A Study of Petri Nets

Modeling, Analysis and Simulation

Project done as part of summer training
Under the guidance of

Dr. A. Venkateswarlu
Dy. Division Head
Control Dynamics and Analysis Division
Control Systems Group
ISRO Satellite Centre
Bangalore – 560 017

submitted in the requirement for the partial fulfillment of Dual Degree in
Aerospace Engineering

by

Abhishek Halder (03AE3009)
Undergraduate Student

Department of Aerospace Engineering
Indian Institute of Technology Kharagpur
Kharagpur – 721302
West Bengal, INDIA
August 2006

Contents

Page No.

Acknowledgement

The information presented in this report is an outgrowth of surveying many journals and books on Petri net, which I consulted at the ISAC library. I would like to take this opportunity to thank Dr. A. Venkateswarlu, Dy. Head, Control Dynamics and Analysis Division, Control Systems Group, for guiding me through the period of two months during my work at ISAC. I greatly appreciate the kind support, thought provoking discussions and suggestions provided by him. I am grateful to him for introducing me to the world of Petri net.

I am deeply indebted to A. Indra, Head, Advanced Systems Development Section, CSG, for the kind mentorship provided by her. She holds the credit for clarifying many intricacies of Petri net, providing some excellent references and arranging many working facilities for me, which would otherwise be difficult to get.

I would like to acknowledge Mr. Jasvinder Singh Khoral, Scientist/Engineer 'SF', CSG for kindly agreeing to accompany on Sunday for finishing the documentation of this project, Dr. V.K. Agrawal, Group Director, CSG for allowing me to work in this group and last but not the least, Dr. N.V. Vighnesam, Head, Orbit Dynamics and Determination Section (ODDS), without his effort, it would be impossible to work in ISAC. Dr. Manoranjan Sinha at the Department of Aerospace Engineering, IIT Kharagpur, took great endeavor to arrange the permission for my summer training at ISAC. Thanks to him.

Chapter 0

Motivation

"A complex unity formed of many often diverse parts to a common plan or serving a common purpose."

- Definition of 'system'
Webster Dictionary.

The above quoted definition captures the current trend and future direction of system engineering, as mere static description of constituent units no longer describes a 'system'. Today's complex system involves multiple communicating units. Understanding such a system demands a closer look at the dynamic interaction between the constituent entities and their emergent behavior.

Modern systems are no more continuous or discrete alone, rather a combination of them where some subsystems are in continuous domain, some in discrete domain and some are event-driven systems. Such a system of multifarious nature is often referred as a *hybrid system*. In particular, modeling and analysis of event-driven systems are beyond the scope of conventional system theory and need special treatment for suitable representation and analysis of such a system, named as *discrete event dynamic system* (DEDS).

Dynamics of DEDS is often event-based, asynchronous and concurrent in nature. While event-driven nature is itself difficult to handle, asynchrony and concurrency poses further challenge for modeling and analysis. Real life events take variable amount of time. Thus time, from a philosophical perspective, only defines a partial ordering of events. Hence any representation of DEDS must be able to represent asynchronous behavior of the system. Concurrency, on the other hand, requires a tractable and flexible theory, as there exist many conflicting views of concurrency. Some early models of concurrency (like interleaving model) viewed system's behavior as partially ordered sequence of events over a period of time. This, however, addresses concurrency in an indirect manner by introducing the concept of "pseudo-concurrency". So there is a need of more robust representation of true concurrency.

Chapter 1

Introduction

In the engineering discipline, system evolution has invariably been facing three major needs.

(1)The need to develop increasingly complex systems.

(2)The need to assess the system's operational risks.

(3)The need to have a cost competitive solution to attain these requirements.

Due to time and money constraints, it is no longer feasible to follow the design cycle of trial and error prototyping. Instead, the industry is leaning more towards simulation, so that the design flaws can be worked out even before the prototype is built.

It's here Petri Net comes in. Petri net is a net-based abstraction, which can be used as a modeling tool (graphical and mathematical), as a simulation tool and as an analysis tool.

As a modeling tool, it helps in system design. The graphical nature aids system visualization, the mathematical nature captures system behavior.

As a simulation tool, it enables one to identify design errors. Extensive simulations may detect errors, which are rare and elusive.

As an analysis tool it reveals various properties of the model and hence of the actual physical system. Thus, one can draw important conclusions about the system without going for experimentation or performing lengthy calculation of conventional system modeling.

Chapter 2

Basic Concepts and Definitions

2.1 Informal Introduction to Petri Nets

Any system consists of a number of activities and the system can be modeled by listing the states of the system, before and after those activities. An activity brings the system from one state to another i.e. activity causes state-transition. All such state-transitions, when graphically represented, are called state-transition diagram.

Example 2.1:

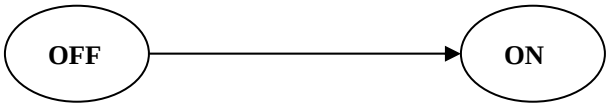


Figure 2.1 State-transition diagram for OFF-ON transition

The above state-transition diagram shows that the system undergoes a transition from OFF state to ON state. The activity, in this case, can be pressing of a switch.

A closer look at Fig. 2.1 reveals that a state-transition diagram is a directed graph composed of two elements: nodes (representing state of the system) and arcs (representing the direction of state-transition).

Pictorially nodes are represented by circles. Arcs are of two types: input arcs and output arcs. In Fig. 2.1, there are two nodes, representing OFF and ON state. The only arc in Fig. 2.1 is the output arc with respect to OFF node and input arc with respect to ON node.

The state-transition representation, as shown above, has some serious limitations. As one can see in Fig. 2.1, there is no representation of the activity itself. Also, there is little or no scope to represent the condition (if any) for which the transition occurs (say, if the temperature is less than 40^{o}C then the switch is pressed to make the system move from OFF to ON state). In addition to that, one has to define first the system's global state and then enumerate all the states and feasible events at each state. A possible consequence is the state-explosion problem for complex systems.

A formalism that can overcome some of these limitations is Petri nets. It gives more modeling flexibility by introducing two kinds of nodes; one (called *places*) to represent the states and/or conditions and the other (called *transitions*) to represent the activities. It uses local states rather than global states, thereby avoiding the state enumeration problems in the modeling stage. It can explicitly represent precedence relations, conflicting situations, synchronization concepts, concurrent operations and mutually exclusive events.

Example 2.2:

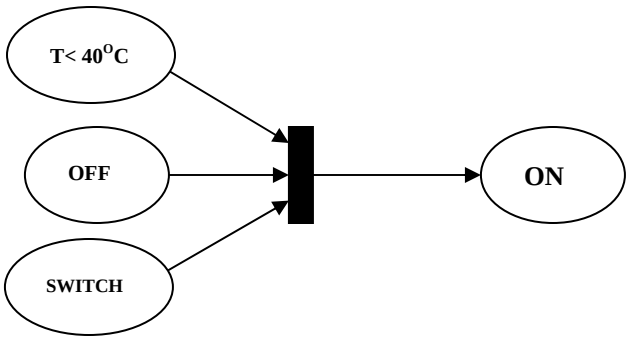


Figure 2.2 Petri net graph for OFF-ON transition

The Petri net graph of Fig. 2.2 shows that when the system is in OFF state *and* when temperature is less than 40^{o}C *and* when the switch is pressed then the system makes a transition from OFF state to ON state. A comparison of Fig. 2.1 and Fig. 2.2 shows that how Petri net graph provides more modeling power and flexibility over state-transition diagram.

A closer look at Fig. 2.2 reveals that a Petri net is a directed graph composed of two elements: nodes (places and transitions) and arcs (input and output).

Pictorially places are represented by circles and transitions by bars. Arcs are labeled with their weights (positive integers) – labels for unity weight are usually omitted.

The nodes and arcs constitute the static structure of Petri net. The dynamic behavior of the net is given by 'token game', representing various states of the system. A particular state is a snapshot of the system's behavior. The state of a place is called its *marking,* represented by the presence (condition holds) or absence (condition does not hold) of black dots, called *tokens*, in the circle representing the place. Current state of the modeled system (marking of the system) is given by the number and type (if tokens are distinguishable) of tokens in each place.

While places and arcs are passive components of the net, transitions are active components. When all input places and no output places of a transition contain tokens, then a transition fires. *Firing of a transition* removes tokens from all of its input places and puts tokens in its output places. Thus, token-flow occurs via the firing of transitions. The system achieves a new marking via the firing of a transition. Introduction of tokens into places and their flow through transitions enable one to describe the discrete-event dynamics of the PN and thereby of the modeled system.

2.2 Formal Introduction to Petri Nets

Before giving formal definition of Petri net, two important points may be noted:

(1) Some people do make a distinction between Petri net graphs (the graphical representation) and Petri net structure (the mathematical structure). No such distinction is made in this report.

(2) The Petri net was originally defined in a way that token-carrying capacity of each place was one. A later extension of Petri nets, called *Place-Transition nets (PT nets)* allowed multiple tokens at a place. In this report, all subsequent theories are given for PT nets, which are more general. Henceforth, the terms PT nets and ordinary Petri nets are interchangeably used.

2.2.1 Definition of Petri net

An *Petri net*, 'N' is a bipartite, weighted, directed multigraph, mathematically represented by a four-tuple $N = (P, T, I, O)$ where

$P = \{p_1, p_2, ..., p_i, ..., p_n\}$ is a finite set of places,
$T = \{t_1, t_2, ..., t_j, ..., t_m\}$ is a finite set of transitions,
} Node definition } $P \cap T = \emptyset$, $P \cup T \neq \emptyset$

$I : (P \times T) \rightarrow \mathcal{N}_o^+$
$O : (P \times T) \rightarrow \mathcal{N}_o^+$
} Arc definition } $\mathcal{N}_o^+ = \{0,1,2,\ldots\}$

The node definition says that the set of places and the set of transitions are disjoint (having no common elements) and there exists at least one node ($x \in P \cup T$) in the net. The set of arcs (F) defines two types of functions: input function (I) and output function (O). These input-output functions describe flow of tokens from places to transitions and from transitions to places. Note that $F \subseteq (P \times T)$ U $(T \times P)$. Again $|P| = n$ and $|T| = m$, meaning an ordinary Petri net has n places and m transitions. This notation will be used in the subsequent discussions. A general place element is denoted by p_i and a general transition is denoted by t_j. As already mentioned, i = 1,2, ….. , n and j = 1,2, ….. , m.

Some authors [2] prefer to use the set of arcs (F) and a weight function (W) in the Petri net definition instead of using input-output functions (I and O). In this notation, a Petri net is represented by a four-tuple, $N = (P, T, F, W)$ where

$F \subseteq (P \times T)$ U $(T \times P)$ defines flow relation,

$W : F \rightarrow \mathcal{N}^+$ is a weight function, $\mathcal{N}^+ = \{1, 2, 3, \ldots\}$

Weight of arc is defined in the following way. If $I(p_i, t_j) = k$, where $k > 1$ is an integer, a directed arc from place p_i to transition t_j is drawn with the label (weight) k. If k = 1, an unlabeled arc is drawn and if it happens that k = 0 then no arc is drawn.

2.2.2 Alternative definition of Petri net

Note that Petri net is defined in Art 2.2.1 with two basic node elements: places and transitions. One can, however, define a Petri net with respect to a single node element i.e. either with respect to place or with respect to transition. To do that the concept of *preset* and *postset* is needed. With respect to transition one can define

$^\bullet t$, called the *preset* of transition t = set of all input places of the transition t

$t^\bullet$, called the *postset* of transition t = set of all output places of the transition t

Similarly, with respect to place one can define

$^\bullet p$, called the *preset* of place p = set of all input transitions of the place p

$p^\bullet$, called the *postset* of place p = set of all output transitions of the place p

Now one can define a *Petri net with respect to transition alone* as a net N_t such that $\forall t \in T$,

$^\bullet t = \{p : p \in P$ and $I(p,t) \neq 0\}$ and $t^\bullet = \{p : p \in P$ and $O(p,t) \neq 0\}$

Following similar notation, one can define a *Petri net with respect to place alone* as a net N_p such that $\forall p \in P$, $^\bullet p = \{t : t \in T$ and $O(p,t) \neq 0\}$ and $p^\bullet = \{t : t \in T$ and $I(p,t) \neq 0\}$

Unless otherwise stated, it is assumed that the Petri net has no isolated node i.e. no node $x \in P \cup T$ exists such that $^\bullet x = x^\bullet = \emptyset$.

2.2.3 Definition of Ordinary Petri net

An *Ordinary Petri net* is one where all arcs are unity-weighted (and hence unlabeled), mathematically represented by a four-tuple $N = (P, T, I, O)$ where $\forall\ p \in P$, $\forall\ t \in T$, $I(p,t) \leq 1$ and $O(p,t) \leq 1$.

It can be mentioned that ordinary and non-ordinary Petri nets have same modeling power since one can always represent an arc of higher weight as a set of arcs, each of unit multiplicity, the cardinality of the set being the weight of the arc of non-ordinary Petri net. Therefore, it is always possible to convert a non-ordinary Petri net into an ordinary Petri net without sacrificing generality but sometimes non-ordinary Petri nets are preferred due to ease of modeling.

2.2.4 Definition of Marked Petri net

A *Marked Petri net* is a five tuple $N = (P, T, I, O, M)$ where M can be viewed as a function, which assigns a natural number with each place, i.e. $M : P \rightarrow \mathcal{N}_0^+$. M can also be viewed as a vector given by $M_k = \{M_1, M_2, ..., M_i, ..., M_n\}$ where the i^{th} entry of M is M_i, which is the marking of the place p_i.

2.2.5 Definition of Pure Petri net

A *Pure Petri net* is one, which does not have any self-loop. It means, there exists no such place in the net, which is simultaneously an input place and an output place to a transition.

