Techniques of petri net reduction

Sreeranga Kalavapalli
New Jersey Institute of Technology

ABSTRACT

Techniques of Petri Net Reduction

by

Sreeranga Kalavapalli

Petri Nets have the capability to analyze large and complex concurrent systems. However, there is one constraint. The number of reachability states of the concurrent systems outweighs the capability of Petri Nets. Previous Petri Net reduction techniques focussed on reducing a subnet to a single transition and hence not powerful enough to reduce a Petri Net. This paper presents six reduction rules and discusses their drawbacks. A new reduction technique called Knitting Technique to delete paths of a Petri Net while retaining all the properties of the original net is presented. Further Structural matrix which facilitates reduction is presented.

TECHNIQUES OF PETRI NET REDUCTION

by

Sreeranga Kalavapalli

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer and Information Science

May 1993

APPROVAL PAGE

Techniques Of Petri Net Reduction

Sreeranga Kalavapalli

Dr. Daniel Yuh Chao, Thesis Adviser (Date)
Assistant Professor of Computer and Information Science,

[illegible]

Dr. David Wang (Date)
Assistant Professor of Computer and Information Science,
NJIT

[illegible]

Dr. M.C. Zhou (Date)
Assistant Professor of Electrical and Computer Engineering,
NJIT

BIOGRAPHICAL SKETCH

Author: Sreeranga Kalavapalli

Degree: Master of Science in Computer Science

Date: May 1993

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 1993
- Master of Business Administration in Computer Systems,
  Osmania University, Hyderabad, India, 1990
- Master of Science in Semi-Conductor Physics,
  S.V.University, Tirupati, India, 1978

Major: Computer Science

ACKNOWLEDGMENT

I would like to thank my Thesis Advisor, Dr. Daniel Yuh Chao for his help and guidance in completing this thesis. His constructive criticism coupled with the time he spent every week on the thesis helped me a great deal in doing this thesis. The group meetings and the presentations held every week helped to improve my knowledge in the field of Petri Net Reduction.

TABLE OF CONTENTS

LIST OF FIGURES

NOTATION

CG	common generation point
PG	pure generation
IG	interactive generation
PP generation	place-place path generation
TT generation	transition-transition generation
N	Petri Net
PSP	pseudo-process
n	the number of PSPs in Petri Net
π^w	new PSP
π	PSP
t	transition
p	place
C	local concurrenet set
C_{gj}	LCN(π_g, π_j)
X	local exclusive set
X_{ab}	LEX(π_a, π_b)
R	reachable set of markings
M	marking
GPN	general Petri Net
$\|\|$	concurrent
$\|$	exclusive
o	cyclic
->	sequentially earlier
<-	sequentially lateer
S_Matrix	Structural matrix
A_{ij}	entries of temporal matrix
CN	concurrent
d	synchronic distance
d^s	structural synchronic distance
ϵ	an element of
σ	pseudo process
L	language of Petri Net
N	the number of PSPs in a PN
VP	virtual PSP

CHAPTER 1

INTRODUCTION

Many real world concurrent systems are dynamic as well as complex. Any tool which aid in analyzing the highly complex and concurrent systems should have the capability to model concurrency, conflict and asynchrony. Petri Nets(PNs) besides having the above capability share other properties such as boundedness, liveness, completeness with the concurrent systems. Hence Petri Nets(PNs) have been adopted to model and analyze a large class of systems and proposed to model software [3]-[5]. However, as the system grows in complexity, the number of reachability states of the concurrent systems outweighs the capability of Petri Nets(PNs). This is called State Explosion Problem.

In order to solve the above problem researchers have proposed many reduction techniques, which emphasize on reducing the size of a given PN while preserving the properties such as liveness, boundedness, dead-lock free and reversibility and can be used to simplify analysis. The objective is to reduce several places and transitions into a manageable number of transitions or places and the conclusion of the analysis is valid to the original net. The reduction techniques proposed by many researchers so far focussed on reducing a subnet or a partial PN to a single transition [11]-[13].

In order to do so, the subnet to be reduced should behave equivalently to a transition. At present, there are only a few types of subnets that may be reduced to a single transaction such as single output transaction or some other simple structures [14]. In most of the existing methods the test of a reduced net has not been automated and is often very complex. Hence there is a necessity to adopt a new approach which ensures deletion of paths of a PN in a gradual and automated fashion.

A new technique which can reduce nets more efficiently than the traditional techniques is proposed here. This technique enhances the vigor of the current reduction techniques. Similar to the generation of paths using synthesis rules, the deletion of paths must preserve certain properties, both desired and undesired, of the original net. Such an approach for generalized PNs was demonstrated by Koh[7]. Yet another technique called Knitting Technique can synthesize PNs[9] that cannot be synthesized by Koh's fusing technique. Adopting same rules of the knitting technique in the reverse direction PNs can be reduced more effectively than the other techniques.

Yaw[8]-[10] developed a rule-based interactive technique called Knitting Technique. The technique contains some simple rules which can guide the synthesis of PNs with the desired properties. New paths and cycles to a PN are added in an incremental way. The system properties such as boundedness, liveness and reversibility are retained by the PNs at each step. Hence the tiresome analysis of these properties can be avoided by the designers while building a Petri Net model for a complicated system. There are two types of knitting rules- TT and PP rules. These rules guarantee incremental correctness at each step of synthesis.

A structural matrix is used to record the relationship (sequential, cyclic, concurrent, exclusive, etc., between any two pseudo-processes in a PN such that the maximum concurrency at any execution stage is available. Hence a structural matrix can be used to perform reduction of a Petri Net. Most of the earlier reduction techniques deal with local transformation. The transformation by structural matrix can be termed as a global technique

The remaining chapters of the paper are organized as follows. Chapter 2 presents some behavioral properties of PNs. Chapter 3 presents general synthesis rules and some illustrations. Chapter 4 discusses six reduction rules and the drawbacks of previous reduction rules. Chapter 5 discusses structural matrix for PNs, the algorithm for constructing a structural matrix and an application as an illustration. Chapter 6 discusses the reduction algorithm and an application of the same. The conclusions are given in Chapter 7.

CHAPTER 2

BEHAVIORAL PROPERTIES OF PETRI NETS

A major strength of Petri Nets is their support for analysis of many properties and problems associated with concurrent systems. Most concurrent systems are designed to achieve cyclic behavior such that the Petri Net model can be made applicable to them. A brief review of various properties of Petri Nets is presented below for better understanding of the paper.

2.1 Reachability

The firing of an enabled transition of a Petri Net will change the token marking. A sequence of firings will result in a sequence of markings. A marking M_n is said to be reachable from a marking M_0 if there exists a sequence of firings that transform M_0 to M_n.

2.2 Boundedness

A Petri Net is said to be bounded if the number of tokens in each place does not exceed a finite number. Places in a Petri Net are often used to represent buffers and registers for storing intermediate data. By verifying that the net is bounded or safe, it is guaranteed that there will be no overflows in the buffers or registers.

2.3 Liveness

A Petri Net is said to be live if no matter what marking has been reached from the original place. It is possible to ultimately fire any transition of the net by progressing through some firing sequence. This means that a PN is said

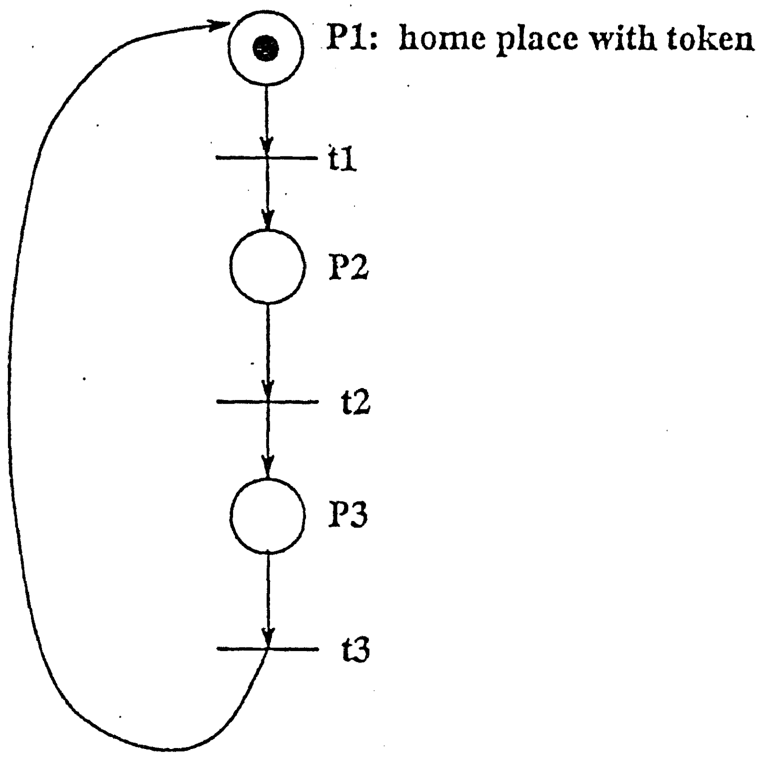


Fig. 2.1 A Basic Process with Home place

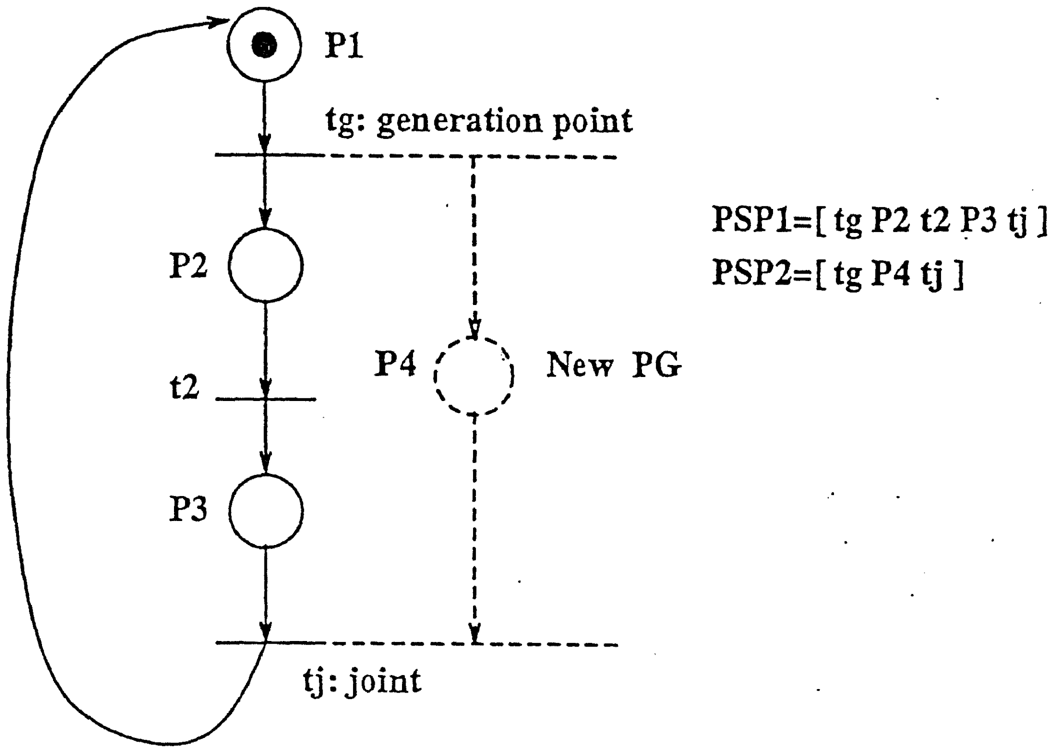


Fig. 2.2 Pure Generation(dashed line) by using TT rule and Forward TT Generation

to be live, if it guarantees dead lock free operation, no matter what firing sequence is chosen.

2.4 Reversibility and Homestate

A PN is said to be reversible if, for each initial marking $M_0 \epsilon R(N,M)$ for every $M \epsilon R(N,M_0)$ i.e., the initial marking is recovered from any other marking.

2.5 Iteration Period

A reversible PN is the PN whose initial markings can always be recovered; the corresponding period is termed as the iteration period.

2.6 Home Place

A home place in a PN is the place with token(s) such that it can start the iteration period. Let transition t_α be enabled in the initial marking of a PN and place p_h an input place of t_α. This p_h is defined as a home place. A basic process in a PN is a cycle path which contains a home place. (shown in Fig. 2.1)

2.7 Pseudo-Process, Generation Point and Joint

A pseudo-process (PSP) in a PN is a directed elementary path where the indegree and outdegree of any node (transition or place) inside the path are both one except its two end nodes; the starting node is defined as the generation point and the end node as the joint (shown in Fig. 2.2). Any other node in the PSP has single input node and a single output one. A PSP cannot be subset of any other PSPs, it is an elementary process of a PN.

