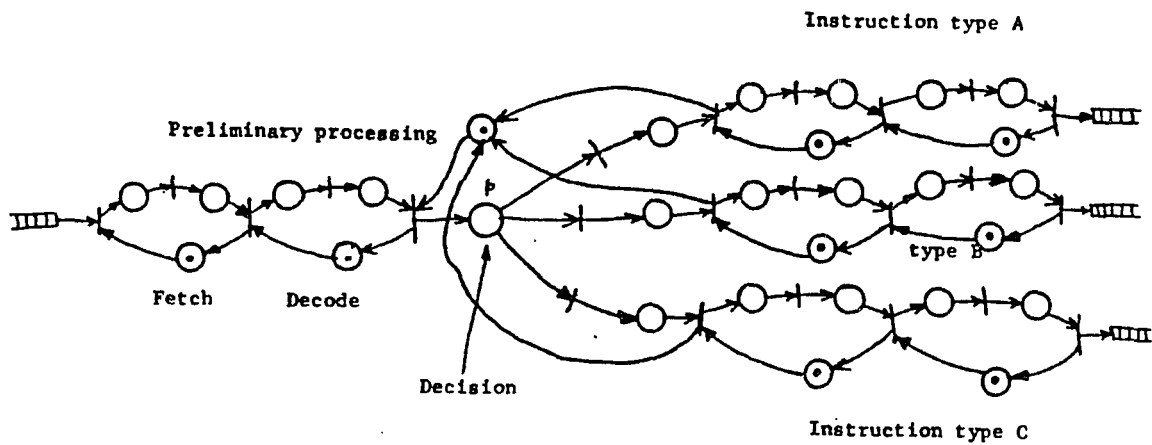


Figure 5-16

5.5 A Communication Protocol

In Merlin [24] the author examines the use of Petri Nets for describing communication protocols. Consider the following description of a protocol P_3 connecting processes P_1 and P_2 .