In mathematical representation, for a pure Petri net N, $P \cap T = \emptyset$ i.e. set of places and set of transitions are mutually disjoint. Following the alternative definition of Petri net, a pure Petri net N must satisfy the criteria that $\forall\ T$, $\{{}^\bullet t\} = \{t^\bullet\} = \emptyset$.

Petri nets having self-loops represent reflexive property and hence self-loop free pure Petri nets are also called *non-reflexive Petri nets*. Any impure Petri net (Petri nets having self-loops) can be made pure by adding appropriate dummy places and transitions to it.

Example 2.3:

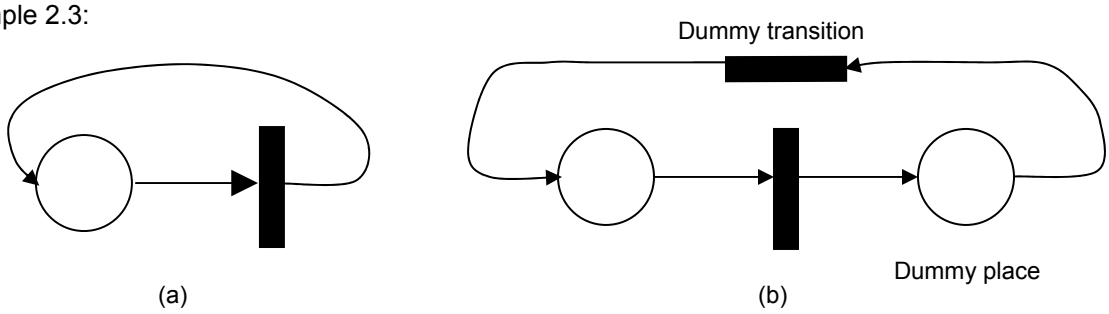


Figure 2.3: (a) Impure Petri net, (b) Pure Petri net

The Fig. 2.3 shows how an impure Petri net (a) can be made pure (b) by suitably adding dummy places and transitions.

2.2.6 Definition of Finite and Infinite Capacity Petri net

An *Infinite Capacity Petri net* is one in which each place can accommodate unlimited number of tokens.

Since in a physical system, tokens signify number of resources or whether a condition is true or whether a process is ongoing, depending upon what the place models, practical constraints limit the maximum number of tokens that each place can hold. Hence, a *Finite Capacity Petri net* is defined as one where each place has a maximum token carrying capacity.

2.2.7 Firing Rule

A transition is said to be *enabled* when each one of its input places is marked with at least one token (This statement assumes the net to be an ordinary one, if the net is non-ordinary then for enabling, each input place of the transition must be marked with at least w tokens, where w is the weight of the input arc of that transition).

An enabled transition may or may not *fire* depending on whether the event actually takes place or not. But once enabled, a transition has the potential to fire; hence, the transition is called *potentially friable*.

Firing enabled transitions performs *execution*. The execution of a Petri net causes its marking to change by removing tokens from its input places and depositing into each of its output places. Transition firing continues as long as there exists at least one enabled transition. When there are no enabled transitions, the execution *halts.*

In mathematical terms, a transition $t \in T$ is enabled iff $M(p) \geq I(p,t)\,;\forall\ p \in P$. If an enabled transition t fires then it causes a change in marking from $M(p)$ to $M'(p)$ given by the equation:

$$M'(p) = M(p) - I(p,t) + O(p,t)\ ;\ \forall\ p \in P\ .$$

As stated above, firing is a two-step process. First step involves removal of tokens from all input places and the second is depositing the tokens in the output places of the particular transition. The above equation clearly depicts this fact. $-I(p,t)$ stands for the first step and $+O(p,t)$ stands for the second.

The firing rule, when applied to finite capacity Petri nets, is called *Strict Firing Rule* and when applied to infinite capacity Petri nets, is called *Weak Firing Rule*. Thus transition enabling condition for finite capacity nets has an additional restriction that the number of tokens in each output place of the transition can not exceed the maximum token carrying capacity $C(p)$.

Theorem: Any pure finite capacity Petri net (where strict firing rule is applicable) can be transformed into a corresponding pure Petri net, where weak firing rule is applicable.

Thus one can apply weak firing rule for both finite and infinite capacity nets. Note that the theorem is stated with an additional constraint that the net has to be pure. As already mentioned in Art 2.2.5 this constraint can always be relaxed by making an impure Petri net pure.

Corollary: Weak firing rule can be applied to all Petri nets irrespective of capacity constraint and purity constraint.

The transformation which converts a finite capacity net (N, M_0), where strict firing rule is applicable to a corresponding Petri net (N', M_0'), where weak firing rule is applicable, is called *Complementary-Place Transformation.* The procedure for obtaining the transformation is given by the following two steps:

Step 1: Add a complementary place p' for each place p, where the initial marking of p' is

$$M_0' = C(p) - M_0(p)\ .$$

Step 2: Between each transition t and a subset of complementary places (p'), new arcs (t, p') and/or (p', t) (input and/or output) are drawn such that $w(t, p') = w(p, t)$ and

$w(p', t) = w(t, p)$; this ensures that the sum of tokens in place and in its complementary place p' equals its capacity $C(p)$ for each place p, before and after the transition t.

Example 2.4

The net in Fig. 2.4(a) is a finite capacity net where strict transition rule is applicable. At the initial marking (1 0), the only enabled transition is t_1. After t_1 fires, new marking becomes (2 0), where t_2 and t_3 are enabled. If t_2 fires then next marking becomes (0 0) and if t_3 fires then the next marking becomes (0 1). Continuing this process one can obtain the reachability graph of the PN as shown in Fig. 2.4 (c) (reachability trees and graphs are introduced in Chapter 5). Using Complementary-place transformation one can transform the net in Fig. 2.4(a) to the one in Fig. 2.4(b) which have the same reachability graph Fig. 2.4(c). The first step for the transformation is to introduce two complementary places p_1' and p_2' with initial markings $M_0'(p_1') = C(p_1) - M_0(p_1) = 2 - 1 = 1$. In the next step, new arcs are added between each transition t and some complementary places, so as to keep the sum of tokens in each place-complementary place pair constant, the value of this constant being capacity C of the place. For example, since $w(t_1, p_1) = 1$, $w(p_1', t_1) = 1$. Similarly, $w(t_3, p_1') = w(p_1, t_3) = 2$ and $w(p_2', t_3) = w(t_3, p_2) = 1$, since firing t_3 removes 2 tokens from the place p_1 and adds 1 token in p_2. Hence a 2-weighted arc is drawn from t_3 to p_1' and unity-weighted arc is drawn from p_2' to t_3. Similarly other additional arcs are drawn to convert the finite capacity net an infinite capacity one. It can be verified that both the nets in Fig. 2.4 have isomorphic reachability graphs.

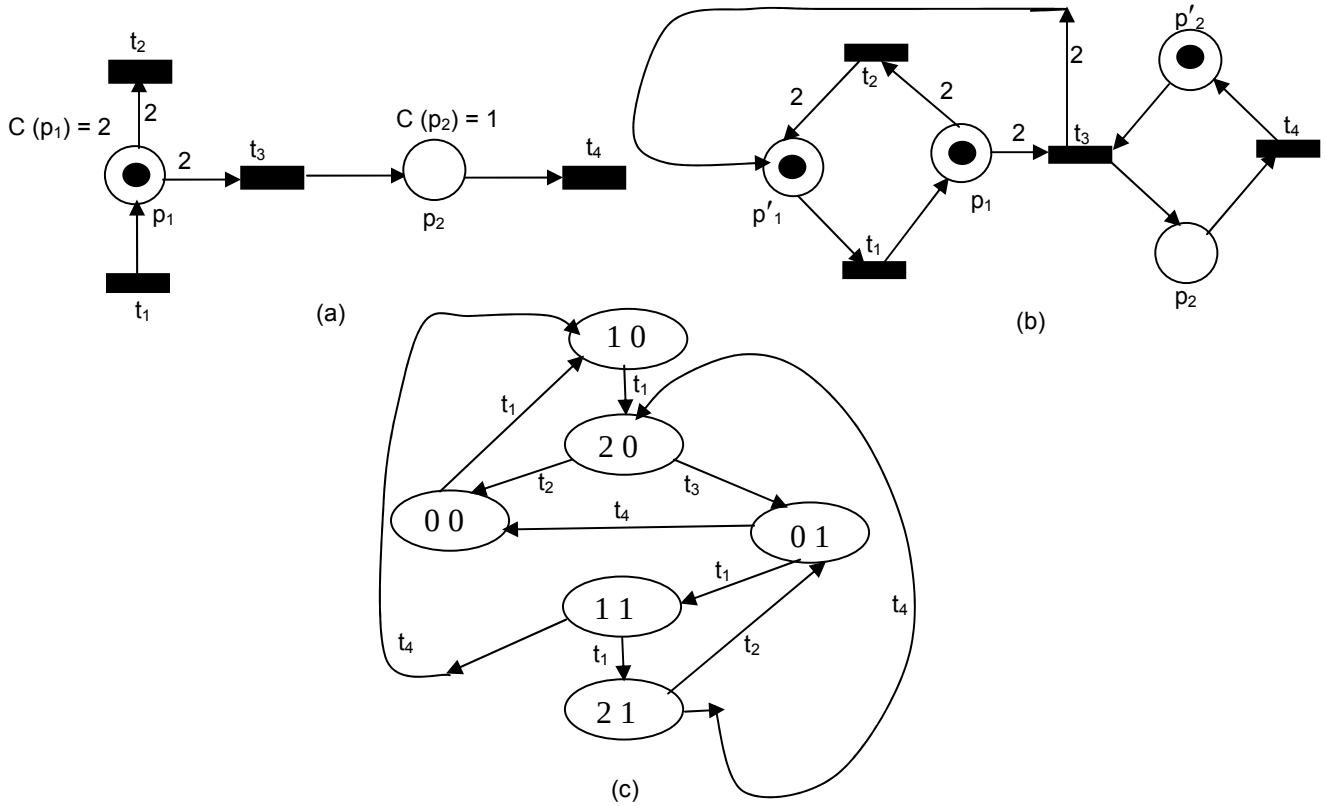


Figure 2.4: (a) Finite-capacity Petri net, (b) Infinite Capacity Petri net after complementary-place transformation, (c) The Complementary-place transformation preserves the reachability graph

Chapter 3

Modeling with Petri Nets

3.1 Petri Nets as Modeling Formalism

Modeling a hybrid system has been attempted from many different perspectives. A hybrid system consists of some continuous functional relations, some discrete signals and some event driven occurrences. Some attempts have gone to model the entire system in continuous domain by making some simplified assumptions; some have attempted to model the system entirely in discrete domain by discretizing the continuous domain by some suitable assumptions. Of course, all these attempts produce results at the expense of losing information about the system due to modeling assumptions. Petri net is an efficient modeling tool for modeling hybrid system since it can inherently capture DEDS and properties like concurrency, asynchronous behavior, non-determinism are intrinsic to Petri nets. This chapter introduces Petri net as a modeling tool and explains how efficiently it can express uncertain and hybrid nature of complex systems.

3.2 Basic Modeling Constructs

In this section, some basic situations are taken which are encountered often during modeling a physical system. This section describes how Petri net handles these real life modeling situations, thus revealing the modeling power and ease of representation of Petri nets.

3.2.1 Sequential execution

Sequential execution poses a precedence constraint among the activities (transitions). In Fig 3.1 transition t_2 can fire only after the firing of t_1.

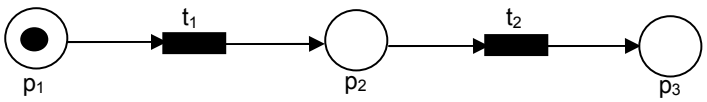


Figure 3.1: Transition t_1 occurs first and then transition t_2 occurs

3.2.2 Synchronization

Petri nets can successfully capture the synchronization mechanism in the modeling phase. In Fig 3.2 transition t_1 will fire only when the empty input place gets a token. Thus, the three input places of t_1 are synchronized for the firing of transition t_1.

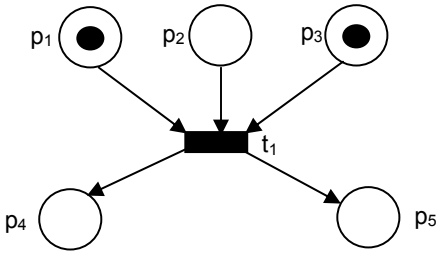


Figure 3.2: Transition t_1 fires when the place p_2 gets a token so that all the input places of transition t_1 have tokens

3.2.3 Conflict

In Fig 3.3 transitions t_1, t_2 and t_3 are in conflict. All three transitions are enabled but only one can fire at a time. Hence, choice has to be made regarding which transition will be fired. Firing one will lead to the disabling of other transitions. The conflict is resolved in a non-deterministic way (e.g. by assigning appropriate probabilities to the conflicting transitions).

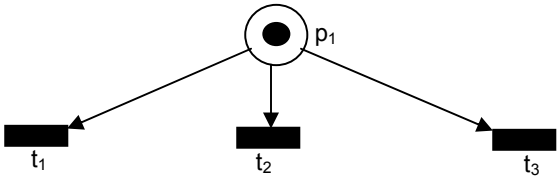


Figure 3.3: Transitions t_1, t_2 and t_3 are in conflict

3.2.4 Concurrency

In Fig. 3.4 transitions t1, t2 and t3 are concurrent. Concurrency is characterized by the existence of a forking transition that deposits tokens simultaneously in two or more output places. In Fig 3.4 t_0 is the forking transition.