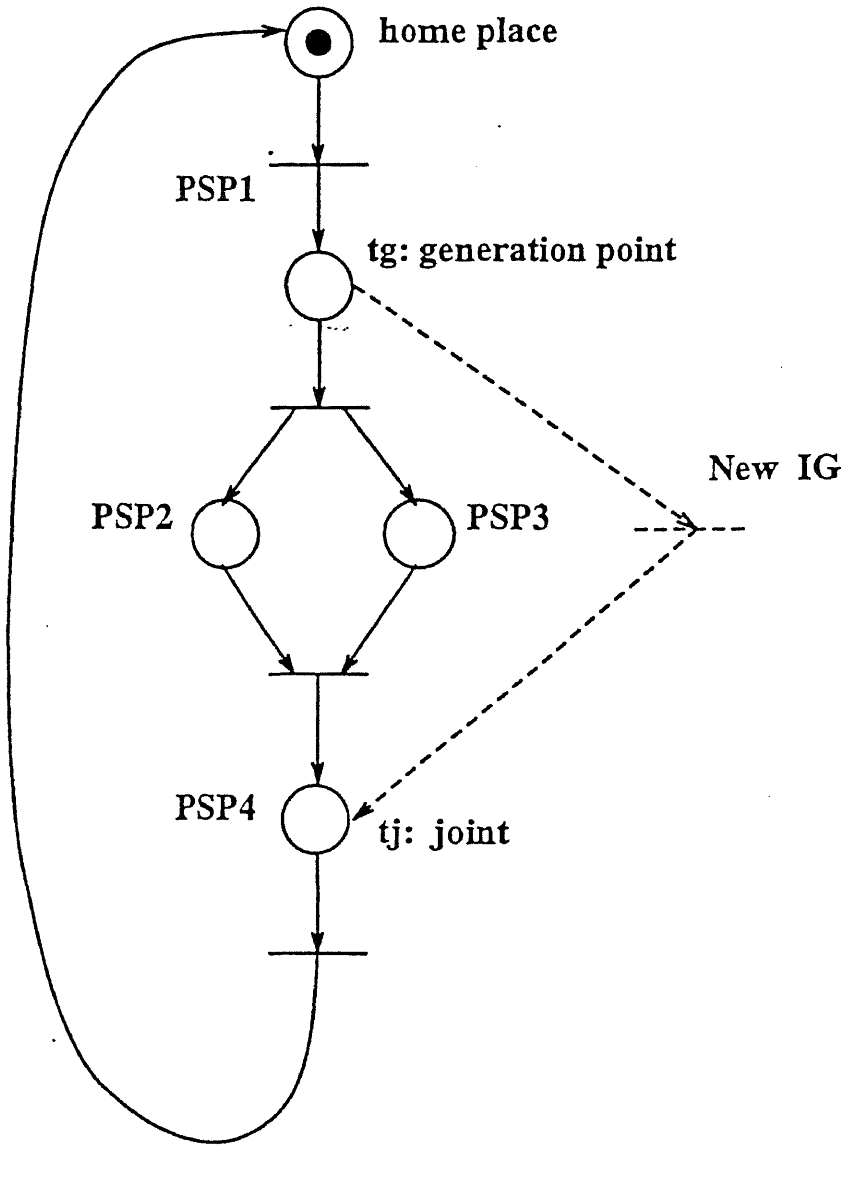


Fig. 2.3 Interactive Generation (dashed line) connecting PSP1 and PSP4 by using PP rule

2.8 Virtual Pseudo Process

A virtual pseudo process (VP) is a PSP with only two nodes, the generation point and joint.

2.9 Pure Generation and Interactive Generation

A new PSP (NP) connects from PSP_g to PSP_j, if $PSP_g = PSP_j$. It is a pure generation (PG). otherwise it is an interactive generation (IG). If both the generation point and the joint of a NP are transitions (places), it is a TT generation (PP generation) as shown in Fig. 2.2 and Fig. 2.3; the corresponding path is called a PP-path(TT-path). t_g (t_j) is the output (input) transition of the generation point (joint) of a PP-path. P_g (P_j) is the output (input) place of the generation point (joint) of a TT-path.

2.10 Structural Relationship between Two PSPs

The structural relationship between two PSPs, π_1 and π_2, within an iteration is the first one of the following definitions that the two PSPs satisfy.

2.10.1 Sequential and Cyclic

π_1 and π_2 are sequential to each other if there exists a P^- not including any home place from (1) π_1 to π_2 or vice versa. If only (1) [(2)] holds, π_1 is sequentially earlier [later] to π_2 , denoted as π_1 -> [<-] π_2. If both hold, then π_1 is cyclic to π_2, denoted as $\pi_1 \circ \pi_2$.

2.10.2 Concurrent (Exclusive)

π_1 is concurrent (exclusive) to π_2, denoted as $\pi_1 \| (|) \pi_2$, if both are not sequential to each other and there exists a prime t_s (p_s) or no directed path exists from π_1 to π_2.

2.11 Structural Relationship between Two Nodes

If two nodes (transition or place) are in two different PSPs, their structurral relationship follows that of the two PSPs. If they are in the same PSP, they are sequential to each other.

It is easy to see that if we execute one iteration of a safe and strongly connected PN involving π_1, then π_1 -> π_2 implies that π_1 is executed earlier than π_2; $\pi_1 \mid \pi_2$ implies that there is no need to execute π_2 to complete one iteration; intuitively, two PSPs, $\pi_1 \mid\mid \pi_2$ are sequential to each other if they are subject to an intra-iteration precedence relationship; i.e., if both are executed in one iteration, then one is executed earlier than the other. π_1 and π_2 are concurrent to each other if they proceed in parallel. π_1 and π_2 are exclusive to each other if it is possible to complete one iteration with only one of them being executed.

2.12 Synchronic Distance

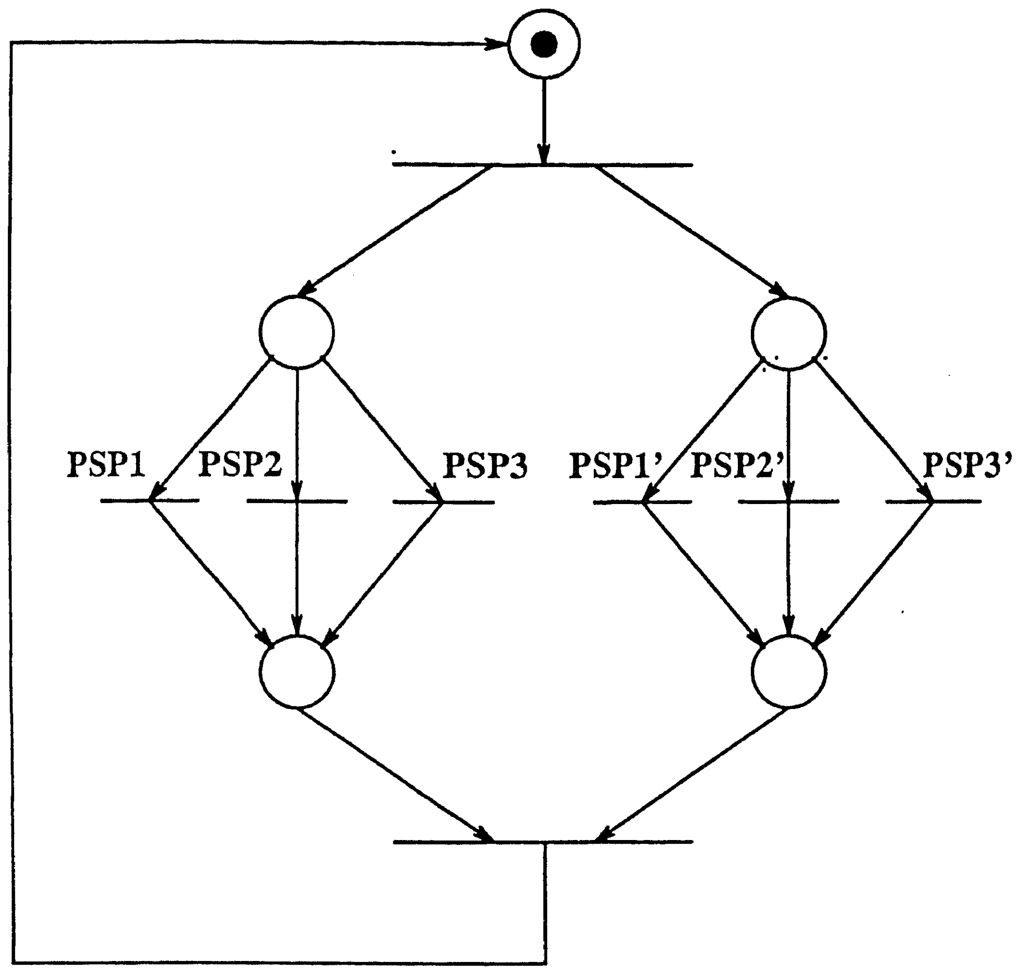
Synchronic distance is a concept closely related to the degree of mutual dependence between two events in a condition/event system. The definitions of Petri Net language and Synchronic Distance are given as follows:

The language of a Petri Net N, L(N, M), is the set of all firing sequences for the net with the initial marking M : $L(N, M) = \{ \sigma \mid M [\sigma > M'] \}$.

The synchronic distance between two transitions t_1 and t_2 in a petri net N is defined as $d_{12} = \text{Max} \{ \sigma (t_1) - \sigma(t_2), \sigma \in L(N, M) \}$, where $\sigma(t)$ is the number of times t appears in σ.

This definition can be extended to sets. Thus d (W,V) is the maximum number of firings of transitions in set W without any transition firing in set V.

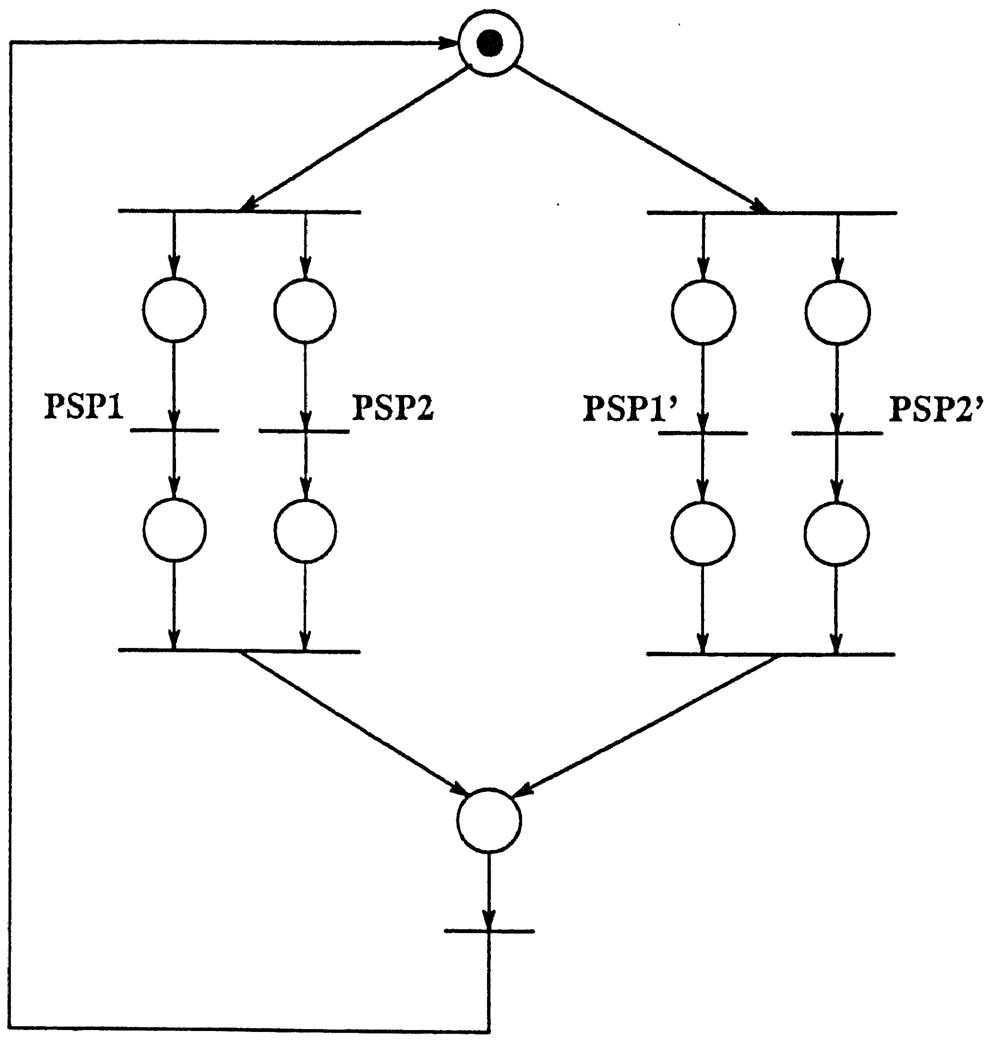
Depending on whether $d_{gj} = \alpha$ or 1, different synthesis rules are constructed for a new TT - generation. If we generate a TT-path between two



LEX (PSP1, PSP1') = { PSP1, PSP2, PSP3 }

LEX (PSP1', PSP1) = { PSP1', PSP2', PSP3' }

Fig. 2.4 Local Exclusive Set



LCN (PSP1, PSP1') = { PSP1, PSP2 }

LCN (PSP1', PSP1) = { PSP1', PSP2' }

Fig. 2.5 Local Concurrent Set

transitions t_g and t_j with infinite synchronic distance, unbounded tokens could accumulate in the TT-path when t_g fires infinitely without firing t_j once. This synchronic distance between two concurrent (or sequential) transitions t_g and t_j may be α if there exists a transition $t_x \mid t_j$. However, the synchronic distance between the two sets of transitions containing t_g and t_j respectively may equal one. One (called X_{gj}) contains t_g and all transitions that are exclusive to t_g but concurrent (or sequential) to t_j ; the other (X_{jg}) contains t_j and all transitions that are exclusive to t_j but concurrent (or sequential) to t_g. The following defines a similar X_{ik} whose members are PSPs rather than transitions.