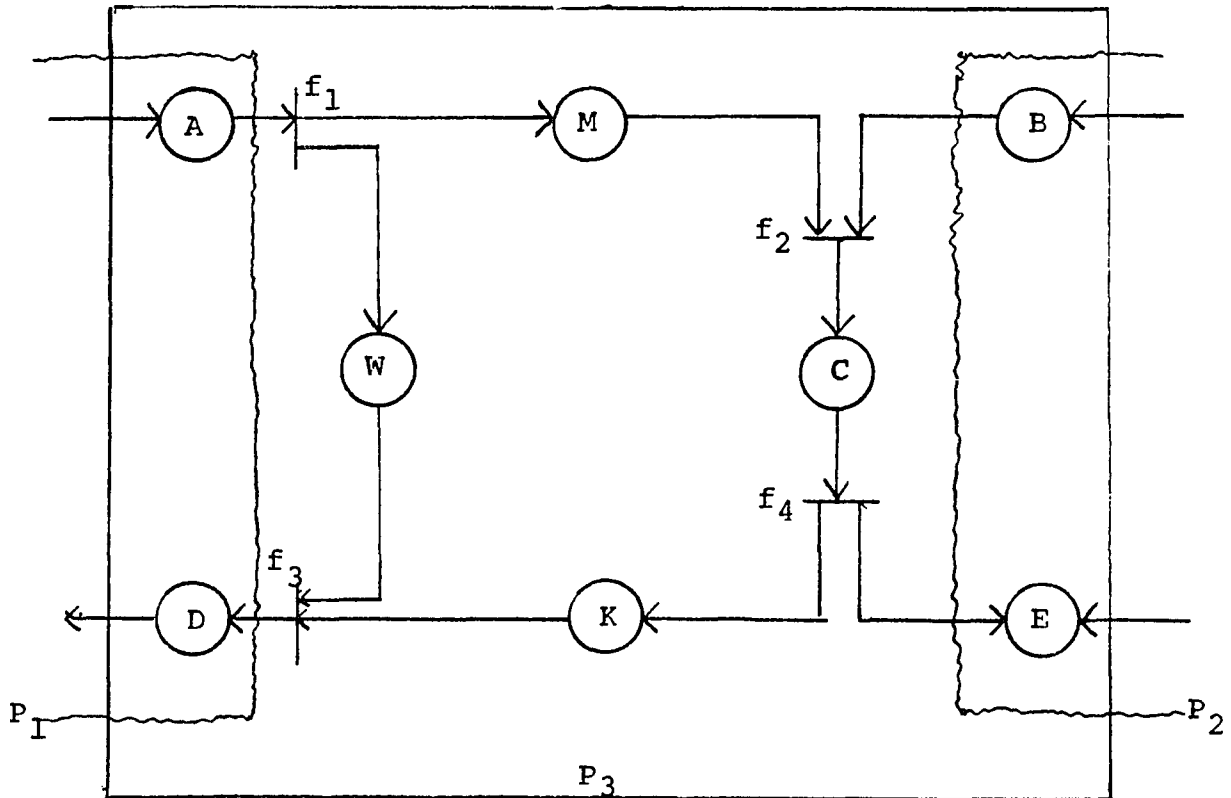


Figure 5-17

In this protocol:

- $A \neq 0$ indicates P_1 wants to send a message
- $M \neq 0, W \neq 0$ indicates the message is sent and now P_1 waits for acknowledgment
- $B \neq 0$ indicates P_2 is ready to receive the message
- $C \neq 0$ indicates message received
- $K \neq 0, E \neq 0$ indicates acknowledgment sent to P_1 and P_2 received message

$D \neq 0$ indicates to P_1 that the transfer has
completed

The author is interested in analysis and verification
of this and other Petri Net modeled protocols.

CHAPTER 6

CONCLUSIONS AND FUTURE RESEARCH

In this thesis we have made contributions to both the theory and application of Petri Nets.

In Chapter 2 we formally presented Petri Net Theory in a new algebraic setting. In so doing we extended the classical theory to include markings over different sets of objects. In particular, in Section 3.3 we gave a new application of Petri Nets by modeling the Banker's Problem. This required that our markings be extended over the real numbers.

Our generalization of marking algebra (i.e., graded marking algebras) will allow us to study hierarchical or meta-nets. By not requiring the places and transitions of a Petri Net to be finite sets we are able to consider Petri Nets which are structurally dynamic. We employed this concept in presenting Protection Systems (Section 3.1) and Data Base Systems (Section 3.2) as "plug-in" models. Of particular interest to us were the issues raised by the concurrent mode of operation inherent in these systems. In Section 2.3 we gave two characterizations of concurrency in Petri Nets and showed how we effectively compute these concurrencies. In Section 2.8 we presented formal definitions of operations which arise in Data Base Systems. This allowed us to show that our "plug-in" model of a Data Base System was both deadlock free and provided for consistent updating of the Data Base.

In Chapter 3 we presented three new applications of Petri Nets. We showed how the problem of safety arising in Protection Systems can be viewed in Petri Net Theory as a reachability problem. With respect to Data Base Systems we presented a Petri Net model under which queries dynamically attached to other queries, all sharing a set of relations. The Petri Net's firing rule insures consistency and a deadlock-free environment. For the Banker's Problem we showed how Petri Nets with markings over the real numbers can be used to study such systems.

In Chapter 4 we presented our work towards developing a system for the design of asynchronous concurrent systems (SYSDACS). We will continue to develop SYSDACS in the following ways:

- i) more formally develop the system language and its translation to a Petri Net;
- ii) develop a friendly user query language;
- iii) develop heuristics for searching the coverability tree in response to queries; and,
- iv) implement the system.

We feel that the underlying theory of SYSDACS and its high-level approach to systems design are important contributions of this thesis.