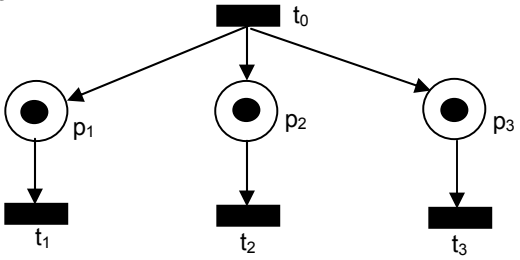


Figure 3.4: Transitions t_1, t_2 and t_3 are concurrent

Petri pointed out that concurrency can be thought of as a binary relation which has:
(1) Reflexive property (event A is concurrent with itself),
(2) Symmetric property (event A and event B are concurrent implies event B and event A are concurrent).

But concurrency does not have transitive property (event A and event B are concurrent, event B and event C are concurrent togetherly, in general, does not imply event A and event C are concurrent. Of course, this may happen in some special cases. Example: Zeroth Law of Thermodynamics).

3.2.5 Confusion

Confusion occurs when conflict and concurrency co-exist. In such a situation, it is not clear that whether a conflict is needed to be resolved or not, in going to the new state (marking). In Fig 3.5 transitions t_1 and t_3 are concurrent whereas transitions t_1 and t_2 are in conflict. Also t_2 and t_3 are in conflict.

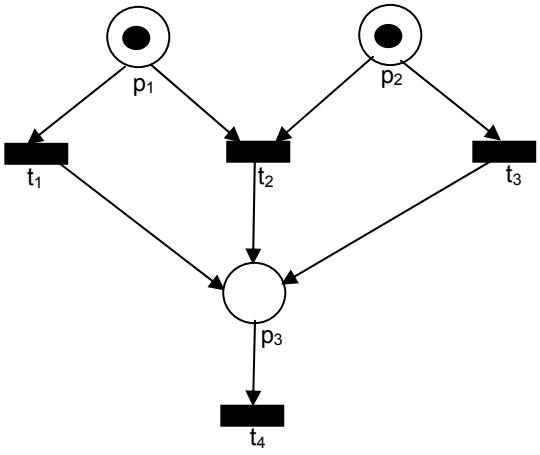


Figure 3.5: Transitions t_1, t_2 and t_2, t_3 are in conflict but t_1, t_3 are concurrent

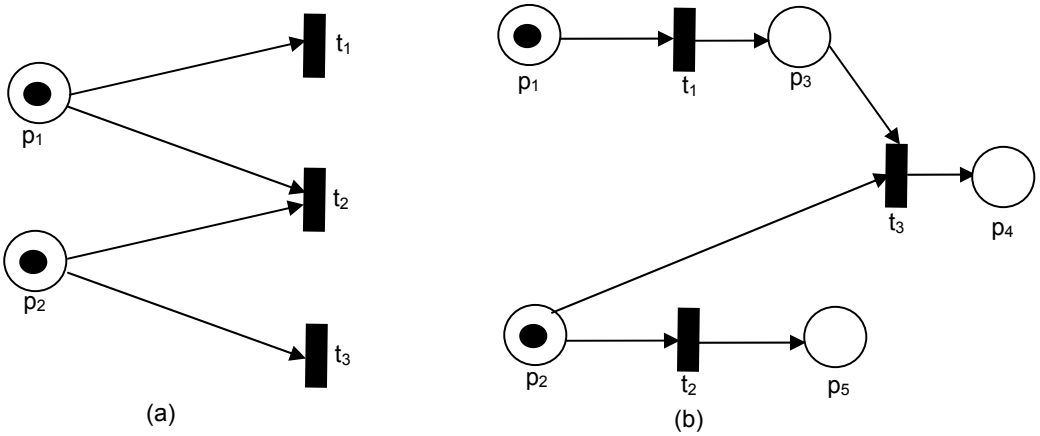


Figure 3.6: (a) Symmetric Confusion (b) Asymmetric Confusion

Confusions can be of two types: Symmetric Confusion and Asymmetric Confusion. Fig 3.6 (a) shows Symmetric Confusion where t_1 and t_3 are concurrent (both enabled and firable) and at the same time they are in conflict with t_2.

In Fig 3.6 (b), t_1 and t_2 are concurrent and if t_1 fires first, then t_3 and t_2 will be in conflict. This situation is called Asymmetric Confusion. Asymmetric confusion occurs when one place feeds to a set of transitions via output arcs from it and there exists another place in the net which feeds to a subset of those transitions. In Fig 3.6 (b) the place p_2 feeds to a set of transitions $\{t_2, t_3\}$ via output arcs from p_2 and there exists a place p_3 in the net which feeds to $\{t_3\} \subseteq \{t_2, t_3\}$.

3.3 Primitives for Programming Constructs

This section describes basic programming constructs in Petri net formalism. This, in turn, will express the modeling power of Petri nets and these constructs will be used in subsequent modeling examples.

3.3.1 Selection (if – else)

(a) *If* condition A then do activity X, else do activity Y.

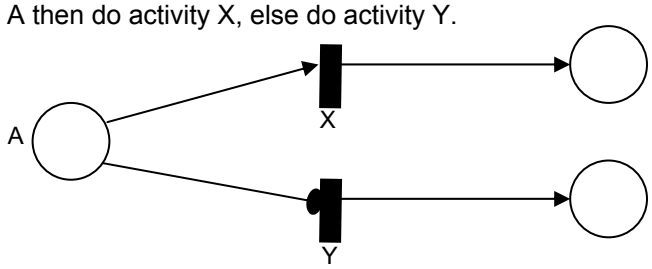


Figure 3.7: If – else condition

(b) *If* condition A *and* condition B hold, then do activity X.

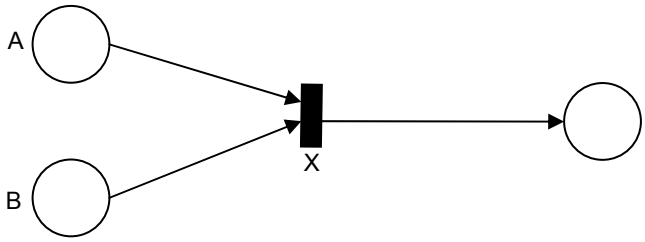


Figure 3.8: If – else with and operator

3.3.2 Case (Switch) statement

If *Case A* do activity P, if *Case B* do activity Q, if *Case C* do activity R, if *Case D* do activity S.

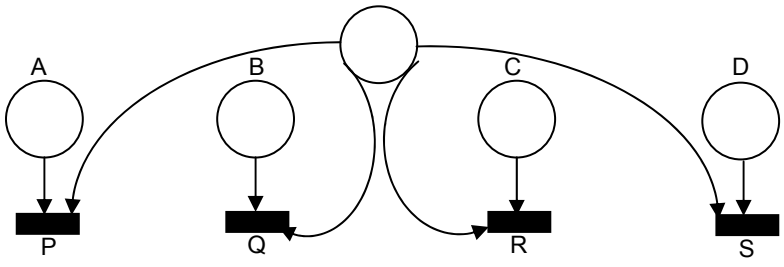


Figure 3.9: Switch statement

3.3.3 While loop

While condition A holds, *do* activity X.

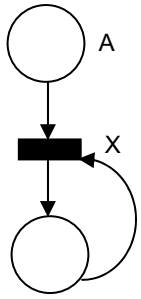


Figure 3.10: While loop

3.3.4 Repeat (for) loop

For condition A, do activity X.

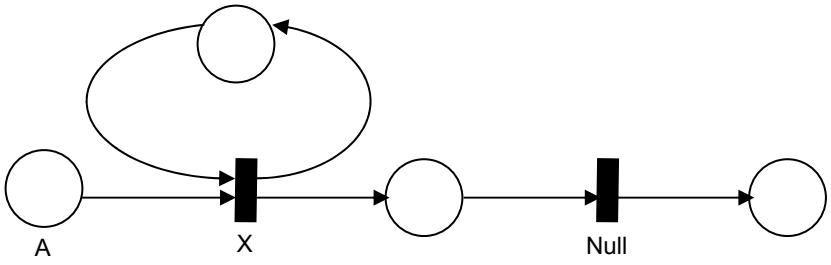


Figure 3.11: For loop

3.3.5 Precedence

Activity X should *precede* activity Y.

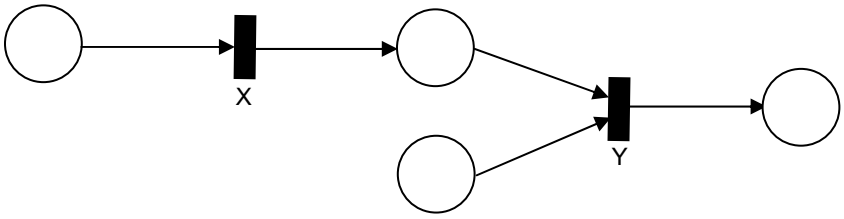


Figure 3.12: Precedence relation

3.3.6 Timed occurrence

After k seconds do activity X

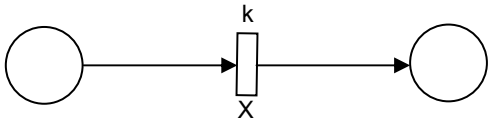


Figure 3.13: Timed transition

3.3.7 Either – or (Mutual exclusion)

(a) *Either* do activity X *or* do activity Y.

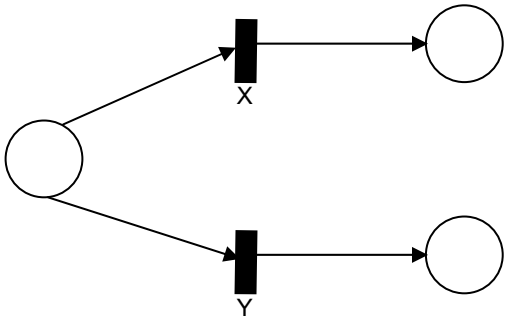


Figure 3.14: Either – or statement

(b) *Either* do activity X *or* do activity Y *with preference to* activity X *(preferential either - or)*

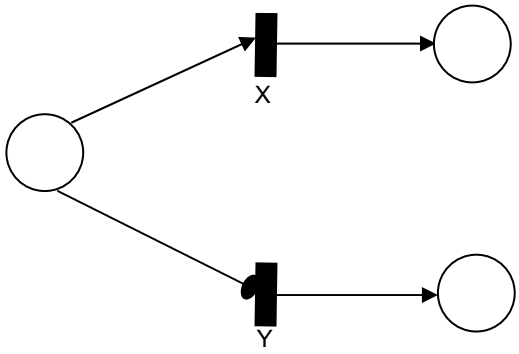


Figure 3.15: Preferential either – or statement

Chapter 4

Sub-structures of Petri Nets

4.1 Importance of sub- structures

Modeling and analyzing a Petri net as it is, often becomes cumbersome and tedious job because even for a physical system with modest complexity, the Petri net size becomes unmanageable and analyzing the model becomes a daunting task. Hence it may be helpful if one can identify some sub-structures in the entire large Petri net and then model and analyze those substructures; in that case it becomes a simpler problem. Moreover if certain properties can be established for these sub-structures then identifying one such sub-structure will immediately allow one to assign those properties without any analysis. Last but not the least, if certain properties can be shown to be preserved by these sub-structures then those properties will also hold for the composition of them i.e. for the original Petri net. This justifies the study of sub-structures of a Petri net.

4.2 Sub-structures of Petri nets

4.2.1 Source and sink

A Source Place is a place that has no input transition i.e. it has no input arcs, only output arcs emanate from a source place. A Sink Place is one that has no output transition i.e. it has no output arcs, only input arcs converge to a sink place.

Similarly, a Source Transition is a transition that has no input place i.e. it has no input arcs, only output arcs emanate from a source transition. A Sink Transition is one that has no output place i.e. it has no output arcs, only input arcs converge to a sink transition.

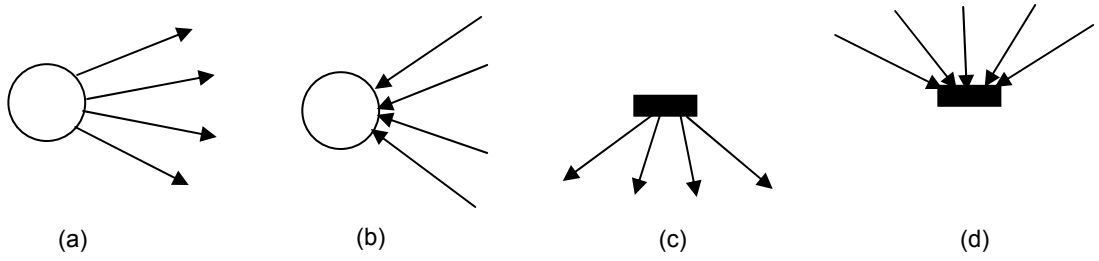


Figure 4.1: (a) source place, (b) sink place, (c) source transition, (d) sink transition

Fig 4.1 (a) and (b) shows source and sink places respectively while (c) and (d) shows source and sink transitions respectively. Mathematically, $^\bullet p = \varnothing$ represents a source place, $p^\bullet = \varnothing$ represents a sink place, $^\bullet t = \varnothing$ represents a source transition and $t^\bullet = \varnothing$ represents a sink transition.

4.2.2 Directed path (DP)

A Directed path (DP) is a path formed by a finite sequence of (not necessarily distinct) places and transitions, present in the Petri net.

Example 4.1

Ina general Petri net (n places and m transitions) p_1, t_2, t_1, p_3, p_8, t_2, t_4 can be a directed path (DP). Note that a DP need not contain a sequence where places and transitions are alternative elements.

4.2.3 Simple Directed path (SDP)

A simple directed path of a PN, N is a sequence of transitions and places given by:
$\Theta = t_0 p_1 t_1 ... p_r t_r$; containing no place or transition more than once such that

$\forall i = 1,2,...,r; [I(p_i, t_i) = O(p_i, t_{i-1}) = 1 \wedge I(p_i, t) = 0$
if $t \neq t_i \wedge O(p_i, t) = 0$ if $t \neq t_{i-1} \wedge I(p, t_i) = 0$
if $p \neq p_i \wedge O(p, t_{i-1}) = 0$ if $p \neq p_i$]

Note that a single transition can be considered as a SDP with no places.