2.13 Local Exclusive Set

A local exclusive set (LEX) of π_i with respect to π_k, X_{ik}, is the maximal set of all PSPs which are exclusive to each other and are equal or exclusive to π_i, but not to π_k. i.e.,X_{ik}= LEX(π_i, π_k) = { $\pi_z \mid \pi_z = \pi_i$ or $\pi_z \mid \pi_i$, $\pi_z \mid$ (not exclusive to π_k}. Fig. 2.4 is an example for a local exclusive set.

2.14 Local Concurrent Set

A local concurrent set (LCN) of π_i with respect to π_k, C_{ik} = LCN (π_i, π_k), is the maximal set of all PSPs which are concurrent to each other and are equal or concurrent to π_i but not to π_k, i.e., C_{ik} = C (π_i, π_k) = { $\pi_z \mid \pi_z = \pi_i$ or $\pi_z \mid\mid \pi_i$, $\pi_z \mid\mid$ (not concurrent to) π_k }. Fig. 2.5 is an example for a local concurrent set.

The definition of LEX [LCN] between two PSPs can be extended to the LEX [LCN] between two transitions [places]. Thus LEX (t_i, t_k) [LCN(p_i, p^k)] is a set of transitions [places], instead of PSPs. Let G(J) denote the set of all π_g (π_j) involved in a single application of the TT or PP rule. To avoid the unboundedness problem, it is necessary to have a new directed path from each PSP in X_{gj} to each PSP in X_{jg}. Both X_{gj} and X_{jg} can be determined from the S_Matrix even though $d_{gj} = \alpha$, which is, therefore, of no use to the synthesis. For instance, we have observed that when $t_g \mid\mid t_j$ and $d_{gj} = 1$, the inforamation of $d_{gj} = \alpha$ is of no use and we define that $ds_{gj} = 1$. On the contrary, when $d_{gj} = \alpha$, the TT-generation from t_g to t_j will cause unbounded places in the TT-path. This problem cannot be detected by just checking the T-Matrix and the information of $d_{gj} = \alpha$ is indeed useful here and the value of ds_{gj} remains to be α. Hence we have the following definition of structural synchronic distance. The structural synchronic distance between π_g and π_j is $ds_{gj} = 1$ if d_{gj} is finite and $d_{gj} = \alpha$ if $d_{gj} = \alpha$.

CHAPTER 3
RULES FOR SYNTHESIS

Consider a Petri Net modeling a set of interacting entities. At the outset, each entity has a home place with tokens indicating that it is ready to fire. Each entity conducts consecutive tasks (which can be considered as transitions), and then returns to the original ready state. Each state can be considered as a place. Executing a task means firing a transition. At the completion of executing each task, the token leaves the original state and enters the next state by firing a transition. Thus, in the corresponding Petri Net model, the token moves from the home place through consecutive places by firing a sequence of transitions. Eventually, the token will return to the home place, forming a cycle as shown in Fig.2.1. The process yielding this cycle is termed a Basic Process.

3.1 Guidelines For Synthesis

By a set of path generations the Petri Net can be expanded starting from the basic process. Each path is a PSP. The path generations are of two types: **Pure Generation (PG)** and **Interactive Generation (IG)**. PG generates paths within a single PSP. That means a single PSP can be expanded into a set of PSPs by adding paths. PG processes are pure growth processes. They do not involve any interactions between PSPs as shown in Fig. 2.2. Interactive Generation (IG) generates paths between PSPs to interconnect a set of PSPs. So this method of generation involves interaction between PSPs as shown in Fig. 2.3.

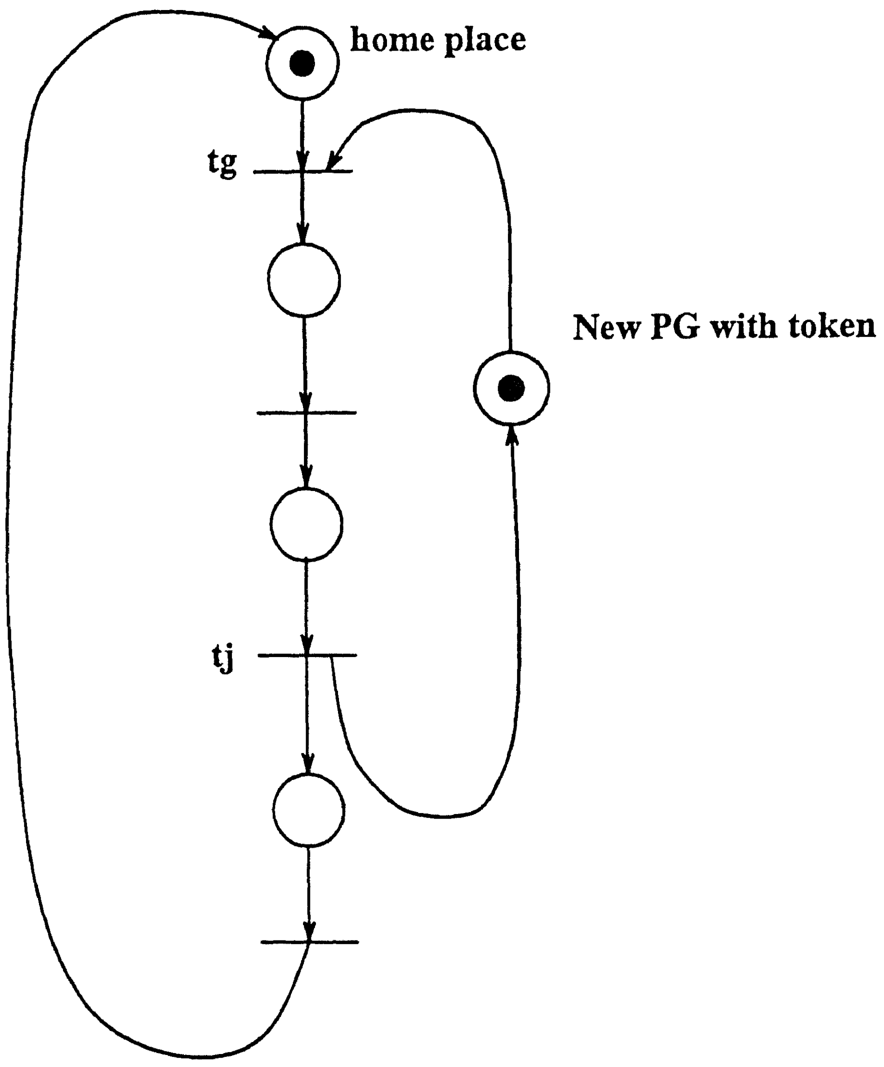


Fig. 3.1 Backward TT Generation

Paths can be generated in two ways for both Pure generation and Interactive generation. The two types are: **TT rule and PP rule.** The TT rule (transition to transition) is applicable to generation of paths from transition to transition, from the generation point t_g to the joint t_j; the PP rule (place to place) is applicable to generation of paths from place to place, from a generation point P_g to P_j. The Figs. 2.2 and 2.3 explain this concept. There are two types of generations: **Forward and backward.** If P_g ->P (t_g-> t_j), then it is a forward PP (TT) generation (Fig. 2.2); it is a backward PP(TT) generation if the direction of -> changes. If it is a backward TT generation, insert token(s) in the new cycle if it does not have any token (Fig. 3.1).

Given a synthesized net, N_1, a set of new paths are generated using some synthesis rules to form another net, N_2. The marking in NP upon the generation is called its initial marking. This synthesis rules should be such that all transitions and places in Petri Net N_2 are also live and bounded. This can be ensured by.

1) Inhibiting any intrusion into normal transitions firings of N_1 prior to the generations. This guarantees that neither unbounded places nor dead transitions would occur in N_1 since it was live and bounded prior to the generations.

2) Inhibiting any dead transitions and unbounded places in NP.

There are two kinds of intrusion. One is to change the marking of N_1; the other is to eliminate some reachable markings. Inorder for NP to be live, it must be able to get enough tokens from P_g or by firing t_g. When tokens in NP disappear, the resultant marking of the subnet N_1 in N_2 must be reachable in N_1 prior to the generation of NP. Further, the marking of NP must be reversible in N_2.

The second intrusion may occur when the joint is a transition which may not be enabled in case of backward TT generation; hence causing some markings in N_1

unreachable. Thus, if the joint is a transition, NP must be able to get tokens from a generation point within each iteration.

Based on the concept of no intrusion, the rules are constructed according to the following guidelines

1) From M_0, each t_g must be potentially (always or has fired for every σ in N_1 enabling the joint which is a transition) firable.

2) These tokens must disappear from NP before it gets unbounded tokens, and return to N_1 to reach a marking M_2. The marking of the subnet N_1 in N_2, M_2 (N_1) of all possible sets of such M_2 are identical to those in N_1 without the new paths.

Guidelines (1) and (2) guarantee no intrusion. Guideline (2) guarantees no dead transitions and no unbounded places in NP (since all tokens can disappear from NP).

Synthesis rules are presented below. It should be noted that the TT and PP rules are merged together. The words and notations inside the parentheses are for the PP rules.

3.2 TT (PP) Rule

Generate a π^W from t_g (P_g) ϵ π_g to t_j (P_j) ϵ π_j

1) TT.1 (PP.1): If $\pi_g = \pi_j$, then signal " a pure TT (PP) generation".

2) TT.2: If $t_g <- t_j$, then signal "forming new circles, creating a new home place ". If there does not exist a σ to fire t_g without firing t_j, then insert a token in a place of π^W. If $\pi_g = \pi_j$, then return and the designer may start a new generation.

If $t_g \mid\mid t_j$ ($p_g \mid p_j$) or t_g (p_g)-> t_j (p_j) or t_j (p_j) -> t_g (p_g), and if π_g not equal to π_j. then signal "an interactive TT (PP) generation ".

3) TT.3: If the structure synchronic distance $d_{gj}^S = \alpha$ then

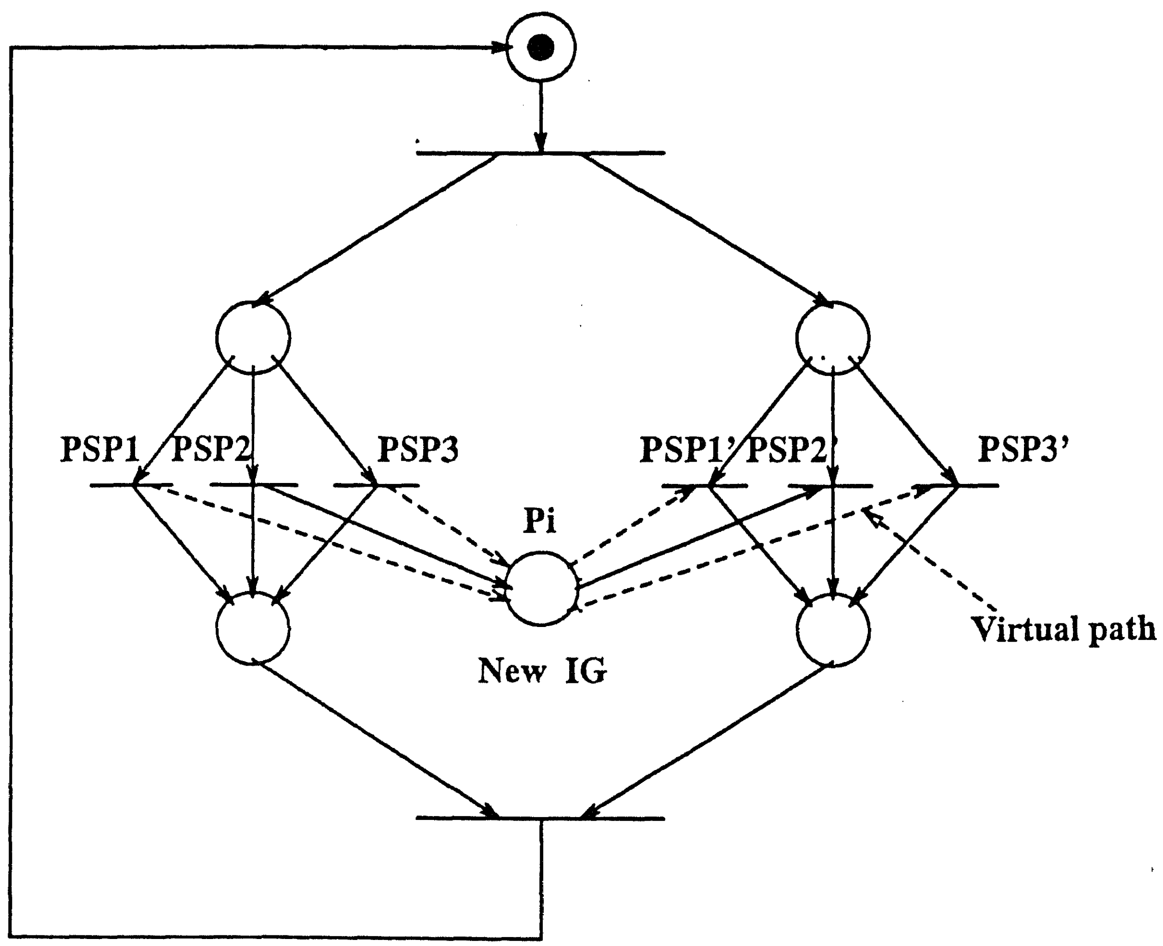


Fig. 3.2 An Example of the Application of the TT.4 Rule (Completeness Rule 1)