BIBLIOGRAPHY

1. ANSHEL, M. Vector Groups and The Equality Problem for Vector Addition Systems. Mathematics of Computation, Volume 32, Number 142, April, 1978, Pages 614-616.
2. ANSHEL, M. Decision Problems for HNN Groups and Commutative Semigroups. Houston Journal of Mathematics, Volume 4, No. 2, 1978.
3. BAKER, H. G. Rabin's Proof of the Undecidability of the Reachability Set Inclusion Problem of Vector Addition Systems. CSG Memo 79, Project MAC, MIT, July, 1973.
4. CODD, E. F. A Relational Model of Data for Large Shared Data Banks. Comm ACM 13, 6 (June, 1970) pp. 377-387.
5. CRAPO, H. H., ROTA, G. C. On the Foundations of Combinatorial Theory: Combinatorial Geometries. MIT Press, 1970.
6. CRESPI-REGHIZZI, S., MANDRIOLI, D. Petri Nets and Szilard Languages, Information and Control 33, 2 (February, 1977), pp. 177-192.
7. DATE, C. J. "An Introduction to Data Base Systems." Addison-Wesley Publishing Company, 1976.
8. DENNIS, J. B. Concurrency in Software Systems. Advanced Course in Software Engineering, Vol. 81, Lecture Notes in Economics and Mathematical Systems, Springer-Verlag.
9. DIJKSTRA, E. W. Co-operating Sequential Processes. "Programming Languages," Academic Press, 1968, pp. 43-111.
10. EILENBERG, S. Automata, Languages and Machines. Volume A, Academic Press, 1974.
11. ESWARAN, K. P., GRAY, J. N., LORIE, R. A., AND TRAIGER, I. L. The Notions of Consistency and Predicate Locks in a Data Base System. Comm. ACM 19, 11 (November, 1976).
12. FUCHS, L. Partially Ordered Algebraic Systems. Pergamon Press, 1963.

13. GINSBURG, S. The Mathematical Theory of Context-Free Languages. McGraw Hill, New York, 1966.
14. GOLD, E. M. Deadlock Prediction: Easy and Difficult Cases. Siam J. Computing, Vol. 7, No. 3, August, 1978, pp. 320-336.
15. HACK, M. H. Decision Problems for Petri Nets and Vector Addition Systems. MAC Technical Memorandum 59, Project MAC, MIT, March, 1975.
16. HACK, M. H. The Equality Problem for Vector Addition Systems is Undecidable. Computer Structures Group Memo 121, Project MAC, MIT, April, 1975.
17. HARRISON, M. A., RUZZO, W., AND ULLMAN, J. Protection In Operating Systems. Comm ACM 19, 8 (August, 1976).
18. HENDERSON, P. B., ZALCSTEIN, Y. Synchronization Problems Solvable by Generalized PV Systems. JACM, January, 1980, Vol. 27, No. 1, 60-71.
19. HOLT, A. W. AND COMMONER, F. Events and Conditions. Record of the Project MAC Conf. on Concurrent Systems and Parallel Computation. ACM, New York, 1970, pp. 3-52.
20. KARP, R. M., AND MILLER, R. E. Parallel Program Schemata. Journal of Computer and System Sciences, Volume 3, No. 2, May, 1969.
21. KARP, R. M., MILLER, R. E., WINOGRAD, S. The Organization of Computations for Uniform Recurrence Equations. JACM 14 (1967) 563-590.
22. LANDWEBER, L. H., ROBERTSON, E. L. Properties of Conflict Free and Persistent Petri Nets. JACM, Vol. 25, No. 3, July, 1978, 352-364.
23. LIPTON, R. J. AND SNYDER, L. A Linear Time Algorithm for Deciding Subject Security. JACM 24, 3 (July, 1977), pp. 455-464.
24. MERLIN, P. M., FARBER, D. J. Recoverability of Communication Protocols. Computer Communications of the IEEE, (February, 1976).
25. PETERSON, J. L. Petri Nets. ACM Computing Surveys, Vol. 9, No. 3, (September, 1977), 223-252.

26. PETRI, C. A. Communication with Automata. Supplement 1 to Technical Report RADC-TR-65-377 Vol. 1, Griffiss Air Force Base, New York, 1966. Originally published in German: Kommunikation mit Automaten, University of Bonn, 1962.
27. RAMCHANDANI, C. Analysis of Asynchronous Concurrent Systems by Timed Petri Nets. Technical Report TR-120, Project MAC, MIT, 1974.
28. SMULLYAN, R. M., First-Order Logic. Springer-Verlag, 1968.