4.2.4 Looping of paths (LOP)

Given a Petri Net N, and k SDPs $\Theta i = t_{i_o} ... t_{i_{ri}}$ $(i = 1,...,k)$, the k paths are said to be looped in N if $\exists p \in P, (\forall i = 1,...,k) \mid [I(p, t_{i0}) = O(p, t_{i_{ri}}) = 1 \wedge (\forall p' \neq p) I(p', t_{i0}) = O(p', t_{i_{ri}})]$

In simple terms, the k-paths are looped if the input (output) transitions of all paths input from (output to) the same place p with a single arc [3].

4.2.5 Directed circuit (DC)

A Directed circuit (DC) is a closed directed path i.e. DC is a DP from one node (place or transition) back to itself.

Example 4.2

Ina general Petri net (n places and m transitions) p_1, t_2, t_1, p_3, p_8, t_4, p_1 can be a directed circuit (DC).

4.2.6 Pure Directed path (PDP)

A *Pure Directed Path (PDP) i*s a directed path (DP) such that each place in the DP has exactly one input and one output transition, and each transition in the DP has exactly one input and one output place except starting or ending ones (place or transition).

Mathematically PDP is a DP such that other than a starting or ending node,
$\forall p \in P_1, \forall t \in T_1, \mid \{p\}\bullet \mid = \mid \bullet\{p\} \mid = \mid \{t\}\bullet \mid = \mid \bullet\{t\} \mid = 1$; where P_1 and T_1 are the sets of places and transitions in the directed graph [5].

This means that, the Petri net might have places which have multiple input and/or output transitions; and/or the net might have transitions which have multiple input and/or output places.

But when the DP contains no such nodes which have multiple input and/or output nodes (of other type; recall a Petri net has two kinds of nodes – places and transitions); then only a DP is called a PDP. So a PDP is a very special kind of DP.

4.2.7 Pure Directed circuit (PDC)

A *Pure Directed Circuit* (PDC) is a directed circuit (DC) such that each place in the DC has exactly one input and one output transition, and each transition in the DC has exactly one input and one output place.

Mathematically PDP is a DP such that $\forall p \in P_1, \forall t \in T_1, |\{p\}\bullet| = |\bullet\{p\}| = |\{t\}\bullet| = |\bullet\{t\}| = 1$; where P1 and T1 are the sets of places and transitions in the directed graph.

This means that, the Petri net might have places which have multiple input and/or output transitions; and/or the net might have transitions which have multiple input and/or output places. But when the DC contains no such nodes which have multiple input and/or output nodes (of other type; recall a Petri net has two kinds of nodes – places and transitions); then only a DC is called a PDC. So a PDC is a very special kind of DC.

Note that the concept of path and circuit are borrowed from graph theory. In the formal definition of Petri net (Art 2.2.1), it was stated that Petri net is a *bipartite, weighted, directed multigraph*. The meaning of this sentence can be made clear now. A graph has only two elements: *vertices* and *edges*. A graph is called *bipartite* when its vertices can be grouped into two subgroups. In Petri graph context the vertices are comparable to nodes which can be grouped into two subgroups: places and transitions. Since Petri graph contains weighted and directed arcs, it is a *weighted* and *directed* graph. Petri graph is called a *multigraph* since multiple arcs can be drawn from one node to another. In a directed graph, a *path* is a finite sequence of edges $e_1, e_2, ..., e_n$ where $ei = (v_{i-1}, v_i); i = 1,2,3,...,n$. A path is called *simple* if all of its edges are distinct and is called *elementary* if all of its vertices are distinct (which implies edges are also distinct). A path is said to be *open* if $v_0 \neq v_n$ and *closed* if $v_0 = v_n$. A closed path is called a *circuit*. Just like simple and elementary path, a *simple circuit* means all edges are distinct and an *elementary circuit* means all vertices are distinct (except of course, the first and last, which coincide). The *length of a path or circuit* is the number of edges in it. These basic notions of graph theory are introduced here in order to clarify the origin and significance of terms introduced in Art 4.1. This allows one to apply the concepts of simple and elementary paths and circuits in Petri graph context. As it will be evident in the subsequent part of this report, some more concepts will be taken from graph theory to draw important conclusions about Petri net.

4.2.8 Subnet (SN)

A subnet (SN) of a PN, N = (P, T, I, O) is another PN, N1 = (P1, T1, I1, O1) such that P1$\subseteq$ P, T1$\subseteq$T and I1 and O1 are simply I and O projected onto (P1 x T1). Mathematically I1 and O1 are given as follows:

I1: (P1 x T1) $\rightarrow \mathcal{N}_o^+$

O1: (P1x T1) $\rightarrow \mathcal{N}_o^+$

Following the alternative definition of PN, a subnet N1 = (P1, T1, F1, W1) of the original Petri net N = (P, T, F, W) satisfies that, $P_1 \subseteq P$, $T_1 \subseteq T$, $F_1 = F \cap ((P1 \times T1) \cup (T1 \times P1))$, where F and F1 are the set of arcs of N and N1 respectively. Arc weights are represented by the weight functions W and W1 respectively.

If a subnet of a Petri net is defined with initial marking $M^0 = m_0$, in that case $M^0_1 = m01 \subseteq M^0 = m_0$. For a general marked PN and subnet of that, $M_1 \subseteq M$.

The above discussion essentially means, subnet is a subset of elements of the original PN, with all arcs between those nodes in the subset intact and no other arcs present. A large PN is nothing but a composition of smaller component subnets. Since subnets of a PN are themselves distinct Petri nets, subnets can not share places [4].

4.2.9 P-Subnet (PSN)

A subnet (SN), N_1 of a PN, N is called a Place subnet or P-subnet (PSN), iff $T_1 = \bullet\{P_1\} \cup \{P_1\}\bullet$.

4.2.10 T-Subnet (TSN)

A subnet (SN), N_1 of a PN, N is called a Transition subnet or T-subnet (TSN), iff $P_1 = \bullet\{T_1\} \cup \{T_1\}\bullet$.

4.2.11 RP-Subnet (RPSN)

A P-subnet (P1, T1, I1, O1) is called a Restricted P-subnet (RPSN) if $T_1 = \bullet\{P_1\} = \{P_1\}\bullet$.

The notion of RPSN was first introduced by V. K. Agrawal in 1986 [5]. The importance of this class of subnet lies in computing invariants, which will be discussed later in this report.

4.2.12 Dual net

The Dual of a PN, N = (P, T, I, O) is given by $\hat{N} = (\hat{P}, \hat{T}, \hat{I}, \hat{O})$ where $\hat{P} = T, \hat{T} = P, \hat{I} = O$ and $\hat{O} = I$. .

Intuitively, the dual of a PN results when its places are changed to transitions and transitions are changed to places.

Theorem: The dual of an unmarked PN is also an unmarked PN.

Proof: Say the original unmarked PN is given by N = (P, T, I, O). From the definition of dual, $\hat{N} = (\hat{P}, \hat{T}, \hat{I}, \hat{O})$, the dual of N, is a four tuple.

Since N was an unmarked net, neither T nor P contain any unconnected elements, so neither do $\hat{P} = T$ or $\hat{T} = P$. Hence, $\hat{M}$ is an unmarked net.

Theorem: The dual of a dual net is original Petri net.

Proof: This is obvious, by repeated application of the definition of Dual.

Theorem: If an unmarked PN is the subnet of other iff dual of first is the subnet of dual of second.

Proof: Say the first PN is given by N = (P, T, I, O). Given that, N is the subnet of another PN, $N' = (P', T', I', O')$ i.e. $N \subseteq N'$. Now by the definition of subnet, $P \subseteq P', T \subseteq T'$; I and O are I' and O' projected onto $(P' \times T')$.

The definition of dual means that

$\hat{P} \subseteq \hat{P}'$ [Since $\hat{P} = T$ and $\hat{P}' = T'$]; $\hat{T} \subseteq \hat{T}'$ [Since $\hat{T} = P$ and $\hat{T}' = P'$]

Again, $\hat{I}$ is a projection of $\hat{I}'$ [since $\hat{I} = O$ and $\hat{I}' = O'$]

$\hat{O}$ is a projection of $\hat{O}'$ [since $\hat{O} = I$ and $\hat{O}' = I'$]

And thus $\hat{N}$ is a subnet of $\hat{N}'$.

Since the dual of $\hat{N}$ is N (from Theorem 3), the above argument with $\hat{N}$ for N gives the reverse implication. This completes the proof.

If dual is thought of as an operation instead of a sub-structure, the above three theorems can be summarized as follows: the dual operation preserves the unmarked, dual and subnet properties.

4.3 Other important Petri net structures

Below mentioned are some important definitions on Petri net structures. The idea behind grouping them separately is that they are not sub-structures of a Petri net. This section introduces the concepts of *composition of nets* which is a super-structure of nets, *projection of nets* which is a transformed structure of nets, *connectivity* and *strong connectivity* of nets which is a special structure of nets and *synchronic distance* which is a special measurement structure for transitions of nets.

4.3.1 Composition of Petri nets and Composed PN

The composition operation can be thought of set theoretic analogue of union. So the problem is how to unite two or more Petri nets to get a single PN, called Composed PN.

Given two nets $N_1 = (P_1, T_1, I_1, O_1)$ and $N_2 = (P_2, T_2, I_2, O_2)$ with initial marking m_{01} and m_{02}.

Let, $\Theta = \{\theta_1, \theta_2, ..., \theta_k\}$ is a set of simple paths present in both the nets.

Let's assume, $P_1 \cap P_2 \setminus \{p \mid (\exists \theta \in \Theta)[p \in \theta]\} = \varnothing$ and $T_1 \cap T_2 \setminus \{t \mid (\exists \theta \in \Theta)[t \in \theta]\} = \varnothing$.

Let's also assume that $(\forall p \in P_1 \cap P_2)[m_{01}(p) = m_{02}(p)]$.

The Concurrent Composition of N_1 and N_2 is the net $N = (P, T, I, O)$ with initial marking m_0.

Where, $P = P_1 \cup P_2; T = T_1 \cup T_2;$

$I(p,t) = I_i(p,t)$ If $(\exists i \in \{1,2\})[p \in P_i \wedge t \in T_i]$

= 0 otherwise.

$O(p,t) = O_i(p,t)$ If $(\exists i \in \{1,2\})[p \in P_i \wedge t \in T_i]$

= 0 otherwise.

$m_0(p) = m_{01}(p)$ If $p \in P_1$

= $m_{02}(p)$ otherwise.

The composed net N is denoted as $N = N_1 \,||\, N_2$.

4.3.2 Projection of Petri nets and Projected PN

Let N be a composed net: $N = N_1 \,||\, N_2 \,||\, N_3 \,||\, ... \,||\, N_n$ and let m be a marking. Let $\vec{\sigma}$ be a firing count vector (will be defined in Chapter 5) and let σ be a firing sequence (will be defined in Chapter 5) defined on $\vec{\sigma}$.

The projection of m over N_i, denoted as $\pi_i(m)$, is the vector obtained from m by removing all the components associated to the places not present in N_i.

The projection of $\vec{\sigma}$ over N_i, denoted as $\pi_i(\vec{\sigma})$, is the vector obtained by $\vec{\sigma}$, removing all the components associated to transitions not present in N_i.

The projection of σ over N_i, denoted as $\pi_i(\sigma)$, is the firing sequence obtained by σ, removing all the transitions not present in N_i.

It follows that,

(1) N generates the string σ sequencing from m_0 to m.

(2) N_i generates $\pi_i(\sigma)$ sequencing from $\pi_i(m_0)$ to $\pi_i(m)$.

4.3.3 Inverse of a PN or Reversed PN

Inverse of a PN or Reversed PN is defined as one which results if the direction of each arc is reversed in the original Petri net.

Thus N^{-1} is obtained from N by reversing all arc directions of N. Mathematically, if $N = (P, T, I, O)$ then by definition $N^{-1} = (P, T, I^{-1}, O^{-1})$ where $I^{-1} = O$ and $O^{-1} = I$. Following the alternative definition of PN, $N = (P, T, F, W)$ and $N - 1 = (P, T, F^{-1}, W)$. Note that in the inverse PN, the places and transitions are kept intact. Only the arc direction reverses. This is the basic difference between reversed net and dual net. In the later, as defined earlier, in addition to arc direction reversal, places and transitions are also swapped.

4.3.4 Connectivity and Strong Connectivity

Before giving formal definition of PN connectivity, it is worthwhile to look into the origin of this concept in graph theory. A directed graph is called connected if its corresponding undirected graph (formed from the directed graph by removing its edge arrows) is connected. An undirected graph is connected if there exists at least one path between any two vertices.

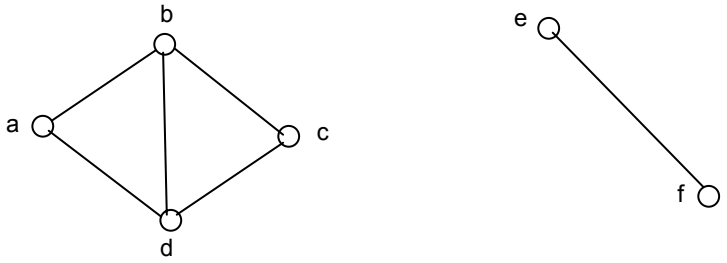


Figure 4.2: Unconnected graphs [12]

A directed graph (often called digraph) is strongly connected if for every two vertices $a, b \in V$ (set of vertices), there is a path from a to b and a path from b to a as well.

A PN is said to be connected if there exists at least one path between any two nodes (of course, the two nodes have to be transition-place doublet as no path can exist between two transitions or two paths).