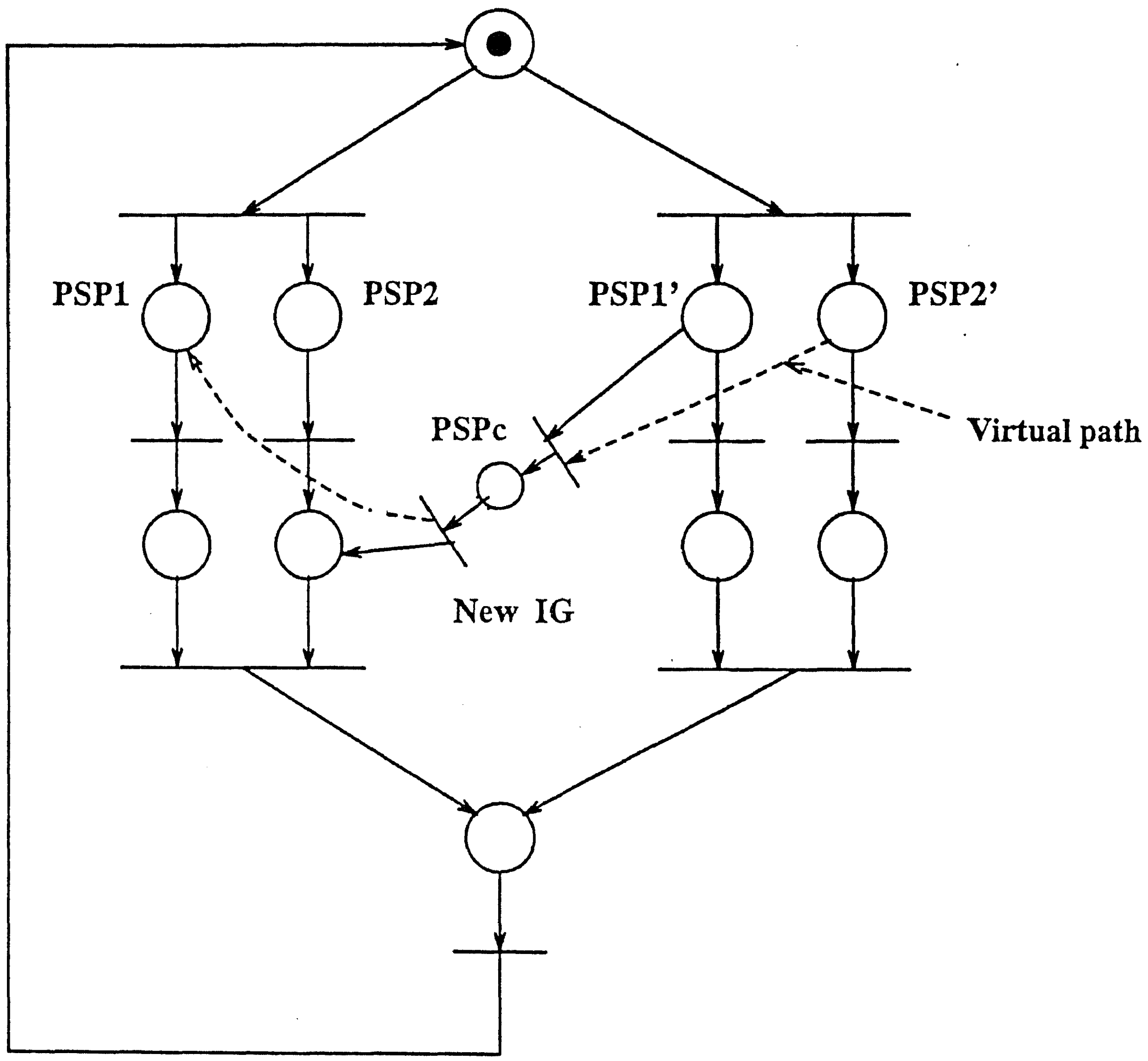


Fig. 3.3 An Example of the Application of the PP.2 Rule (Completeness Rule 2)

3.a) TT.3.1: Apply TT.4 rule,

3.b) TT.3.2: Generate a new TT-path to synchronize t_g and t_j so that after step 3.c, d_{gj}^S changes from α to 1.

3.c) Go to step 2.

4) TT.4: Or Completeness Rule 1 (PP.2 or Completeness Rule 2)

4.a) TT.4.1 (PP.2.1): Generate a TP- path from a transition t_g of each π_g in X_{gj} (a t_k of a π^W) to a place p_k (p_j of each π_j in C_{jg}) in a π^W.

4.b) TT.4.2 (PP.2.2): Generate a virtual PT- path from the place p_j (a p_g of each π_g in C_{gj}) to a transition t_j of each π_j in X_{jg} (the t_g in NP).

5) TT.5 (PP.3): For all other cases, (e.g., $t_g \mid t_j$ and $p_g \mid\mid p_j$), signal "forbidden, delete the new PSP" and return. Note that during each application of a rule, more than one π^W may be generated; we call such a generation a macro generation. Otherwise, it is a singular generation. An example of the application of the TT-4 rule (completeness rule) is shown in fig. 3.2. Similarly an example of the application of the PP.2 rule (completeness rule 2) is shown in fig. 3.3.

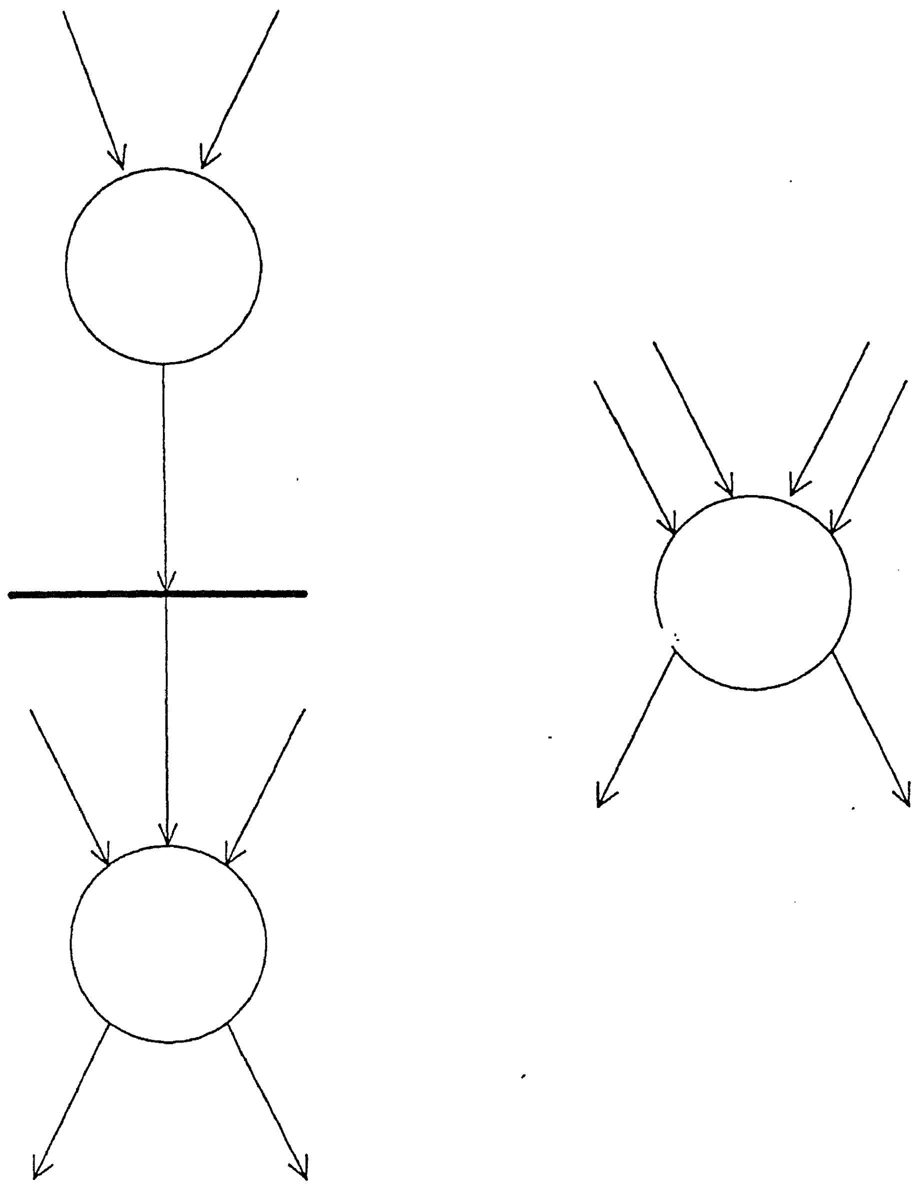


Fig. 4.1 Fusion of Series Places

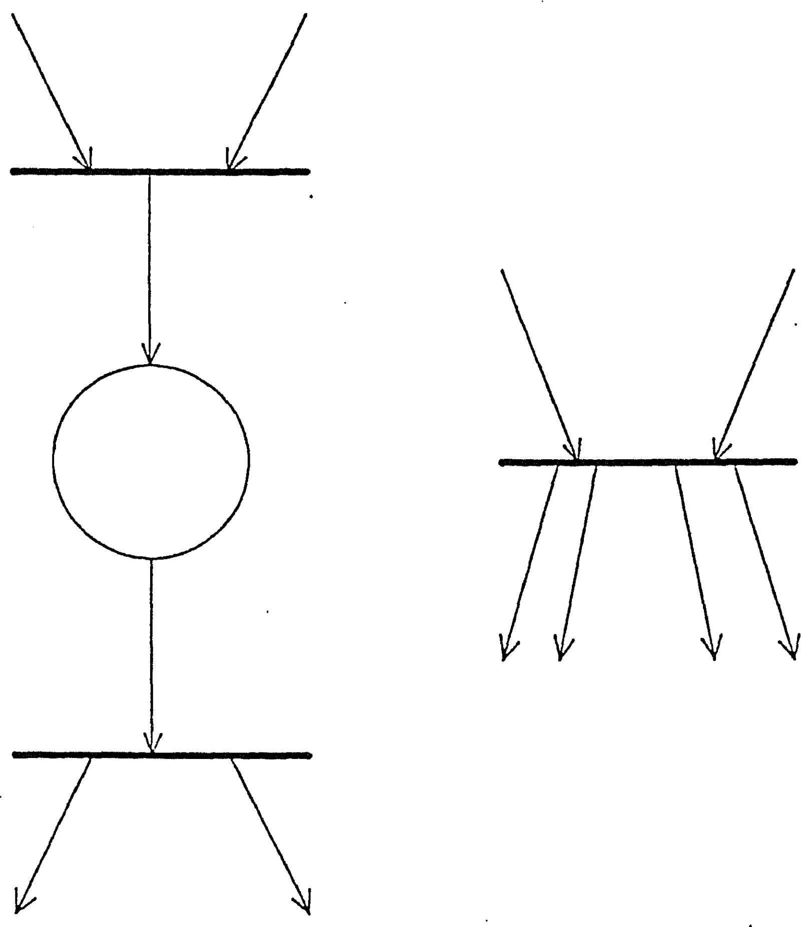


Fig. 4.2 Fusion of Series Transitions

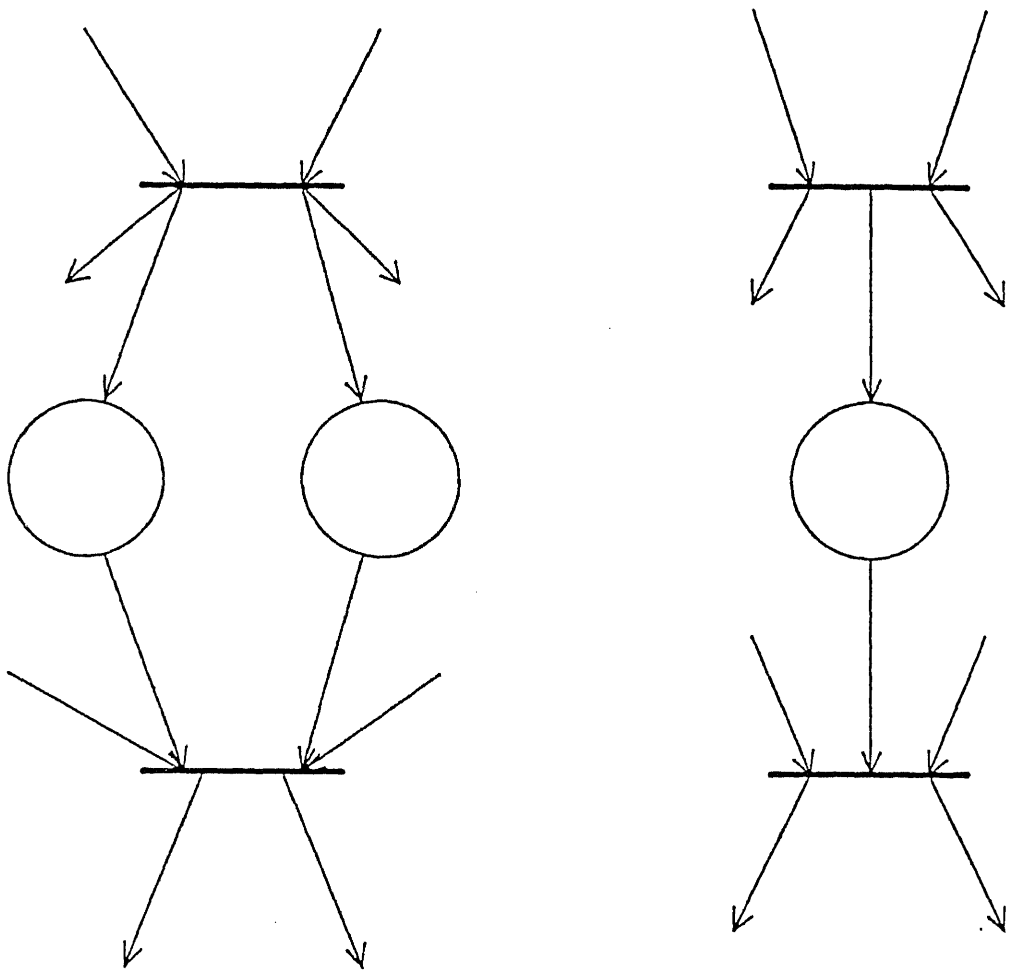


Fig. 4.3 Fusion of Parallel Places

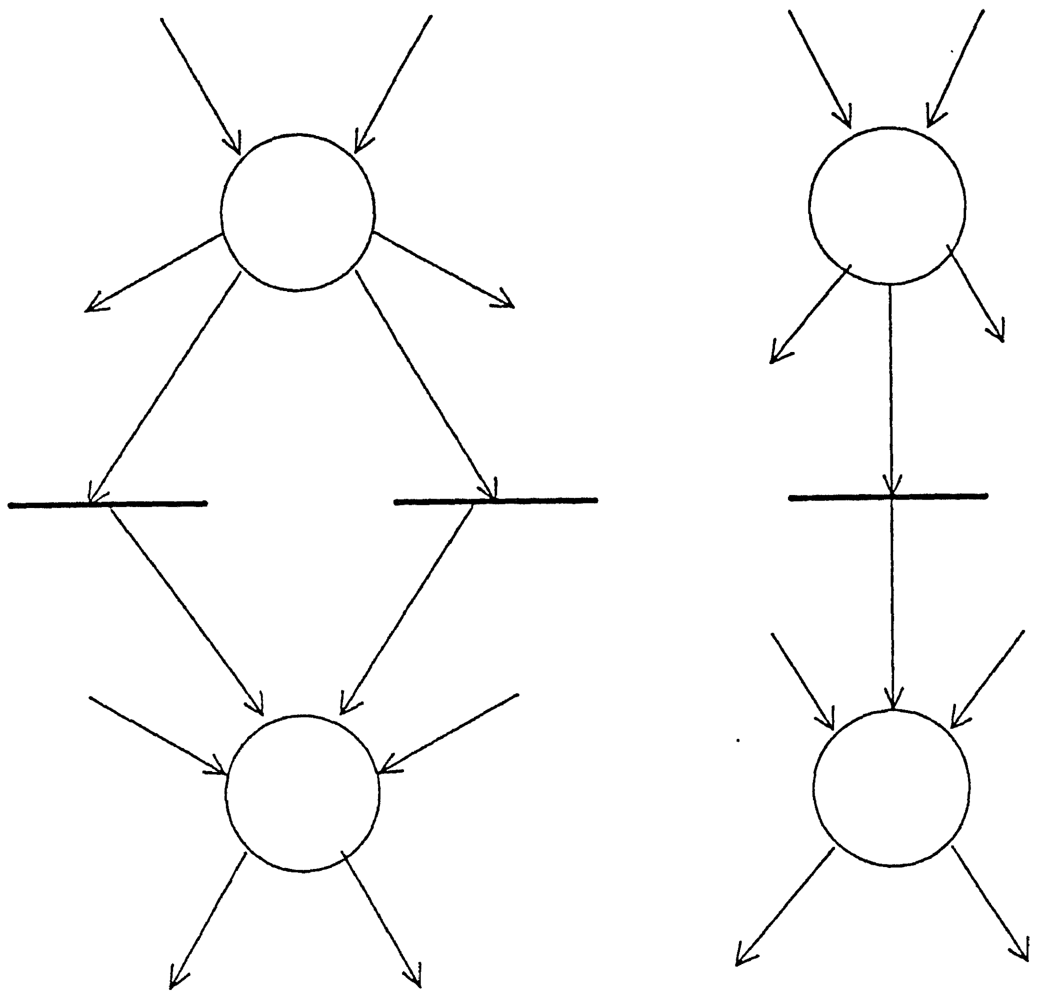


Fig. 4.4 Fusion of Parallel Transitions

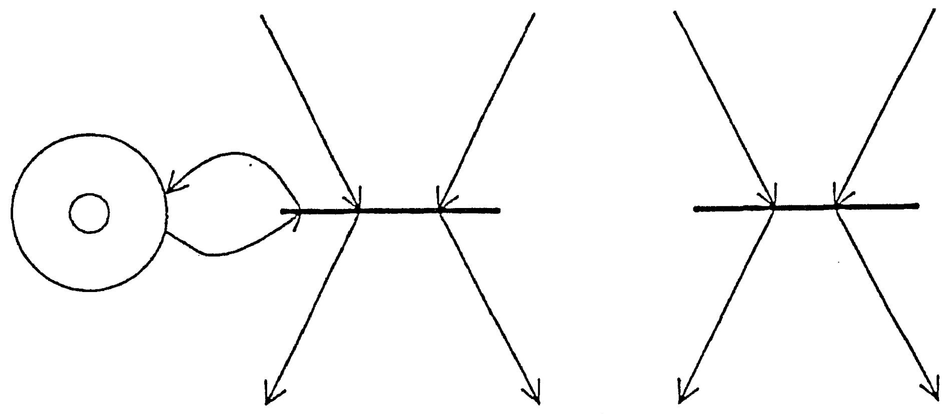


Fig. 4.5 Elimination of Self Loop Places

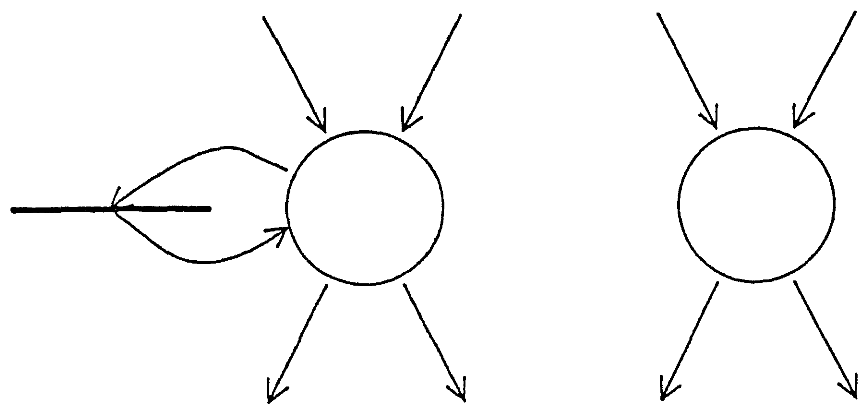


Fig. 4.6 Elimination of Self Loop Transitions

CHAPTER 4
SIX RULES OF REDUCTION

The six Reduction rules are as follows

1. Fusion of series places (FSP)
2. Fusion of series transitions (FST)
3. Fusion of parallel places(FPP)
4. Fusion of parallel transitions(FPT)
5. Elimination of self loop places (ESP)
6. Elimination of Self loop Transitions(EST)

The above six operations are shown in the figures 4.1 thru 4.6. From the figures it can be seen that the reduced places or transitions retain all the properties of the original sub nets.

4.1 Drawbacks in Existing Reduction Techniques

The present techniques reduce subnets or partial Petri Nets to a single place or transition. To satisfy the requirement the subnet to be reduced must behave, viewing from both inside and outside of the subnet, equivalently to a transition. This has limited the types of subnets that may be reduced to a single transition. Currently, the reducible subnets need to be of single input-transition, single-output-transition,or some other simple structures. Further these techniques do not detect dead locks or unbounded places during the reduction process. Further the reduction techniques discussed above do not reduce Petri Nets with identical characteristics. Petri Net reduction is to replace a module with a transition. A module in a Petri Net is subnet whose interfaces with other parts of the net are through certain well defined

transitions in the module. A set of modules can be reduced to only one module under the following conditions:

1. They have common output places and input places;
2. They are disjoint;
3. Each module has only one entry transition.

A set of Transitions can be reduced to a single transition,if they have identical characteristics viz., same input/output places and the number of arcs in between.

4.2 Enhancement of Reduction Techniques

Structural matrix can be used to perform reduction of a Petri Net. All the techniques mentioned above do not guarantee complete elimination of analysis. Most of them deal with local transformation. The Transformation by Structural matrix can be a Global Technique. Transitions and places can be deleted according to certain rules.

CHAPTER 5
STRUCTURAL MATRIX

This chapter discusses the structural relationships between PSP$_S$ recorded by Structural matrix (S_Matrix) , a brief procedure for finding S_Matrix , a detailed algorithm for finding S_Matrix and also an example of the application of the algorithm.

5.1. Structural Relationships Between PSPs

A Petri Net can be expanded by inserting new paths having some physical meaning such as increasing concurrency, alternatives etc as per the above discussed Knitting Technique. The resultant net will be an ordinary net with properties such as liveness, boundedness, safeness, and consistency. With the help of this technique many entities can be synthesized. This strategy takes advantage of the fact that the structural relationships between PSPs in a logically correct PN must satisfy some constraints. Structural matrix records relationships between PSPs in the PN. If the relationships among a set of PSPs violate some synthesis rules and which are complete, then the net has some undesired properties such as unboundedness, deadlocks. Hence S_Matrix is very useful for analysis.

An entry of Structural matrix, S_Matrix[i][j] corresponding to row i and column j, stands for the structural relationship between PSP$_i$ and PSP$_j$. It may be one of the following.

Sequential: PSP_i and PSP_j are sequential to each other during the iteration period.

Earlier (PSP_i SE PSP_j): If PSP_i is sequential earlier to PSP_j, then S_Matrix[i][j] = 'SE'; reversely, S_Matrix[j][i] = 'SL';

Later (PSP_i SL PSP_j): If PSP_i is sequential later to PSP_j, then S_Matrix[i][j] = 'SL'; reversely, S_Matrix[j][i] = 'SE'.

Previous (PSP_j SP PSP_j): If PSP_i is sequential previous to PSP_j , then S_Matrix[i][j] = 'SP'; reversely, S_matrix[j][i] = 'SN'.

Next (PSP_i SN PSP_j): If PSP_i is equal next to PSP_j, then S_Matrix[i][j] = 'SN"; reversely, S_Matrix[j][i] = 'SP'. Exclusive (PSP_i SX PSP_j) : if PSP_i and PSP_j are sequential exclusive to each other during iteration periods, then S_Matrix[i][j] = S_Matrix[j][i] = 'SX'.

Cyclic(PSP_i CL PSP_j): If PSP_i and PSP_j are cyclic to each other during iteration periods, then they are both earlier and later than each other. Then S_Matrix[i][j] = S_Matrix[j][i] = 'CL'.

Concurrent (PSP_i CN PSP_j): If PSP_i and PSP_j are concurrent to each other during the iterati∩n period, then S_Matrix[i][j] = S_Matrix[j][i] = 'CN'.

Exclusive (PSP_i EX PSP_j): If PSP_i and PSP_j are exclusive to each other during the iteration period, then S_MAtrix[i][j]= S_Matrix[j][i] = 'EX'.

5.2 Brief procedure for finding S_Matrix

Two search strategies such as depth-first or breadth-first are used to construct a S_Matrix for a Petri net. In Petri Net theory sequential means depth, and concurrent means breadth. In this paper "depth-first' search is adopted to find the S_Matrix of a Petri Net. There are five major steps in finding the S_Matrix of a Petri Net.

1) Generate a new PSP in depth-first search;

2) Enter the sequential relationships between this new PSP and other generated PSPs respectively;

3) Update the S_Matrix if there exists a cycle.

4) Find the structural relationships between PSPs from current considered PSPs and all other existing PSPs;

5) Check whether there exists sequential exclusive between PSPs. If so update the S_Matrix. The detailed algorithm is furnished below.