Mathematically a PN is said to be connected if $\exists$ a path between any arbitrarily chosen place-transition pair (p_i, t_j). A PN is said to be strongly connected if $\exists$ reversible paths between any arbitrarily chosen place-transition pair (p_i, t_j) .

4.3.5 Traps and Siphons

4.3.5.1 Traps

A *Trap* is a state of places in a PN, such that such that every transition that inputs from one of these places, also outputs to one of these places [9].

Formally, a non-empty subset of places Q in a PN, is called a *trap* if $Q\bullet \subseteq \bullet Q$, i.e. every transition having an input place in Q has an output place in Q [2].

Example 4.3

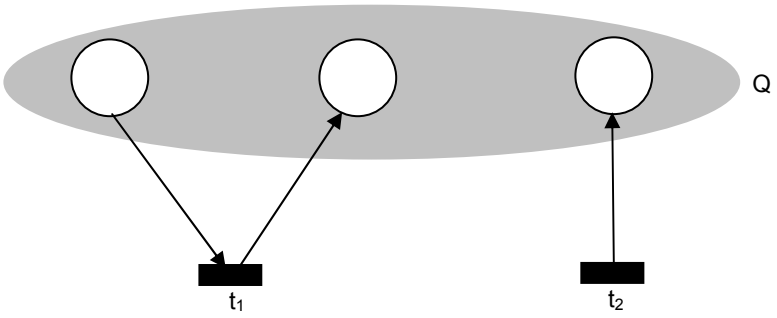


Figure 4.3: Traps

In Fig. 4.3 the three places shown, form a trap. From the figure, $Q\bullet = \{t_1\}$; $\bullet Q = \{t_1, t_2\}$; $Q\bullet \subseteq \bullet Q$. Token count in this trap remains the same by firing t_1 but increases by firing t_2 .

4.3.5.2 Siphons

A *Siphon* (or *Deadlock*) is a set of places in a PN, such that every transition that outputs to one of these places also inputs from one of these places [9].

Formally, a non-empty subset of places S in a PN, is called a *siphon* if $\bullet S \subseteq S \bullet$, i.e. every transition having an output place in S has an input place in S [2].

Example 4.4

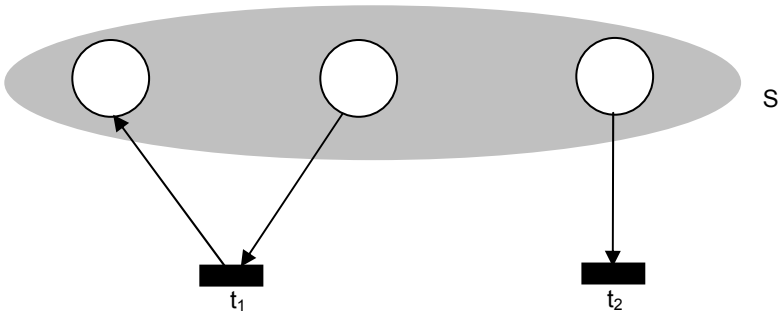


Figure 4.4: Siphons

In Fig. 4.3 the three places shown form a trap. From the figure, $\bullet S = \{t_1\}; S\bullet = \{t_1, t_2\}$; $\bullet S \subseteq S \bullet$. Token count in this siphon remains the same by firing t_1 but decreases by firing t_2 .

Table 4.1 provides a quick overview of the properties of traps and siphons.

Properties	Trap	Siphon
Behavioral property	By definition, once a place in a trap has a token, there will always be a token in at least one of the places in the trap. Hence, a trap having at least one token can never lose all of its tokens. In other words, if a trap is marked under some marking, it remains marked under each successor marking.	By definition, once all places in a siphon have no token, there will never be a token in any one of the places in the siphon. Hence, a siphon having lost all of its tokens can never obtain a token again. In other words, if a siphon is token-free under some marking, then it remains token-free under each successor marking.
Union	Union of two traps is again a trap [2].	Union of two siphons is again a siphon [2].
Nominal or Basic	A trap is called a *basic trap* or *nominal trap* if it can not be represented as a union of other traps.	A siphon is called a *basic siphon* or *nominal siphon* if it can not be represented as a union of other siphons.
Minimal	A trap is said to be *minimal* if it does not contain any other trap.	A siphon is said to be *minimal* if it does not contain any other siphon.

Table 4.1: Properties of traps and siphons

Theorem: All minimal traps (siphons) are basic traps (siphons) but not all basic traps (siphons) are minimal.
Theorem: A subset of places in the reversed net N^{-1} will be a siphon (trap) iff it is a trap (siphon) in the original net N .
Theorem: Traps and siphons are duals of each other.

Example 4.5

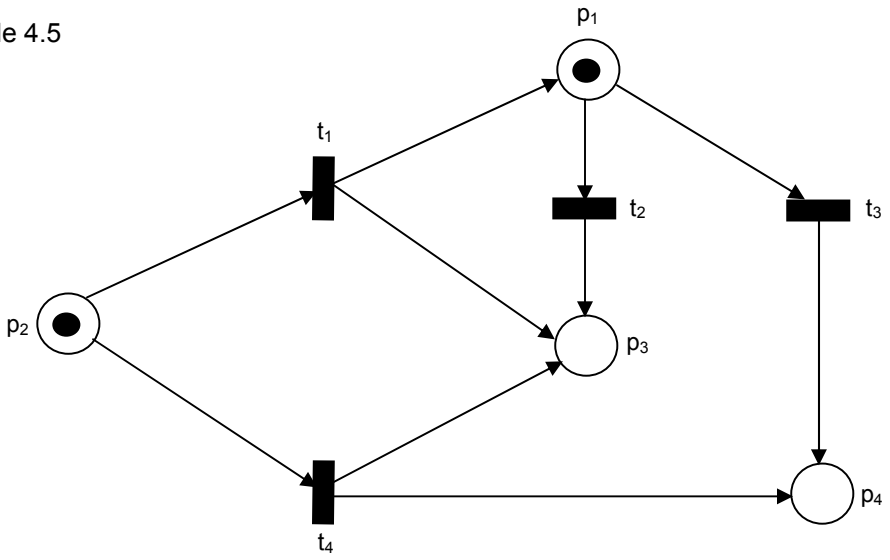


Figure 4.5: Example of traps and siphons

Let $S_1 = \{p_1, p_2, p_3\}; S_2 = \{p_1, p_2, p_4\}$; $S_3 = \{p_1, p_2, p_3, p_4\}; S_4 = \{p_2, p_3\}$ and $S_5 = \{p_2, p_3, p_4\}$. Then, one can verify that $\bullet S_1 = \{t_1, t_2, t_4\} \subseteq S_1 \bullet = \{t_1, t_2, t_3, t_4\}$. Thus S_1 is a siphon. Again $S_4 \bullet = \{t_1, t_4\} \subseteq \bullet S_4 = \{t_1, t_2, t_4\}$. Thus S_4 is a trap. Similarly it can be shown that S_2 is a siphon, S_3 is both a siphon and a trap and S_5 is a trap. Also, S_1 and S_2 are both minimal and basic siphons. S_3, S_4 and S_5 are basic traps, S_3 and S_5 are not minimal traps.

4.3.5.3 TC and SC net

A PN in which the set of places in every directed circuit (DC) is a trap (siphon), is called a *Trap-circuit net* (*Siphon-circuit net*) or *TC* (*SC*) net.

4.3.5.4 TCC and SCC net

A PN in which the set of places in every directed circuit (DC) contains a trap (siphon) is called *TCC* (*SCC*) net.

Chapter 5

Analysis of Petri Nets

5.1 Importance of analysis

The objective of using Petri nets in system study is to draw important conclusions about the system without going for time and cost ineffective trial and error prototyping. To do so, the first step is to model the system. Once a model is ready, the next task is to analyze the model to draw conclusions about the properties of the model and hence about the actual system. Then only one can answer questions like what the system behavior is supposed to be under specific operational conditions, what properties are inherent to the structure of the net, what to expect and what not to expect from the system during operation and whether there is any pitfall in the system design which must be avoided in operational phase.

The first step i.e. modeling has been addressed in Chapter 3. This chapter deals with analysis of the modeled system. Chapter 4 serves as a link between Chapter 3 and Chapter 5, since dealing with substructures eases both modeling and analysis.

It is worthwhile to mention that some authors prefer to discuss PN properties and PN analysis separately. This approach may seem more systematic but in the process it can only introduce behavioral properties, because definition and understanding of structural properties are so closely related to structural analysis that they can not be discussed separately. Keeping this in mind, this report discusses the properties and analysis techniques together to emphasize the fact that analysis is more than a mere tool, the analysis methods themselves reveal elegance of PN model and its properties.

5.2 Analysis approaches

There are three major approaches for PN analysis:(1) Behavioral approach, which is a tree based approach, (2) Structural approach, which is a matrix based approach and (3) Reduction or Refinement approach, which is a net simplification approach. These three approaches are discussed here.

5.2.1 Behavioral approach

This approach deals with the behavioral properties of PN. Behavioral properties are those which are dependent on the initial marking. In the following discussion, behavioral properties are treated in detail. Then effort has been given to make conclusions about them from analysis.

5.2.1.1 Reachability

The *Reachability Problem* is stated as follows. Given a Petri net, given an initial marking m_0, given another marking m_r; the question is whether there exists a sequential firing of transitions which will bring the net from m_0 to m_r. If the answer is 'yes', then m_r is said to be *Reachable* from m_0.The set of all possible markings reachable from m_0, is called the *Reachability Set*, denoted by the symbol R(m_0) for a given PN. Note that reachability set is defined for a given PN, for a given initial marking m_0. This dependency on initial marking clearly reveals that *reachability is a behavioral property*.

It may happen that m_r is reached from m_0 by the firing of a single transition, in that case m_r is said to be immediately reachable from m_0. In the general case, m_r is reached via the sequential firing of r transitions, called the *firing sequence*, denoted by $\sigma_r = t_{j_1} t_{j_2} ... t_{j_r}$; where $r \in [1, m]$, m being the total number of transitions present in the PN. This means that firing sequence σ_r contains an ordered string of transitions, the length of the string being equal to r. Note that the transition string defining firing sequence may contain repetition. Symbolically, $m_0 \xrightarrow{\sigma_r} m_r$. The fact that m_r is reachable from m_0 via σ_r, is sometimes represented [14] by the notation $m_0[\sigma_r > m_r$. For a given PN, the set of all possible firing sequences from initial marking m_0 is denoted by $L(N, m_0)$ or simply $L(m_0)$.

Example 5.1

Suppose a PN has total 7 transitions $(m = 7)$. Let there exists a firing sequence of length 4 $(r = 4)$, which brings the PN from an initial marking m_0 (given) to another marking m_4 (given). This firing sequence which brings m_0 to m_4 is given by $\sigma_4 = t_1 t_3 t_2 t_3$ implying $j_1 = 1, j_2 = 3, j_3 = 2$ and $j_4 = 3$. Symbolically, $m_0 \xrightarrow{\sigma_4} m_4$. In alternative notation, $m_0[\sigma_4 > m_4$.

With every firing sequence σ_r, there exists an associated $(m \times 1)$ *Firing Count Vector*, which is a column vector (single column, multiple rows), $\vec{\sigma}_r$ whose elements correspond to number of times that particular transition has fired in that firing sequence σ_r. Since this vector keeps a count of the number of times a particular transition gets fired in a particular firing sequence, hence the term Firing Count Vector.

Example 5.2

In the previous example, corresponding to the firing sequence $\sigma_4 = t_1 t_3 t_2 t_3$, there exists an associated (7 x 1) Firing vector $\vec{\sigma}_4$, which is a column vector, given by $\vec{\sigma}_4 = (1120000)^T$, which means t_1 has fired only once, t_2 once, t_3 twice and t_4, t_5, t_6, t_7 never, in that firing sequence σ_4.

With the above understanding of firing count vector, now one can formally represent Firing Count vector as $\vec{\sigma}_r = (n_1 n_2 n_3 .... n_m)^T$; where n_k refers to the number of firings of t_k.

A somewhat subtle point is to be noted here. It was mentioned that with every firing sequence there exists an associated firing count vector. But this is not a one-to-one mapping, in the sense that, given a firing count vector, there exists more than one firing sequences. The reason of this is that firing sequence gives very precise information about which transitions are fired, how many

times they are fired and in which order; whereas firing count vector tells which transitions are fired and how many times they are fired – it does not talk about the order of firing. For the firing count vector $\vec{\sigma}_r = (n_1 n_2 n_3 .... n_m)^T$, there exists a total of N_S firing sequences where N_S is given by:

$$N_S = \frac{[n_1 + n_2 + ... + n_m]!}{[n_1! n_2! ... n_m!]} = \frac{(\sum_{i=1}^{m} n_i)!}{\prod_{i=1}^{m} (n_i!)}$$

Example 5.3

In the previous example, corresponding to the firing vector $\vec{\sigma}_4 = (1120000)^T$, one can associate a total of $N_S = \frac{[1+1+2]!}{[1!1!2!]}$ =12 firing sequences, $\sigma_4 = t_1 t_3 t_2 t_3$ being one among those twelve.

With all these definitions above, it is better to have a fresh look at the *reachability problem*. Equivalent to the earlier definition of reachability problem, one can give an alternative definition of Reachability Problem as one, where given a PN with marking m, given a marking m', it is needed to determine whether $m' \in R(m)$. A related problem which is slightly more general, is the *Coverability Problem*, defined as, given a PN with initial marking m_0, given a marking m', is there a reachable marking $m'' \in R(m_0)$, such that m'' covers m'? In order to answer these questions, it is required to exhaustively enumerate all the possible reachable marking by firing the enabled transitions one by one, starting with an initial marking. After each firing one will reach a new state (marking). When all the reachable states (markings) have been enumerated, then one can search for the desired state (marking). If it is present, then one can say that, the desired state (marking) is reachable from the initial marking by the PN. This process results in a tree representation of the markings. Each node of the tree represents marking generated by its parent marking and each arc represents a transition firing, which transforms one marking to another. Such a tree is called a Reachability Tree, $T = T(N, m_0)$.