5.3 Detailed Algorithm for finding a S_Matrix

```
5.3.1 Unmark and uncheck all nodes in the given PN;
5.3.2 Select an input place of an enable transition in the initial marking as the home place;
5.3.3 Find the PSP containing this home place ;
Let PSP.number =0;
5.3.4 Let the joint of this PSP be the current node;
5.3.5 Repeat the following steps until every node is checked;
        5.3.5.1 If (the current node is a place ) then S = 'P';
                                                  else S = 'T';
        5.3.5.2 while (1)
              {
                     if (all output nodes of this current
node have been marked)
           {
                            check this node;
                            backtrack to the generation point of
                            a PSP whose joint is this current node;
```

```
                              let this generation point be the
    current node;
                                          }
                                          else
                                              break;
                                                  } /* while(1)*/
    5.3.5.3 let start_psp = PSP.number + 1;
    let farther_psp = the PSP whose joint is the current node;
5.3.5.4 pick one of the unmarked output nodes from this current node to find a
new PSP;
    mark this output node;
    let the current node is the generation point of this new PSP;
    /* Generate a new PSP */
    while (both the indegree & outdegree of this output node = 1)
    {
        insert this node into the new PSP;
        check this node;
        let the output node of this current output node be the new output
node;
        } /* while */
5.3.5.5  let this output node be the joint of this new PSP;
    let this output node be the current node ;
    let PSP.number++;
5.3.5.6  If (PSP.number == start_psp)
    {
        S_Matrix[farther_psp][ PSP.number] = 'SP';
        S_Matrix[PSP.number][father_psp] = 'SN';
        }
    else
        {
                S_Matrix[PSP.number-1][PSP.number] = 'SP';
                S_Matrix[PSP.number][PSP.number-1} = 'SN';
                                                    } /*if*/
        for (i = start_psp; i < PSP.number-1; i++)
                    {
                            S_Matrix[i][PSP.number] = 'SE ';
                            S_Matrix[PSP.number][i] = 'SL ';
                    } /*for*/
5.3.5.7 Let end_psp = PSP.number; m = 0;
        if (this output node was marked)
        {
                Let G_psp be a PSP whose joint is the generation
point of the start_psp;
                Let G1_psp be a PSP which has the same generation point as the
start_psp;
                Let J_psp be a PSP whose generation point is the joint of the end
_psp;
                let J1_psp be a PSP which has the same joint as the end_psp;
        5.3.5.7.1 if (S_Matrix [J1_psp][G1_psp] = ('SE' | 'SP'))
                    {
                            for (i= start_psp; i < end_psp; i++)
                            {
                            for ( j= start_psp; j<end_psp; j++)
                if (i != j)
                                        {
                   S_Matrix[i][j] = 'CL';
                          S_Matrix[j][i] = 'CL';
                }   /*  if and for (j) */
              for (k=o; k<start_psp; k++)
               {
               if (S_Matrix[k][G_psp] = ('SL'| 'SN '))
                {
                   if (S== ' T ')
   S_Matrix[i][k]=S_Matrix[k][i]= 'CN ';
 else
   S_Matrix[i][k]=S_Matrix[k][i]= 'EX ';
               }
 else
  S_Matrix[i][k]=S_Matrix[k][i]=S_Matrix[k][G1_psp]
}  /* for (k) */
} /* for (i) */
if (S_Matrix[j1_psp][k] =( 'SE '|'SP ') &&
   S_Matrix[G1_psp][k] =( 'SL' |'SN '))
  /* start_psp is cyclic to end_psp , so need updating
     the previous relationship between them to 'CL ' */
  for (j=start_psp; j<end_psp; j++)
  {
   S_Matrix[k][j] = 'CL ';
   S_Matrix[j][k] = 'CL';
   D[m++] = j; /* matrix index, used for updating previous
relationship of 'SE','SP','SL','SN' between start_psp and end_psp to 'CL'
                         */
   for (g=o; g<m-1; g++)
       {
     if (S_Matrix[D[g]][D[m-1]]=('SE'|'SP'|'SL'|'SN')
    {
       S_Matrix[D[g]][D[m-1]]= 'CL';
              S-Matrix[D[m-1]][D[g]]='CL';
      } /* if  */
     }  /* for (g) */
    } /* for (j) */
  }
5.3.5.7.2 /* Find the structure relationships between PSPs from
         start_psp to end_psp and all other existing PSPs
    */
    else  /* 5.7.1. if */
    {
     for (i=o; i<start_psp; i++)
      {
        for (j=start_psp; j<end_psp; j++
         {
           if (S_Matrix[i][G_psp]= ('SL' | 'SN '))
            {            if (S_Matrix[j1_psp][i]!=('SE' |'SP'))
           {
                if (S== 'T ')
S_Matrix[j][i]=S_Matrix[i][j]='CN,;
        else
S_Matrix[j][i]=S_Matrix[i][j]='EX';
}
else
 {
S_Matrix[i][j]=S_Matrix[i][j1_psp];
S_Matrix[j][i]=S_Matrix[j1_psp][i];
     }
    }
    else
     if(S_Matrix[i][G1_psp]=='CL')
      {
       if (S=='T')
         S_Matrix[j][i]=S_Matrix[i][j]='CN ';
        else
         S_Matrix[j][i]=S_Matrix[i][j]='EX ';
      }
     else
      {
         S_Matrix[i][j]=S_Matrix[i][G1_psp];
         S_Matrix[j][i]=S_Matrix[G1_psp][i];
      }
    if (i is an input PSP of j)
 {
       S_Matrix[i][j]= 'SP ';
       S_Matrix[j][i]= 'SN ';
```

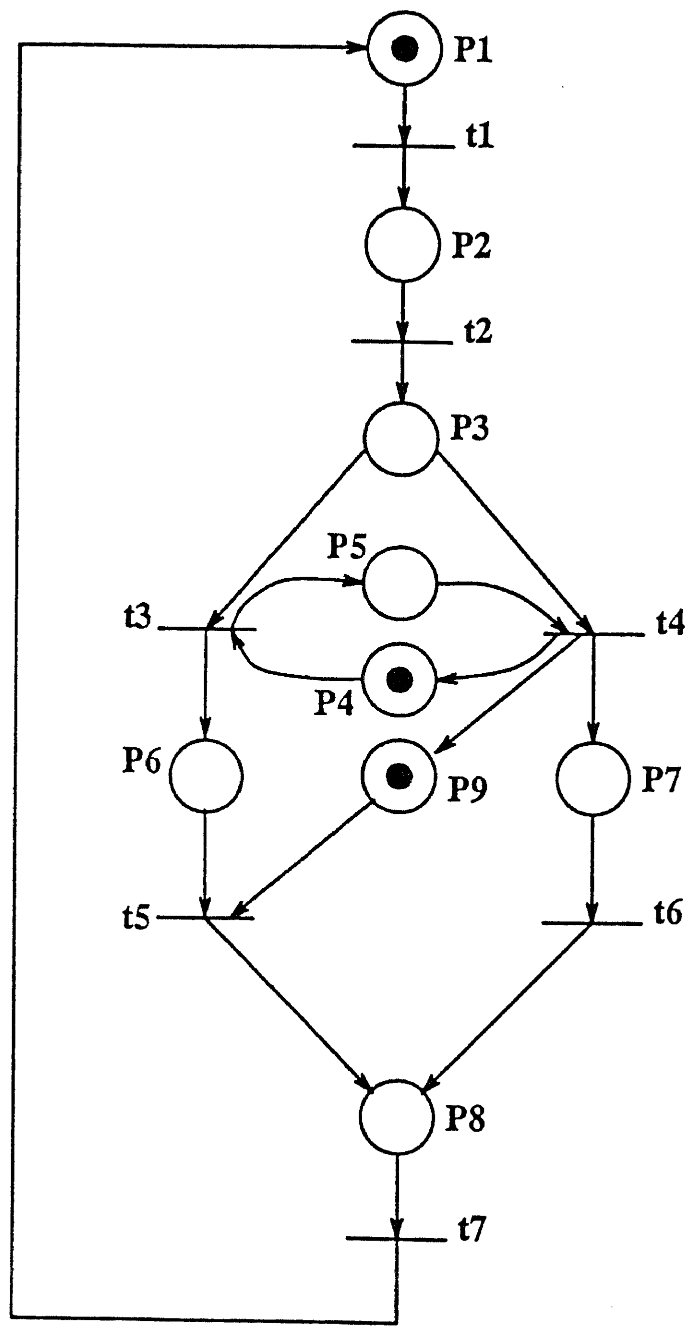


Fig. 5.1 An Example of a Petri Net

	0	1	2	3	4	5	6	7	8
0		SP	SE	SE	SE	SE	SE	SE	SE
1	SN		SP	SE	SP	SX	SX	SE	SX
2	SL	SN		SP	CN	SX	SX	CN	SX
3	SL	SL	SN		CN	SN	SX	CN	SL
4	SL	SN	CN	CN		CN	CN	CL	SL
5	SL	SX	SX	SP	CN		CN	CN	SN
6	SL	SX	SX	SX	CN	CN		CN	SN
7	SL	SL	CN	CN	CL	CN	CN		SN
8	SL	SX	SX	SE	SE	SP	SP	SP	

Fig. 5.2 The S_Matrix for the PN in Fig. 5.1

```
    }
   else
     if (i is an output PSP of j)
      {
      S_Matrix[i][j] = 'SN ';
      S_Matrix[j][i] = 'SP ';
      }
    if (i is cyclic to a G1_psp and a J1_psp)
        S_Matrix[i][j] = S_Matrix[j][i] = 'CL ';
   } /* for (j) */
  } /* for (i) */
}
```

5.4 Application of the Algorithm

Consider Fig. 5.1. The steps for finding the S_Matrix are presented below. The corresponding S_Matrix is shown in Fig. 5.2.

1. In the initial marking, t1 is enabled, hence pick P1 as the home place. The PSP containing p_1 is PSP_0 = [p_8 t_7 p_1 t_1 p_2 t_2 p_3]. Mark t_7 and check p_8 since t_7 is the only output node of p_8. The current node is p_3.

2. t_3 and t_4 are two output nodes of p_3. Pick t_3. t_3 is not a simple node, hence the new PSP_1 = [p_3 t_3] and S_Matrix[0][1] = 'SP', S_Matrix[1][0] = 'SN'. Mark t_3.

3. p_5 and p_6 are two output nodes of t_3. Pick p_6, which is a simple node and it's output node t_5 which is not simple. Hence the new PSP_2 = [t_3 P_6 t_5], and S_Matrix[0][2] = 'SE', S_Matrix[2][0] = 'SL', S_Matrix[1][2] = 'SP', S_Matrix[2][1] = 'SN'. Mark p_6.

4. The output node p_8 of t_5 is not simple, hence the new PSP_3 = [t_5 p_8]. Mark p_8 and check t_5. Since p_8 has been checked, backtrack to t_5 which has been checked. So backtrack to t_3 and pick its unmarked output node p_5 which is a simple node. Its output node t_4 is not a simple node, hence the new PSP_4 = [t_3 p_5 t_4], and S_Matrix[2][3] = 'SP', S_Matrix[4][1] = 'SN', (PSP_0, PSP_1) -> PSP_3, (PSP_2, PSP_3) | | PSP_4. Check t_3 since all it's output arcs have been traced.

5. p_9, p_7 and p_4 are three output node of t4. Pick p_9 and find a new PSP_5 = [t_4 p_9 t_5]. S_Matrix[4][5] = 'SP', and S_Matrix[5][3] = 'SP', (PSP_4 , PSP_5) | | PSP_2, PSP_1 -> (PSP_4, PSP_5) and PSP_0 -> (PSP_4, PSP_5). Mark p_9.

6. Since t_5 has been checked, backtrack to t_4 and pick an unmarked output node p_7, find a new PSP_6 = [t_4 p_7 t_6 p_8]. PSP_6 | | (PSP_0,PSP_1, PSP_4) -> PSP_6. Mark p_7.

7. Since t_8 has been checked, backtrack to t_4 again and pick the remaining unmarked output node p_4 and find a new PSP_7 = [t_4 p_4 t_3] which is cyclic to PSP_4. Hence PSP_4 o PSP_7, (PSP_0, PSP_1) -> PSP_7 and (PSP_2, PSP_4) | | PSP_7. Update S_Matrix[4][5] = S_Matrix[4][5] = S_Matrix[5][4] = 'CN'. Mark p_4 and check t_4.

8. Since t_3 has been checked, backtrack to p_4. Pick the remained untraced path p_3->t_4 and find a new PSP_8 = [p_3 t_4]. Since t_4 has been marked and is on a cycle, PSP_8 SX (PSP_1,PSP_2), PSP_8 | | (PSP_5, PSP_6, PSP3) and PSP_1 SX (PSP_5, PSP_6). Both transactions t_3 and t_4 on this cycle are the joints of the VPs with a CGP p_3. Y_1 = { PSP_2 , PSP_3 } and Y_2 = {PSP_3, PSP_5, PSP_6 }. PSP_2 is 'SX' to PSP_5 and PSP_6, and PSP_3 is 'SX' to PSP_6. Check p_3.

9. All PSPs are found and stop.

CHAPTER 6

A NEW APPROACH TO REDUCTION OF PETRI NETS

Knitting Technique for Synthesis rules described in Chapter 3 is successfully applied to Reduction theory of Petri Nets. Path generations previously forbidden are now allowed without incurring logical incorrectness. For illustration in Fig. 9, a TT-path generation (PSP_4) between two exclusive transitions(t_3 and t_4) was previously not allowed. But the new rules allow additional path generations (PSP_7) such that the two exclusive transitions become synchronized and hence sequentially exclusive to each other. Further it is also found that TP and PT generations, which were not allowed before, may be allowed however by adding new paths. Each TP(PT) generation must be associated with a new PT(TP) generation to avoid unboundedness (deadlocks). Each new generation must be synchronized with existing PT paths with the same generation point as the new PT path. TP rule and PT rule, the names given to the above rules, enable us to find out that any PNs synthesized using the previous rules in [8] have synchronic distance 1 or infinity between any two transitions. In accordance with the new technique it is possible to synthesize PNs with arbitrary synchronic distance between any two transitions. Further it is possible to allow all generations which were hitherto forbidden. Thus the new technique called Knitting Technique(synthesis technique) is more powerful than the traditional techniques. In addition to the above process another process called reduction, which is a reverse procedure to synthesis, is more efficient and useful than the conventional techniques. The S_Matrix is useful in finding any mismatch on account of existence of some paths which do not conform to synthesis rules.