Example 5.4

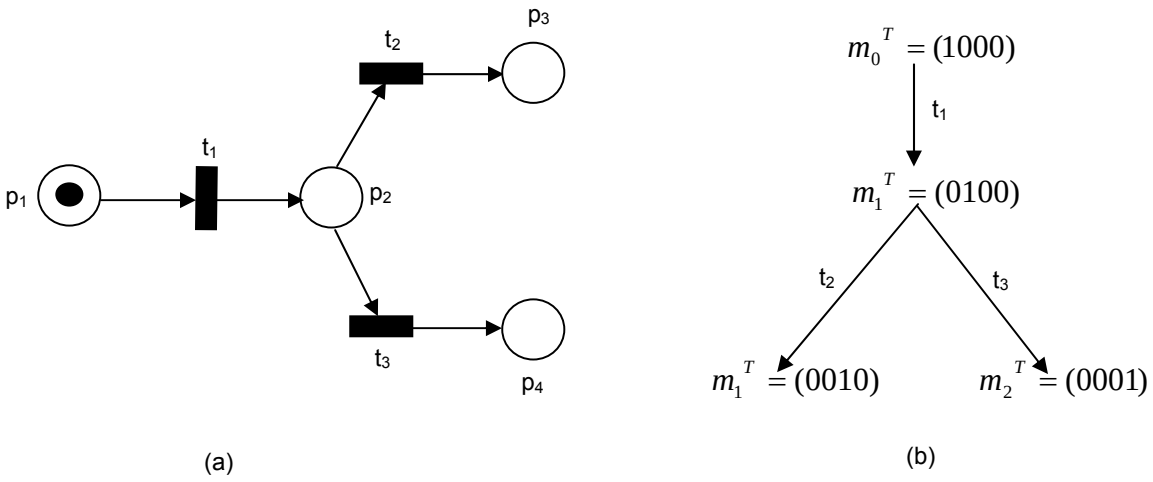


Figure 5.1: (a) Petri net graph, (b) reachability tree for (a)

In Fig. 5.1 a PN and its reachability tree is shown. It can be noted that transitions are marked along the arcs of the tree and conflict ($t_2 t_3$) can be effectively represented in the tree as shown.

It is, however, very much possible that the Reachability Tree could grow indefinitely. To circumvent this problem, the concept of 'pseudo-infinity' is introduced, so that the tree size can be kept finite. 'Pseudo-infinity', as the name suggests, can be thought of as a finite representation of infinity. 'Pseudo-infinity' is denoted by the symbol ω which is subject to the following properties: $\omega > a$, $\omega \geq \omega$, $\omega \pm a = \omega$; where a is any integer.

With the above concept of 'pseudo-infinity'(ω) one can always keep the tree finite. When the (ω) symbol is absent (truly finite size tree), then the term Reachability Tree is used. If, however, the (ω) symbol is present (truly infinite size tree, represented as finite size tree by introducing ω) then the term Coverability tree is used. This means the term Coverability Tree is more general. When there is no (ω) symbol present then the terms Reachability Tree and Coverability Tree are synonymous. Otherwise the term Coverability Tree should be used.

Now the question is, given a PN (N, m_0), how to construct the coverability tree. The following algorithm [2] is used for this purpose.

Algorithm:

Step 1: Label the initial marking m_0 as the 'root' and tag it 'new'.

Step 2: While 'new' markings exist, do the following:

Step 2.1: Select a new marking m ;

Step 2.2: If no transitions are enabled at m , then tag as m 'dead-end';

Step 2.3: If m is identical to a marking on the path from the 'root' to m , then tag m as 'old' and go to another new marking;

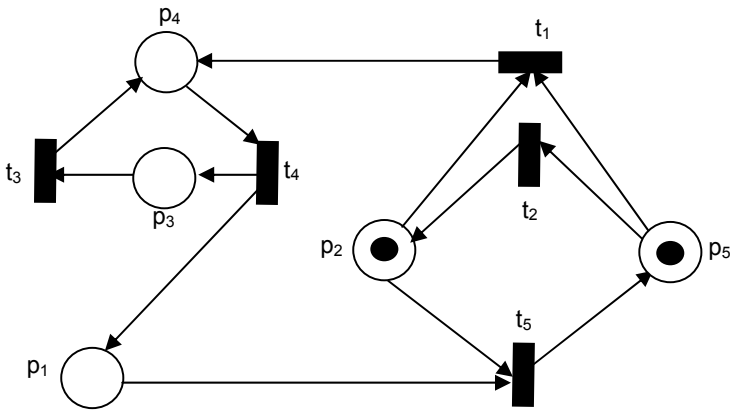
Step 2.4: While there exist enabled transitions at m , do the following for each enabled transition t at m :

Step 2.4.1: Obtain the marking m' by firing t at m .

Step 2.4.2: On the path from root to m , if there exists a marking m'' such that $m'(pi) \geq m''(pi) \forall i = 1,2,...,n$ and $m' \neq m''$ i.e. m' covers m'' , replace $m'(p_i)$ by ω wherever $m'(p_i) \geq m''(p_i)$:

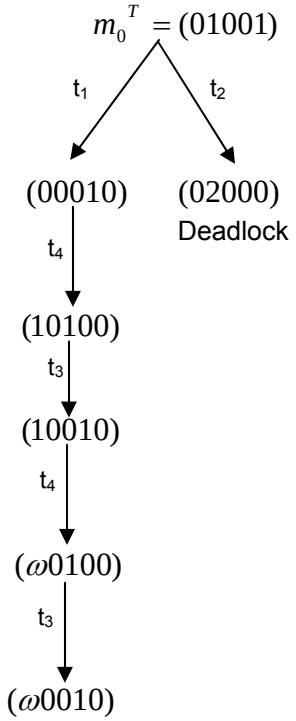
Step 2.4.3: Introduce m' as a node in the tree, draw an arc with label t from m to m' , and tag m' as 'new'.

Example 5.5



(a)

Example 5.5 (contd.)



(b)

Figure 5.2: (a) Petri net graph, (b) coverability tree construction for (a)

While constructing reachability(coverability) tree, the nodes of the tree corresponding to new markings are called *Frontier Nodes*, those corresponding to old markings are called *Duplicate Nodes* and those corresponding to markings having tag 'dead-end' are called *Terminal nodes*.

Often one is interested in the markings of a subset of places and does not bother about the rest of the places in the net. This approach leads to the so called *Submarking Reachability problem*. This problem aims to find if $\exists$ an $m' \in R(m_0)$, where m' is any marking whose restriction to a given subset of places agrees with that of a given marking m [2]. More formally, for $P' \subseteq P$ and a marking m', does there exist a $m'' \in R(N, m_0)$ such that $m''(p_i) = m'(p_i) \forall p_i \in P'$ [7]?

Submarking reachability problem is important for model checking and verification. One must give answer in the form that this, this, this markings should never be reached during system's operational life-span. The answer must be given as negation since only negation is conclusive [7].

Similarly, one can define *Zero Reachability Problem* which asks if the specific marking with zero tokens in all places, is reachable. Formally, is $m' \in R(N, m_0)$ with $m'(p_i) = 0 \forall p_i \in P$? (i.e. is $0 \in R(N, m_0)$?)

In a similar way, one can define *Single Place Zero Reachability Problem* which asks if it is possible to empty all the tokens out of a particular place. Formally, for a given place $p_i \in P$, does there exist $m' \in R(N, m)$ with $m'(p_i) = 0$?

In the context of reachability (coverability) problem, it can be a good idea to mention other important problems of PN. In PN context, the *Reducibility problem* is important. This problem addresses the questions like whether a given problem can be reduced to another problem, whose solution is already known. In the following example, reducibility problem is illustrated and *Equality* and *Subset Problems* are also introduced.

Example 5.6

Suppose it is desired to solve the *Equality problem* of PN, which says that, whether reachability sets of given Petri nets are equal or not. Formally, given two Petri nets N_1 and N_2 given by: $N_1 = (P_1, T_1, I_1, O_1, m_{01})$ and $N_2 = (P_2, T_2, I_2, O_2, m_{02})$; it is required to find whether $R(N_1, m_{01}) = R(N_2, m_{02})$. Another important problem is the *Subset Problem*, which seeks to determine whether $R(N_1, m_{01}) \subseteq R(N_2, m_{02})$ .Now the equality problem can be reduced to two subset problems since to show $R(N_1, m_{01}) = R(N_2, m_{02})$, it is sufficient to show that $R(N_1, m_{01}) \subseteq R(N_2, m_{02})$ and $R(N_2, m_{02}) \subseteq R(N_1, m_{01})$. Thus, the Equality Problem is reducible to Subset Problem. This illustrates the concept of reducibility.

In the previous discussion, many types of reachability problems were introduced. Instead of solving each of them separately, isn't it a nice idea if it can be shown that, at least some of them are reducible to other problems? The following figure shows which reachability problems are reducible to which problems.

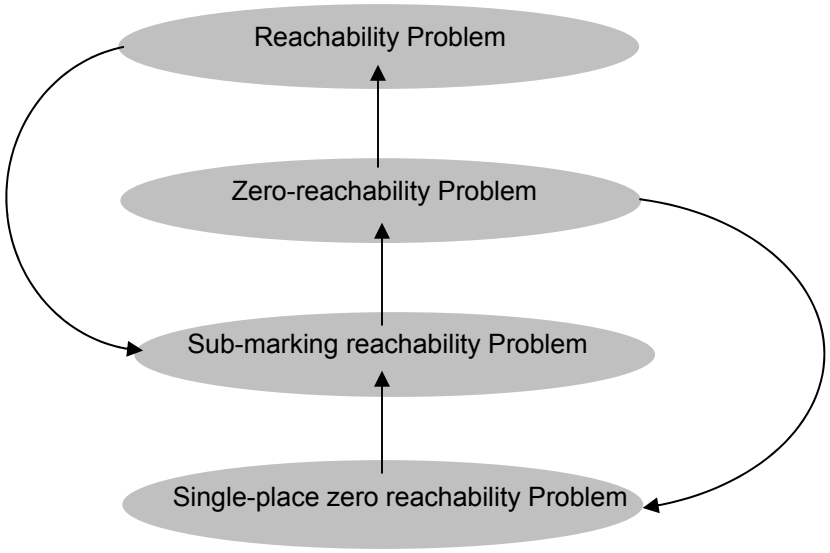


Figure 5.3: Reducibility among reachability problems [7]

From now on, this report will use the term reachability tree for non-existence of omega case and coverability tree for the trees where (ω) will appear. But, in a coverability tree, information is lost through the use of the symbol (omega). Hence, if someone enquires whether a particular state (marking) is reachable or not, then one may not be able to give any conclusive answer using coverability tree, since information was lost during introduction of the symbol (ω). *Thus the reachability test from coverability tree is inconclusive.*

5.2.1.2 Boundedness

A PN is called *k-bounded* with respect to an initial marking m_0, if each place in the net gets at most k tokens for all markings belonging to the reachability set $R(m_0)$, where k is a finite positive integer.

Mathematically, for Boundedness it should always happen that with respect to an initial marking m_0, $m(p_i) \le k \forall i \in [1, n]$ and this should happen $\forall m \in R(m_0)$.

If $k = 1$, then the PN is called Safe. Therefore, safeness is a special case of boundedness. In a safe PN, a place can either contain no token or it can contain only one token. In a safe net, there exists no such marking belonging to the reachability set (with respect to an initial marking) for which number of tokens in any place of the PN exceeds one.

Example 5.6

The PN in example 5.4 is unity bounded and hence safe (by definition). The PN in example 5.5 is unbounded. The coverability graph of Fig. 5.2 (b) shows unboundedness can be ensured from reachability tree from the appearance of ω in the tree.

Now it is worthy to think what can be the physical significance of a bounded or safe net. Places in the PN often represent buffers and registers which can store intermediate data. In this context, a bounded or safe net means there will be no data overflows in the buffers or registers, no matter what firing sequence is selected [2].

The reachability tree of an unbounded net will grow indefinitely and hence the symbol (omega) is introduced and coverability tree is constructed using the above stated algorithm. Thus, the very existence of the symbol (omega) in the tree means the PN is unbounded and the tree is coverability tree. Moreover, it also indicates which places of the PN are unbounded. Thus from reachability (coverability) tree one can make conclusions about boundedness (unboundedness) of the PN model, and hence of the actual system. *Thus the boundedness test from reachability (coverability) tree is conclusive.*

5.2.1.3 Liveness

A PN is called *Live* with respect to an initial marking, if for every marking belonging to the reachability set; it is possible to fire all the transitions at least once by some firing sequence.

Mathematically, a PN is called *Live* with respect to an initial marking m_0, if $\forall m \in R(m_0)$, it is possible to fire all the transitions at least once by some firing sequence.

The liveness property, as defined above, is a very strong property. However, it is impractical and too expensive to verify such a strong property for systems like operating system of a computer [2]. For this reason, the notion of liveness is relaxed by introducing degrees of liveness. In this approach, degrees of liveness of individual transitions are introduced and then the degrees of liveness of the entire PN are defined.

It is said that, a transition t of the PN, (N, m_0) is live at

Level 0: if t can never be fired. Then it is said that t is *L0 live* or t is *dead*.

Level 1: if t can be fired at least once in some firing sequence $L(m_0)$. Then t is *L1 live*.

Level 2: if, t can be fired any finite positive integral number of times in some firing sequence $L(m_0)$. Then t is *L2 live*.

Level 3: if $\exists$ at least one infinite length firing sequence $\in L(m_0)$, in which t appears infinitely often. Then t is *L3 live*.