Hence S_MAtrix can be used to perform reduction on
any Petri Net. By iterative adoption of the above technique petri nets can be simplified to the maximum level possible. The algorithm for the above discussed technique is furnished below.

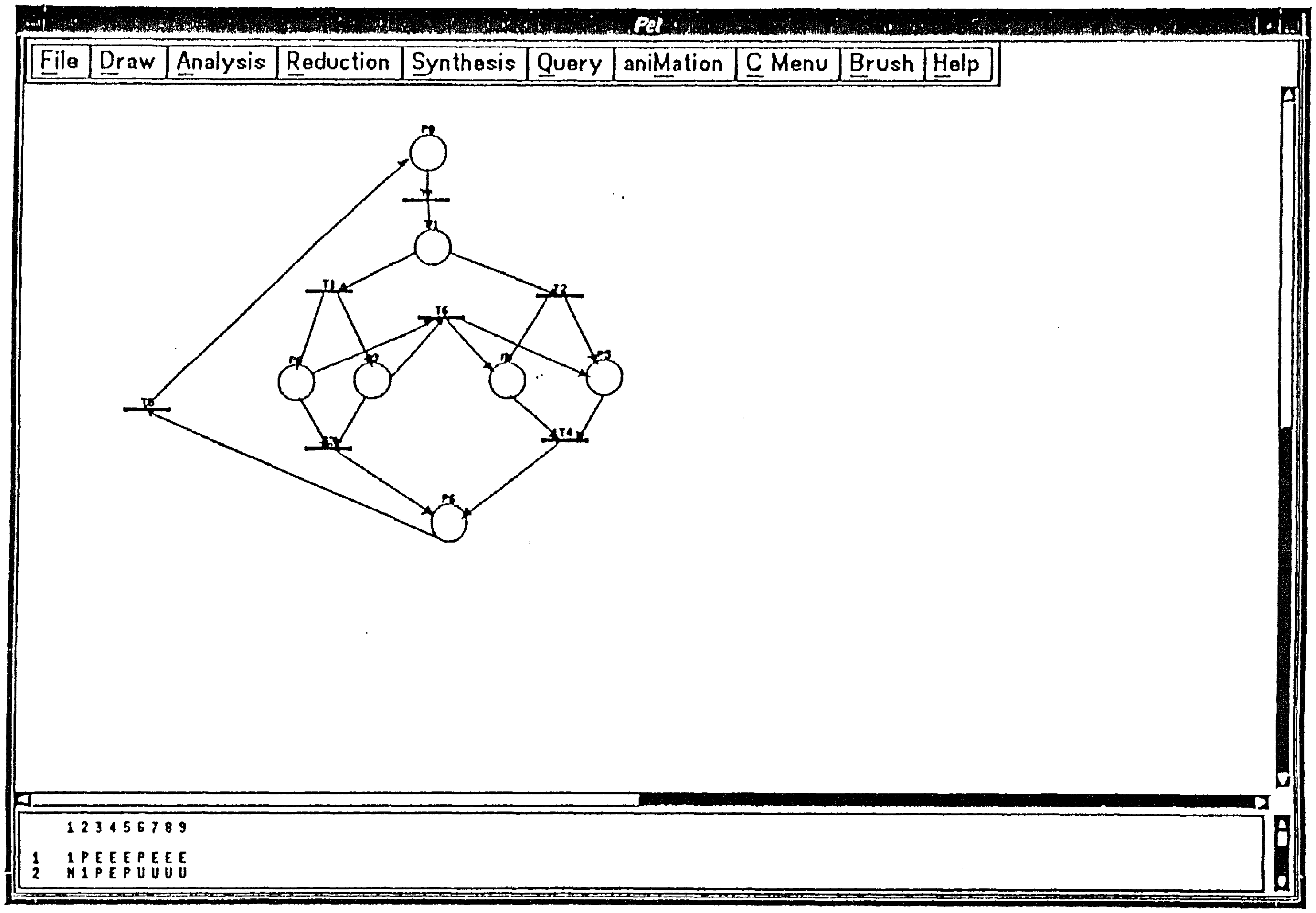
6.1 An Algorithm for Reduction of Petri Nets

Let GP be a PSP whose joint is the generation point of the start_psp;
JP be a PSP whose generation point is the joint of the end _psp;
PSPc the current PSP under consideration and [X] the cardinality of set X;
At the beginning of the algorithm, set flag = TRUE;
Repeat the following steps (1) to (4) indefinitely until flag = FALSE or errors of designs detected;

```
6.1.1 If flag = TRUE and (path,rel,Q1,Q2)
=(TTP,SQ,LEX(GP,JP),LEX(JP,GP))|
(PPP,SQ,LCN(GP,JP),LCN(JP,GP)) |(TTP,CN,LEX(GP,JP),LEX(JP,GP
(PPP,EX,LCN(GP,JP),LCN(JP,GP)),
/* where rel is the structural relationship between GP and JP, and
|Q1| = |Q2| = 1 (i.e., each set contains only one member) */
6.1.1.1 If path=PPP, then delete the PSPc; else flag=FALSE;
6.1.1.2 If (path,rel) = (TTP,CN) and there does not exist a PSP such that
PSP <- GP and JP -> PSP, then
signal "TT.3 rule violation, unbounded places in the PSP" and exit;
else delete the PSPc and flag=TRUE;
6.1.1.3 If (path,rel) = TTP,SE) and PSPc does not have tokens, then
    /* PSPc is in a cycle */
6.1.1.3.a fire all enabled transitions in PSPs which satisfy
    JP->PSP->GP until all of them are no longer enabled;
```

```
6.1.1.3.b If the generation point of PSPc has fired more than once,
delete PSPc and flag1=TRUE;
  else signal "TT.2 rule violation, deadlock at the generation
  point of PSPc" and exit;
6.1.1.4 If (path,rel) = (TTP,EX), then
     signal "exclusive TT interaction, potential deadlock at the joint
  and unbounded input place of the joint of the PSPc" and
6.1.2 If (path,rel) = (PPP,CN), then
     signal "PP.3 rule violation, potential deadlock at a PSP <- GP and
   PSP <- JP" and exit;
6.1.3 For all VPs (VP1, VP2, ..., VPk) whose common generation point is a place
   /* There is only one GP and k JPs (JP1,JP2,... JPk) */
6.1 3.1 If LEX(JP1,GP) = {JP1,JP2,...,JPk},
     else break;
6.1.3.2  A ={};  /* create an empty set A */
6.1.3.3  If the generation point of GP is a place P, then /* get */
6.1.3.3.a for each input PSP of P
     If its generation point is a transition and which has more
    than one output PSP, then
    insert one of the input PSP of this transition in to A;
   else it is a place, make this place be the new P;
    go to step 3.3.a.;
6.1.3.3.b let pa be a member PSP in A;
    If JP1 -> pa and none of the PSPs,both of whose generation
point and joint have been traced, has tokens, then
perform a similar steps to 1.3.a. & 1.3.b. to ensure
no TT.2 rule violation;
```

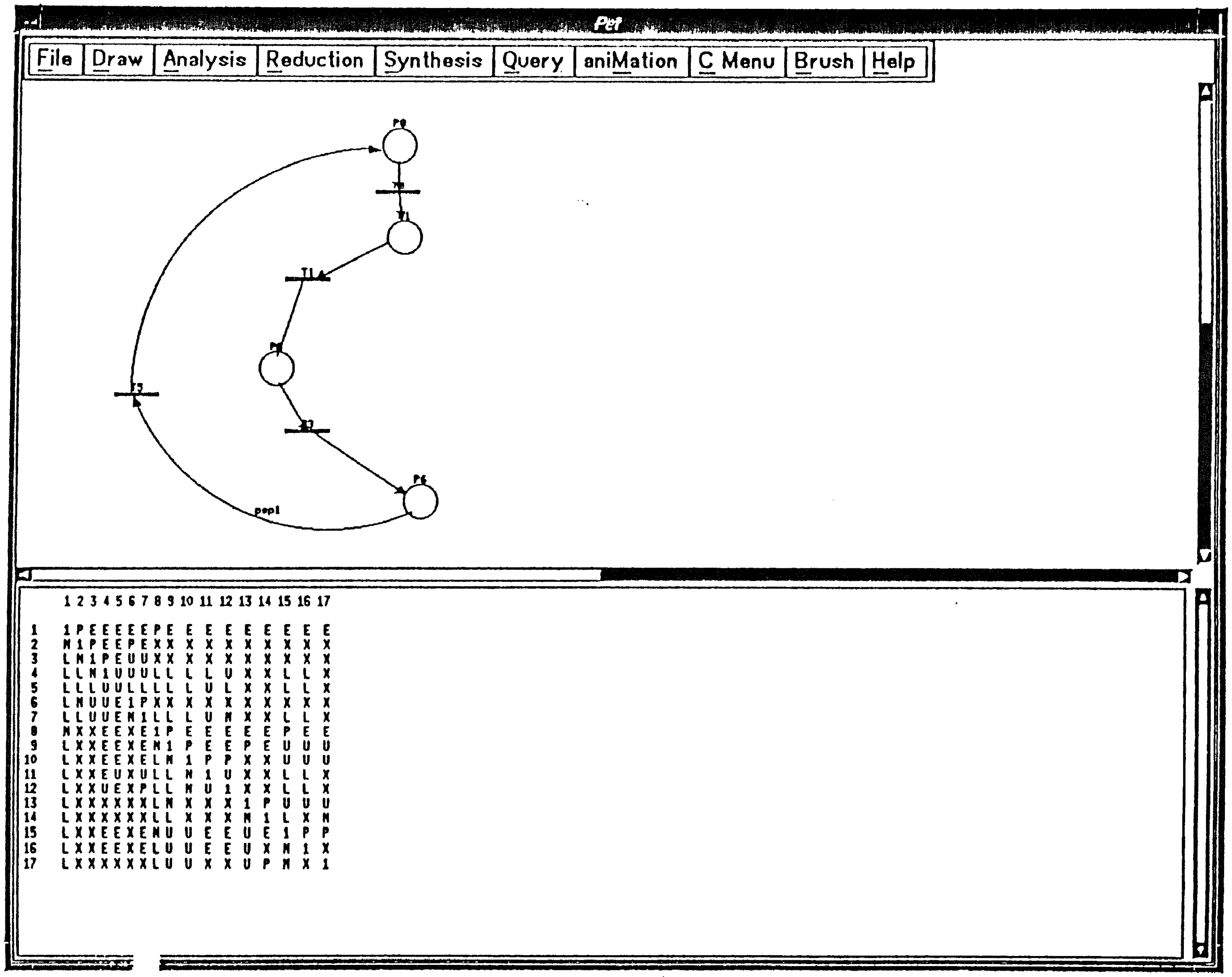
```
6.1.3.3.c else if JP1 || pa, then
    perform a similar step to 1.2. to ensure no TT.3 rule violation;
6.1.3.3.d else if JP1 | pa, then
    perform a similar step to 1.4. to signal "exclusive TT
    interaction, potential deadlock at the joints and unbounded
    input places of the joints of JPs" and exit;
6.1.3.3.e  else if A = LEX(pa,JP1),then
    delete all VP1,... VPk, and all PSPs, both of whose generation
    point and joint have been traced, and flag=FALSE;
6.1.4 For all VPs (VP1, VP2,...,VPk) whose common joint is a transition
  /* There is only one JP and k GPs (GP1,GP2,...GPk)*/
6.1.4.1 If GP1|| JP, then
   signal "PP.3 rule violation, potential deadlock at a PSP <- GP and
PSP <-JP" and exit;
6.1.4.2 If LCN(GP1,JP) = { GP1, GP2,..., GPk},
   else break;
6.1.4.3  B = {};  /* create an empty set B */
6.1.4.4  If the joint of JP is a transition T, then /* get B*/
6.1.4.4.a for each output PSP of T,
    if its joint is a place and which has more than one input PSP,
    insert one of the input PSP of this place in to B;
  else it is a transition, make this transition the new T;
     go to step 4.4.a.;
6.1.4.4.b let pb be a member PSP in B;
   If B = LCN(pb,GP1),then
     delete all VP1,... VPk, and all PSPs, both of whose
     generation point and joint have been traced, and flag=FALSE;
```



```
    1 2 3 4 5 6 7 8 9

1   1 P E E E P E E E
2   N 1 P E P U U U U
```