Level 4: if t is L1 live $\forall m \in R(m_0)$. Then t is called *L4 live* or *live*. Note that *liveness* (i.e. *L4 liveness*) for a transition, just like liveness of a PN, is very strong property.

Just like degrees of liveness of a transition, now one can introduce degrees of liveness of the entire PN. A PN, (N, m_0) is said to be *Lk* (*Level k*) *live* if every transition in the net is Lk live, $k = 0,1,2,3,4$. Following this definition, a PN is said to be live (i.e. L4 live), if every transition in the net is live (i.e. L4 live). Now one can appreciate the statement: liveness of a PN is a very strong property. Degrees of liveness, is a relaxed property. In between these two concepts,

another idea called *Strict degrees of Liveness* is introduced, which as a property, is stronger than degrees of liveness but weaker than liveness. A transition is said to be *Strictly Lk live* if it is *Lk live but not L(k+1) live*, where $k = 1,2,3$. Similarly, a PN is called Strictly Lk live if it is Lk live but not L(k+1) live, $k = 1,2,3$.

Example 5.7

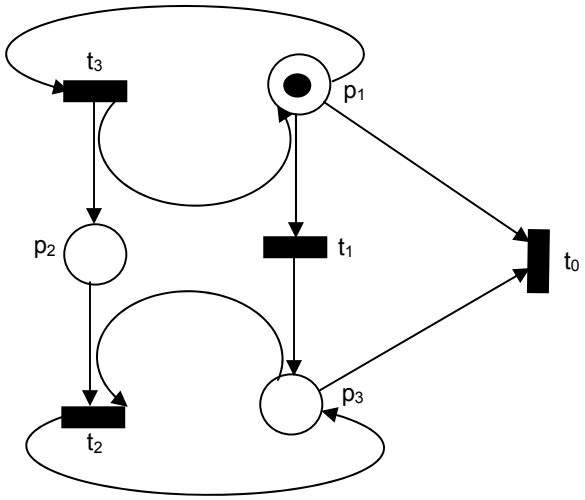


Figure 5.4: Transitions t_0, t_1, t_2, t_3 are L0 live (dead), L1 live, L2 live and L3 live respectively [2]

So far, liveness has been defined in terms of transitions. One can, however, define the liveness with respect to marking also [8]. To do so, one has to define live and dead transition, live and dead PN, and live, non-live and dead marking – all with respect to marking.

Live Transition: Let N be a PN and $t \in T$; t is said to be live iff $\forall m \in R(m_0)$ $\exists$ a marking $m' \in R(m_0)$, such that m' enables t or t is m' enabled.

Dead Transition: Let N be a PN and t(belongs to)T; t is said to be dead iff $\forall m \in R(m_0)$, t is not m -enabled.

Live PN: Let N be a PN. Iff $\forall t \in T$, t is live then the PN is said to be live.

Dead PN: Let N be a PN. Iff $\forall t \in T$, t is dead then the PN is said to be dead.

Live Marking: Let N be a PN. A marking $m' \in R(m_0)$ is said to be live iff $\forall t \in T$, t is m' enabled; where t is m' enabled means the transition t gets enabled by the marking m'. But as it is clear from the definition itself, live marking is a very strong condition.

Non-live Marking: Let N be a PN. A marking is called non-live if it is not live.

Dead Marking: Let N be a PN. A marking m is called dead iff no transition belonging to the set T, is m -enabled.

Corollary: A dead-marking is a non-live marking but a non-live marking may not be a dead marking.

It can also be perceived that

(1) all dead markings are deadends or terminal nodes but all deadends or terminal nodes are not dead markings.
(2) a term called *Dead-code* is often found in the literature, existence of which means a part of the net never gets executed (does not get any token) for a given initial marking. If the PN is modeling an algorithm or code, that part of the code always remains idle. Hence the name Dead-code.

After introducing the concept of liveness, now it's time to ask whether one can make conclusions about liveness from the reachability (coverability) tree. Because of the loss of information through the use of the symbol (omega), coverability tree can not give conclusive answer about liveness [2]. However, one can give conclusive answer as long as the net is bounded i.e. it is a reachability tree.

Example 5.8

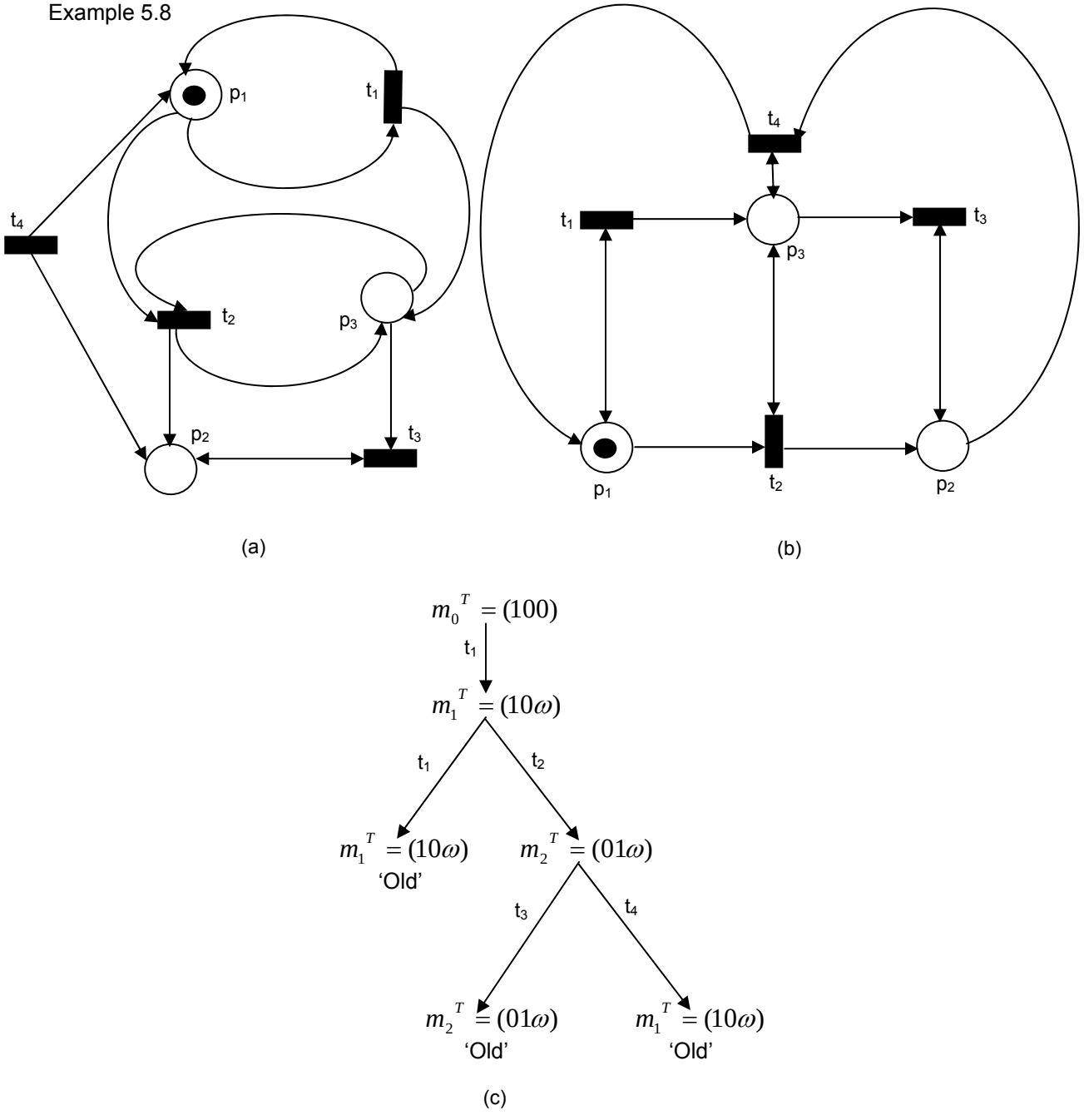


Figure 5.5: (a) A Live PN, (b) A non-live PN, (c) both (a) and (b) have same coverability tree (c) – hence liveness test from coverability tree is inconclusive [7]

5.2.1.4 Coverability

A marking m in a PN, (N, m_0) is said to be *Coverable* if $\exists$ a marking $m' \in R(m_0)$ such that $m'(pi) \geq m(pi) \forall i \in [1, n]$.

The concept of coverability is very closely related to the concept of potential firability or L1 liveness [2].

Suppose, m is the minimum marking needed to enable the transition t. Then t is dead (L0 live and not L1 live) iff m is not coverable. This means, t is L1 live iff m is coverable.

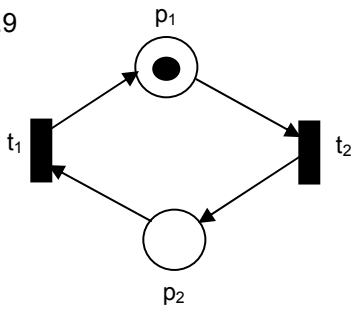
The definition of coverability itself shows that one can give conclusive answer about coverability as long as the PN is bounded i.e. it is a reachability tree. Following the same logic of reachability, the coverability test from coverability tree is inconclusive.

5.2.1.5 Reversibility and Home State

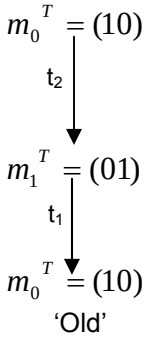
A PN is said to be *Reversible* or *Proper* if the initial marking is reachable from all reachable markings.

Mathematically, a PN is called *Reversible* or *Proper* if $\forall m \in R(m_0)$, m_0 is reachable from m. Equivalently, a PN is reversible if $\forall m \in R(m_0)$, $m_0 \in R(m)$.

Example 5.9



(a)



(b)

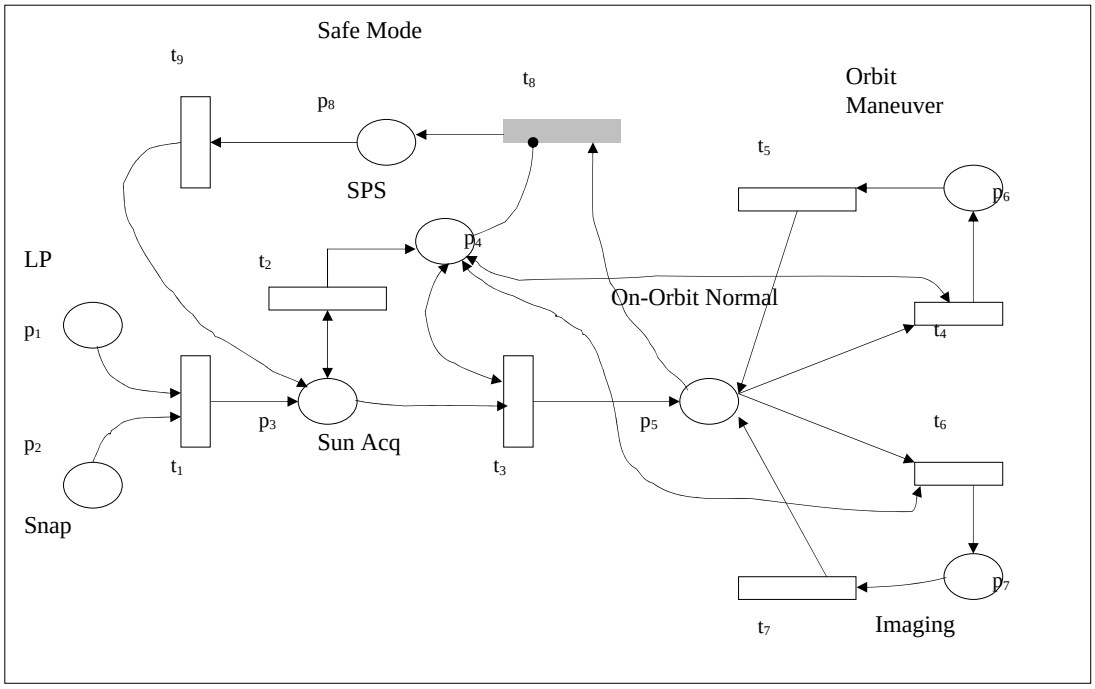
Figure 5.6: (a) A reversible PN, (b) the reachability tree of (a) shows that the net is reversible

In practical applications, it is often either not possible or not necessary to get back to the initial marking as long as one can get back to some marking covered by the initial marking. This particular marking (state) is called *Home Marking* or *Home State*. In this case also the PN is reversible. Thus when checking reversibility of a PN, the above stated definition of reversibility should not be followed blindly because the existence of home state was not taken into account in the definition. Formally, a marking m' is said to be a home marking or home state if, $\forall m \in R(m_0)$, m' is reachable from m.

A PN is reversible with respect to an initial marking m_0, iff every node in the coverability tree is in a directed circuit containing m_0. A PN is called Partially Reversible if a directed circuit containing m_0 includes only some of the nodes. Hence, reversibility test from the reachability (coverability) tree is conclusive.