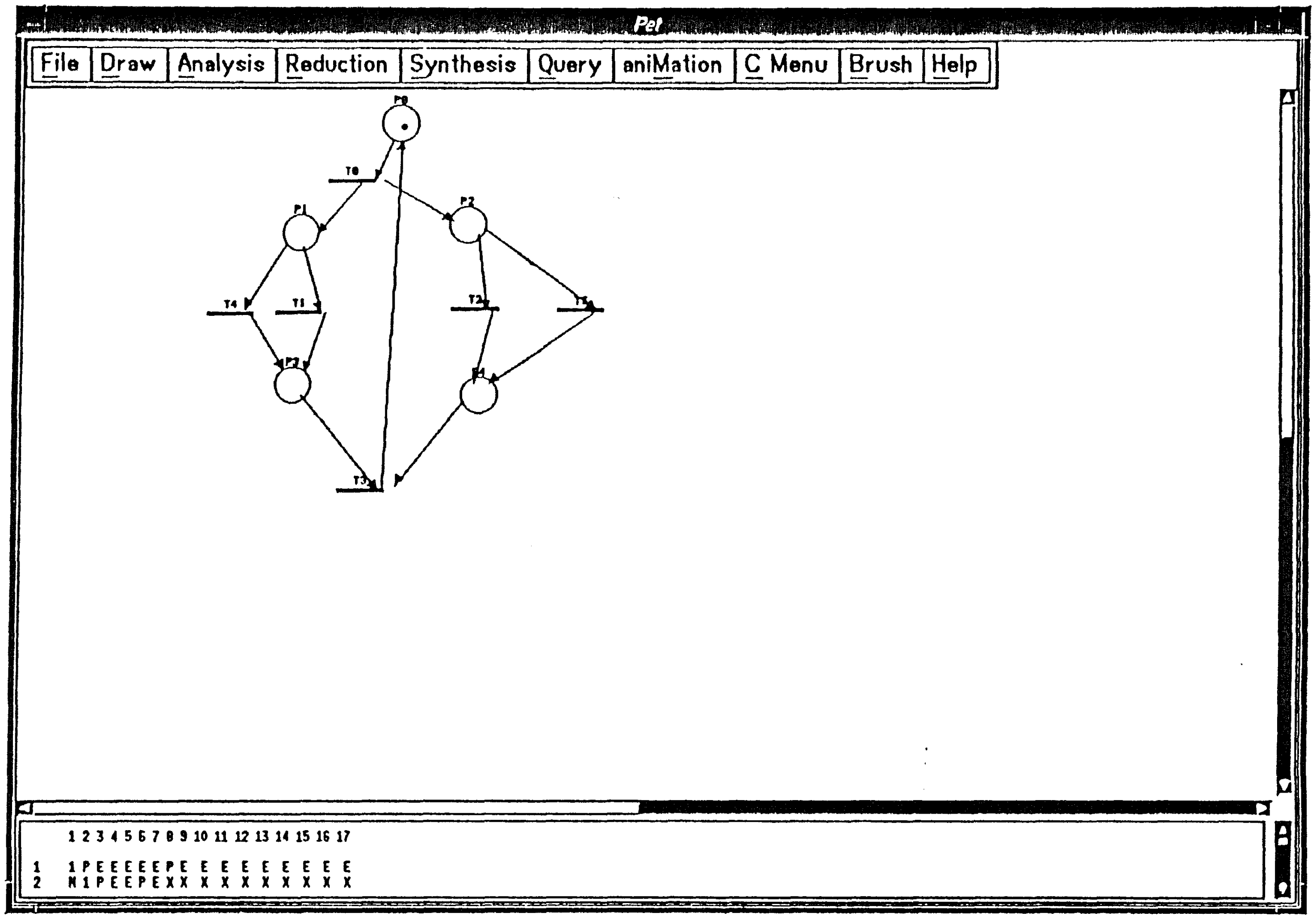
Fig. 6.1 An Example for Petri Net Reduction



```
      1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

 1    1 P E E E E E P E  E  E  E  E  E  E  E  E
 2    N 1 P E E P E X X  X  X  X  X  X  X  X  X
 3    L N 1 P E U U X X  X  X  X  X  X  X  X  X
 4    L L N 1 U U U L L  L  L  U  X  X  L  L  X
 5    L L L U U L L L L  L  U  L  X  X  L  L  X
 6    L N U U E 1 P X X  X  X  X  X  X  X  X  X
 7    L L U U E N 1 L L  L  U  N  X  X  L  L  X
 8    N X X E E X E 1 P  E  E  E  E  E  P  E  E
 9    L X X E E X E N 1  P  E  E  P  E  U  U  U
10    L X X E E X E L N  1  P  P  X  X  U  U  U
11    L X X E U X U L L  N  1  U  X  X  L  L  X
12    L X X U E X P L L  N  U  1  X  X  L  L  X
13    L X X X X X X L N  X  X  X  1  P  U  U  U
14    L X X X X X X L L  X  X  X  N  1  L  X  N
15    L X X E E X E N U  U  E  E  U  E  1  P  P
16    L X X E E X E L U  U  E  E  U  X  N  1  X
17    L X X X X X X L U  U  X  X  U  P  N  X  1
```

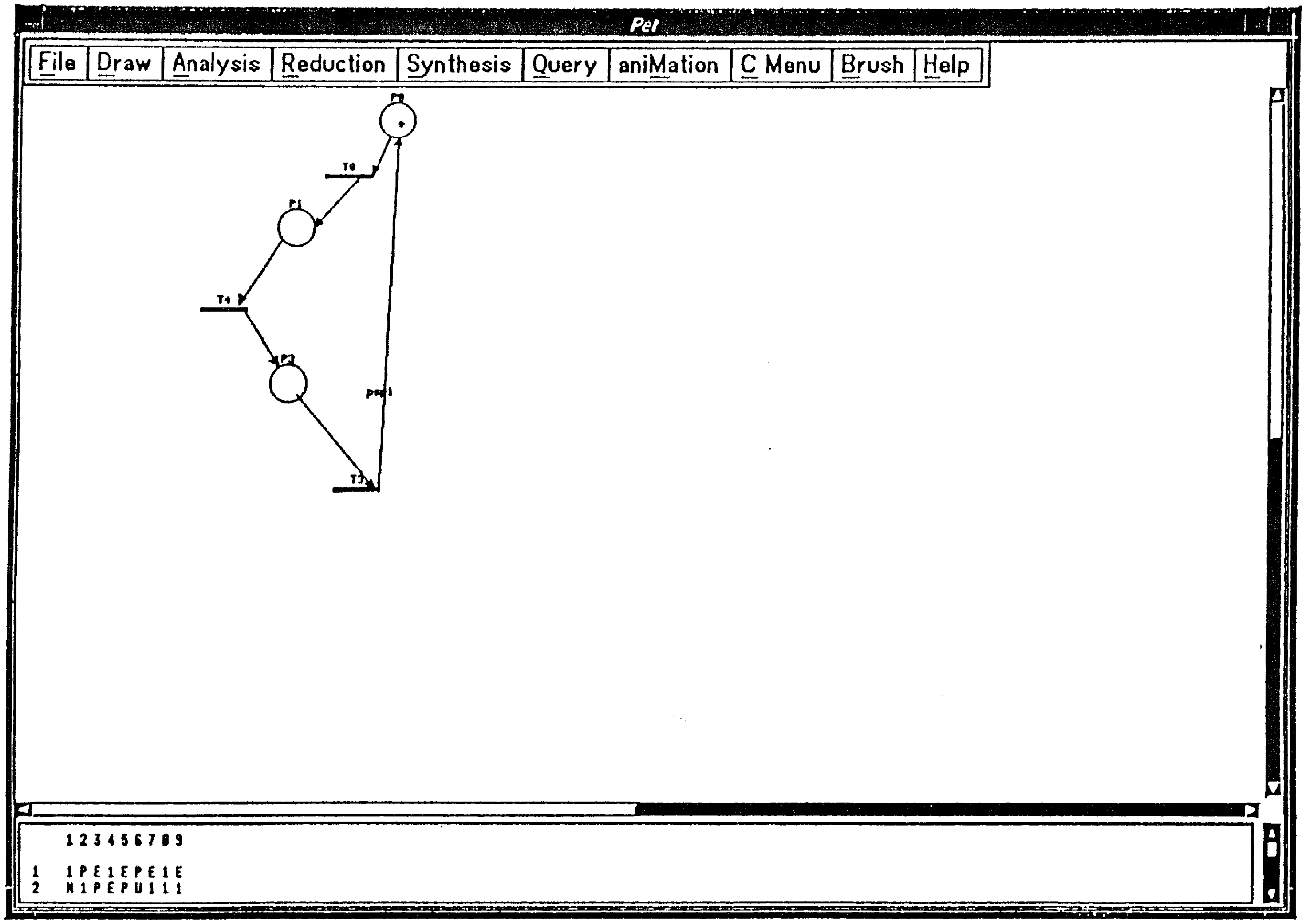
Fig. 6.2 A Reduced Net for Petri Net in Fig. 6.1



```
     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

1    1 P E E E E E P E  E  E  E  E  E  E  E  E
2    N 1 P E E P E X X  X  X  X  X  X  X  X  X
```

Fig. 6.3 An Example for Petri Net Reduction



```
    1 2 3 4 5 6 7 8 9

1   1 P E 1 E P E 1 E
2   N 1 P E P U 1 1 1
```

Fig. 6.4 A Reduced Net for Petri Net in Fig. 6.3

6.2 An Illustration of Reduction

The reduction was implemented with the help of a CAD tool which can facilitate analysis of Petri Nets. The reduction as well as Structural matrix algorithms were implemented in C language in X-Window/Motif environment. The interactive CAD tool with which reduction of Petri Net was accomplished is discussed below.

The tool animates firing of transitions and analyzes the theory and tests their correctness. The "FILE" button allows files to be imported, copied, deleted and edited. The reduction button reduces the net by using knitting technique. The "DRAW" button helps us in selecting a function such as clear, print, undo, draw arcs, arrows, places etc. In addition to reduction the tool can be used for synthesis also. The figures of the petri nets which were reduced using the above tool are furnished in Fig.6.1 to Fig.6.4

CHAPTER 7
CONCLUSIONS

Conventional reduction techniques such as·six reduction rules discussed in this paper focuss on reducing a subnet or a partial PN to a single transition as such they are not powerful enough to reduce a Petri Net to a simplest possible net. This paper proposes an iterative reduction technique called Knitting Technique which is more efficient than the previous techniques. An algorithm for implementing the new reduction technique is also presented. An algorithm to represent structural relationship of any given Petri Net is also presented. The proposed iterative reduction technique by S_Matrix is a global technique and hence is not a transformation technique. It deletes transitions and places according to certain rules. It has the potential of completely substituting for analysis. The influence of the proposal is as mentioned below.

1) Advance the Petri Net reduction theory by introducing (a) Automation of reduction and

(b) provision of more powerful iterative reduction technique than the conventional ones.

2) The technique may possibly eliminate the reachability analysis.

REFERENCES

[1] Murata, T., "Petri Nets: Properties, Analysis and applications," IEEE Proceedings, Vol. 77,No. 4, April 1989, pp. 541-580.

[2] Peterson, J.L., Petri Net Theory and the Modeling of Systems, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1981.

[3] Murata, T., "Modeling and Analysis of Concurrent Systems," in Handbook of Software Engineering , edited by C.Vick and C.V. Ramamoorthy, Van Nostrand Reinhold, 1984, pp.39-63.

[4] Ramamoorthy, C. V., and So, H., "Software Requirements and Specifications: Status and Perspectives," IEEE Tutorial: Software Methodology, 1978.

[5] Yau, S.S., and Caglayan, M.U., "Distributed Software System Design Representation Using Modified Petri Nets," IEEE Trans. on Software Engineering, Vol. SE-9, No. 6, Nov.1983, pp. 733-745.

[6] Agerwala, T. and Choed-Amphai, Y., "A Synthesis Rule for Concurrent Systems," Proc. of Design Automation Conference, pp. 305-311,1978.

[7] Koh, I., and DiCesare, F., "Transformation Methods for Generalized Petri Nets and Their Applications to Flexible Manufacturing Systems," The 2nd International Conference on Computer Integrated Manufacturing, Troy, NY, MAY 1990, pp. 364-371.

[8] Yaw, Y., "Analysis and Synthesis of Distributed Systems and Protocols," Ph.D. Dissertation, Dept. of EECS, U.C. Berkely, 1987.

[10] Yaw, Y., and Foun, F.L., "The Algorithm of A Synthesis Technique for Concurrent Systems," 3rd International Petri Net Conf., Dec. 1989, pp. 266-276.

[11] C.V. Ramamoorthy, Y. Yaw, and W.T. Tsai, "A Petri Net Reduction Algorithm for Protocol Analysis," Computer Communication Review (USA), Vol. 16, No.3 Aug. 1986, pp. 157-166.

[12] I.Suzuki and T. Murata, "A Method of Stepwise Refinement and Abstraction of Petri Nets," Journal of Computer and System Sciences 27, 1983,pp 51-76.

[13] R. Valette, "Analysis of Petri Nets by Stepwise Refinement, "Journal of Computer and System Sciences 18, 1979, pp. 35-46.

[14] G. Berthelot, "Transformations and Decompositions of Nets," LNCS, Advances in Petri Nets, pp. 359-376, Part I, 1986.

[15] Hyung, Lee-Kwang, "Generelized Petri Net Reduction Method" IEEE Trans. Syst., Man, Cybern., Vol. SMC-17, No.2, 1987, pp. 297-303.

[16] Ramamoorthy, C. V., Dong, S.T., and Usuda, Y., "The Implementation of an Automated Protocol Synthesizer (APS) and Its Application to the X.21 Protocol," IEEE Trans. on Software Engineering, pp.886-908, No.9, 1985.