Example 5.10

(a)



(b)

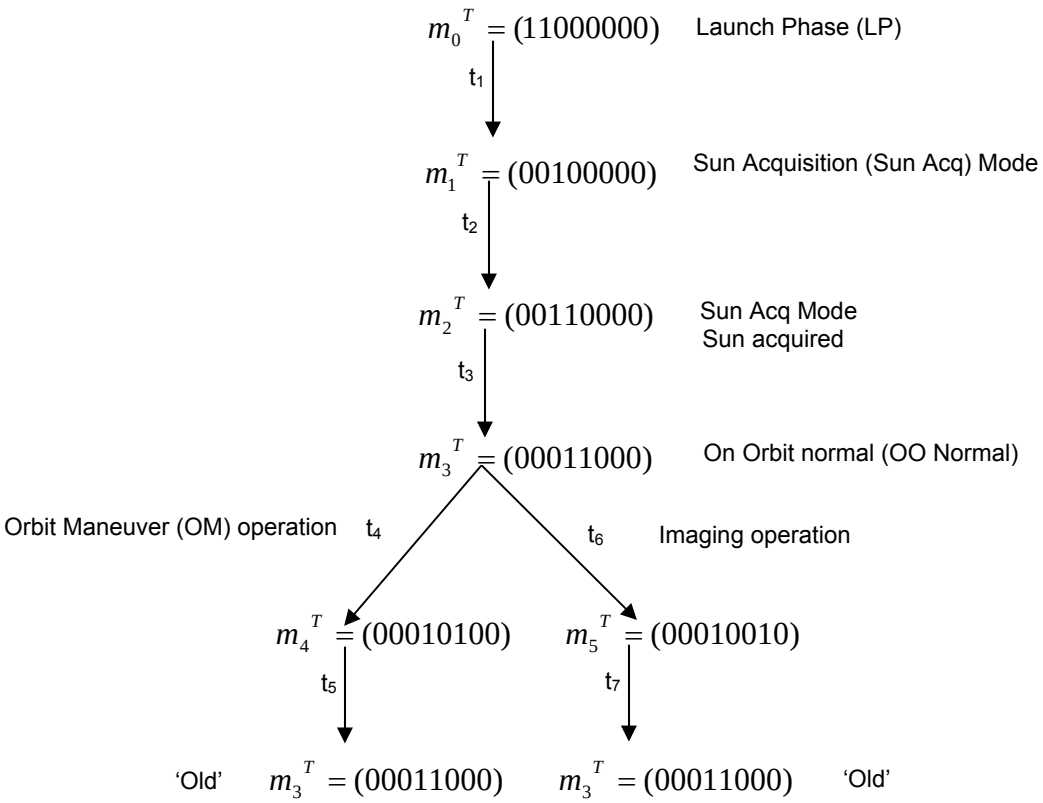


Figure 5.7: (a) AOCS PN graph, (b) Reachability tree shows initial marking m_0 is never attained since Launch Phase or SNAP can never be reached again. But m_3 becomes the home state and the PN (a) is reversible.

5.2.1.6 Repetitivity

A PN is *Repetitive* with respect to an initial marking m_0 iff the coverability graph has a directed circuit (not necessarily elementary) containing all the transitions infinitely often. It is *Partially Repetitive* if such a directed circuit contains only some of the transitions.

More formally, a PN, (N, m_0) is said to be (partially) repetitive with respect to an initial marking m_0 if $\exists$ a firing sequence σ such that every (some) transition occurs infinitely often in σ.

Alternatively, a PN, (N, m_0) is said to be (partially) repetitive with respect to an initial marking m_0 iff the coverability graph has a directed circuit (not necessarily elementary) containing all (some) the transitions infinitely often [9].

It can be mentioned at this point that *reversibility does not necessarily imply repetitivity and vice versa*. The definition itself clearly reveals that the repetitivity test from reachability (coverability) tree is conclusive.

5.2.1.7 Persistence

A PN is said to be *Persistent* if, for any two enabled transitions, firing of one transition will not disable the other. In other words, a PN is persistent if, for any $m \in R(m_0)$, an enabled transition can be disabled only by its own firing.

This means that, a transition in a persistent PN, once enabled, will remain enabled, until it fires. This implies, a PN having conflicts can not be persistent since the same place is an input to more than one transition. In a conflict situation an enabled transition can be disabled by the firing of other transitions and hence can never represent persistence. In a nutshell, persistent nets are always conflict-free nets.

The notion of persistence is important in the context of speed-independent asynchronous circuits and parallel program schemata [2].

5.2.1.8 Consistency

A PN is *Consistent* with respect an initial marking m_0 iff the coverability tree has a directed circuit (not necessarily elementary) containing all the transitions at least once [9]. It is *Partially Consistent* if such a directed circuit contains only some of the transitions.

The definition shows that consistency test from reachability (coverability) tree is conclusive.

5.2.1.9 Conservation

A PN is said to be *Conservative with respect to an initial marking* m_0, iff the weighted sum of the tokens in every node of the reachability tree remains constant. A PN is called *Strictly Conservative* if the sum of the tokens in the net always remains constant [9].

In simpler terms, a PN is conservative if it does not lose or gain tokens but merely moves them around. Now one may ask why the definition of conservation is given in terms of weighted sum of tokens rather than sum of tokens. The answer is that, in a PN two tokens can be encoded in one token and that single token can produce two tokens via the firing of a transition. This is the reason why a weighting vector is introduced to define the value of a token in each place; the weights are always non-negative.

Example 5.11

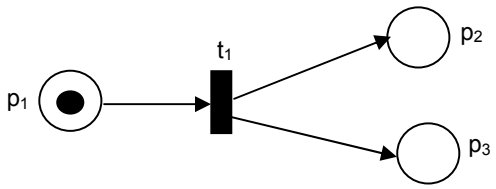


Figure 5.8: Although place p_1 contains a single token it has a potential to produce two tokens. Hence weight associated with place p_1 must be 2.

Conservation can be effectively tested using reachability tree. Since reachability tree is finite, the weighted sum can be computed for each marking. If this sum remains constant for all reachable markings, the net is conservative with respect to the given weight.

Thus to find out whether a net is conservative or not, first one has to construct the reachability tree with respect to a given initial marking m_0. Let w be a $(n \times 1)$ column vector (single column, n rows), called *weight vector*, where n is the number of places. The i^{th} entry of w is given by w_i which is nothing but the weight associated with the place p_i. If it happens that $w^T m$ = constant, where m is the marking of any node in the reachability tree with respect to an initial marking m_0, then one can conclude that the PN is conservative with respect to m_0. Since m_0 itself is a node in the reachability tree, instead of checking $w^T m$ =constant, one can check whether $w^T m = w^T m_0$, this time m being marking of any node in the reachability tree except the top most node m_0. More succinctly, conservation with respect to m_0 demands $w^T m = w^T m_0 \ \forall m \in R(m_0)$.

Note that nothing has been said so far about how to find w. The conclusion about conservation depends a lot on one's ability to find such a $(n \times 1)$ vector w, which satisfies the above stated condition. Up to this point the only thing that is known about w is that it has non-negative entries. But obtaining w by trial and error method seems an impossible proposition, particularly when n is large. Is there any systematic procedure which can generate such a weight vector? The matrix method based structural approach has an invariant analysis algorithm to determine this weight vector. In behavioral approach, one can use the condition $w^T m$ = constant and thus generate a set of k linear algebraic equations (k being the number of nodes in the reachability tree) in $(n+1)$ unknowns ($w_1, w_2, ..., w_n$ and the constant), subjected to the constraint $w_i \geq 0$; $i = 1,2,...,n$. This is a well-known linear programming problem having many algorithms for solution. So if a solution exists, it can be computed. The solution obtained from these techniques, in general, will be rational numbers. Multiplying them by a common denominator yields non-negative integral solution. If no such weighting vector exists, which can be conclusively told from these techniques, then the PN is not conservative.

One interesting situation comes when the net is unbounded i.e. the tree is coverability tree. Then the basic problem is how to incorporate (omega). The answer is that, if a marking has (omega) for a place pi, then the weight of that place must be zero for the PN to be conservative [7].

Thus the conservation test from reachability (coverability) tree is conclusive.

However, one can make an intelligent guess about the entries of the weight vector by recalling the fact that, the very need of introducing the weight vector is to indicate how many tokens are encoded in a particular token. Looking at each place and keeping the initial marking in mind one can make guess about how much weight should be assigned to a particular place. This can be done easily by thinking the PN as an infinite capacity one (each place can accommodate infinite number of tokens) and then mentally visualizing the number of tokens coming to each place. For example in Fig. 5.8, weight associated with place p_1 must be 2. The following example will further illustrate this technique.

Example 5.12

In Fig. 5.9 PN graph of a Flexible Manufacturing Cell [9] is shown. The tokens in this context represent resources (e.g. robots, conveyors etc.). One expects the PN of this system to be conservative from the resource limitation point of view. One way to find the weight vector is to construct the reachability tree and then for all the nodes of the tree form a set of linear algebraic equations based on $w^T m$ = constant subjected to an inequality constraint $w_i \geq 0$; $i = 1,2,...,n$. Then one can use linear programming techniques to get weight vector, which turns out to be $(123123123111)^T$.

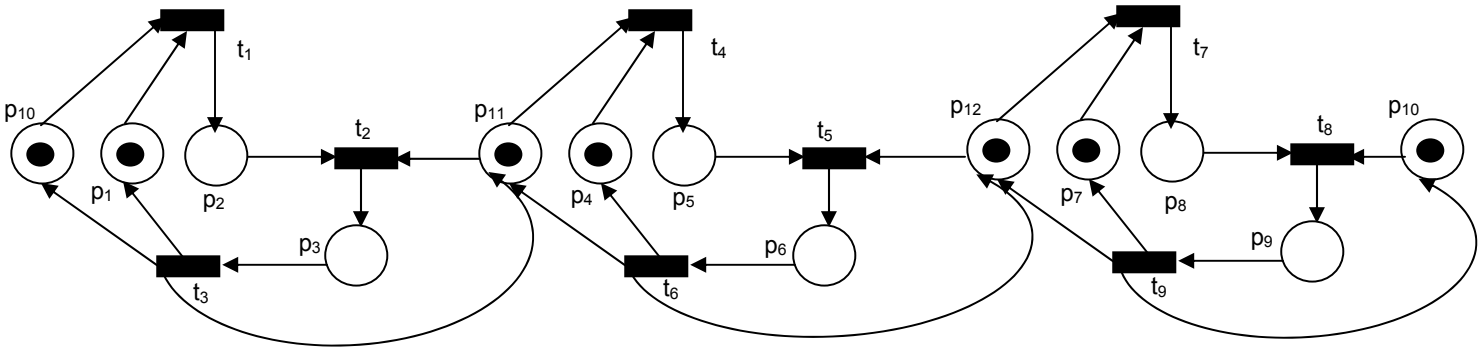


Figure 5.9: PN for a Flexible Manufacturing Cell – the PN should be conservative

Alternatively one can observe from Fig. 5.9 that the place p_1 has got the potential of depositing only one token to the output place p_2 via transition t_1. But two tokens can encoded in the place p_2 as it is the output place of two places p_1 and p_{10} via t_1. Again, p_3 has got three tokens encoded in it as it can get two (as explained) encoded tokens from p_2 and one encoded token from p_{11}. Hence one must assign weights 1, 2 and 3 to places p_1, p_2 and p_3 respectively. Similarly one can show that the weights associated with p_4, p_5 and p_6 are 1, 2 and 3. Also those with p_7, p_8 and p_9 are 1, 2 and 3. Similarly, weights associated with p_{10}, p_{11} and p_{12} can be shown to be 1, 1 and 1.

5.2.1.10 Synchronic Distance

Synchronic Distance measures the degree of mutual dependence between two transitions in a PN. This concept owes its origin to Carl Adam Petri.

Formally, *Synchronic Distance* (d_{12}) between two transitions t_1 and t_2 in a PN, (N, m_0) is defined as:

$$d_{12} = \max_{\sigma \in L(m_0)} |\#(t_1/\sigma) - \#(t_2/\sigma)|$$

where $\#(t_j/\sigma)$ denotes the number of occurrences of transition t_j in the firing sequence σ; $j = 1,2,...,m$. In short, by saying $d_{j_1\ j_2} = K$, one means that t_{j_1} can not fire more than K times, without firing t_{j_2} once (for all possible firing sequences over $R(m_0)$.

Example 5.13

In the PN shown in Fig. 5.10, $d_{12} = 1, d_{34} = 1$ and $d_{13} = \infty$.

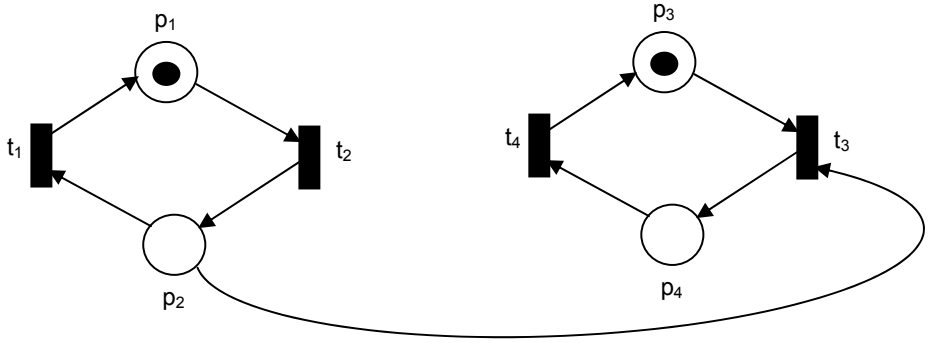


Figure 5.10: Example of Synchronic Distance

5.2.1.11 Fairness

Many different notions of fairness have been proposed in the literature. Here two basic fairness concepts are defined:

(1) Bounded Fairness (B- Fairness) and
(2) Unconditional (Global) Fairness.

Two transitions t_1 and t_2 are said to be in a *Bounded-fair (B-Fair)* relation if the maximum number of times that either one can fire while the other is not firing is bounded.

A PN, (N, m_0) is said to be a *B-Fair net* if every pair of transitions in the PN are in a B-Fair relation.

A firing sequence σ is said to be *unconditionally (Globally) Fair* if it is finite or every transition in the net appears infinitely often in σ.

A PN, (N, m_0) is said to be an *unconditionally (Globally) Fair net* if every firing sequence σ from $m \in R(m_0)$ is unconditionally fair.

Theorem: Every B-fair net is an unconditionally-fair net.

Theorem: Every bounded unconditionally fair net is a B-fair net.

Example 5.14

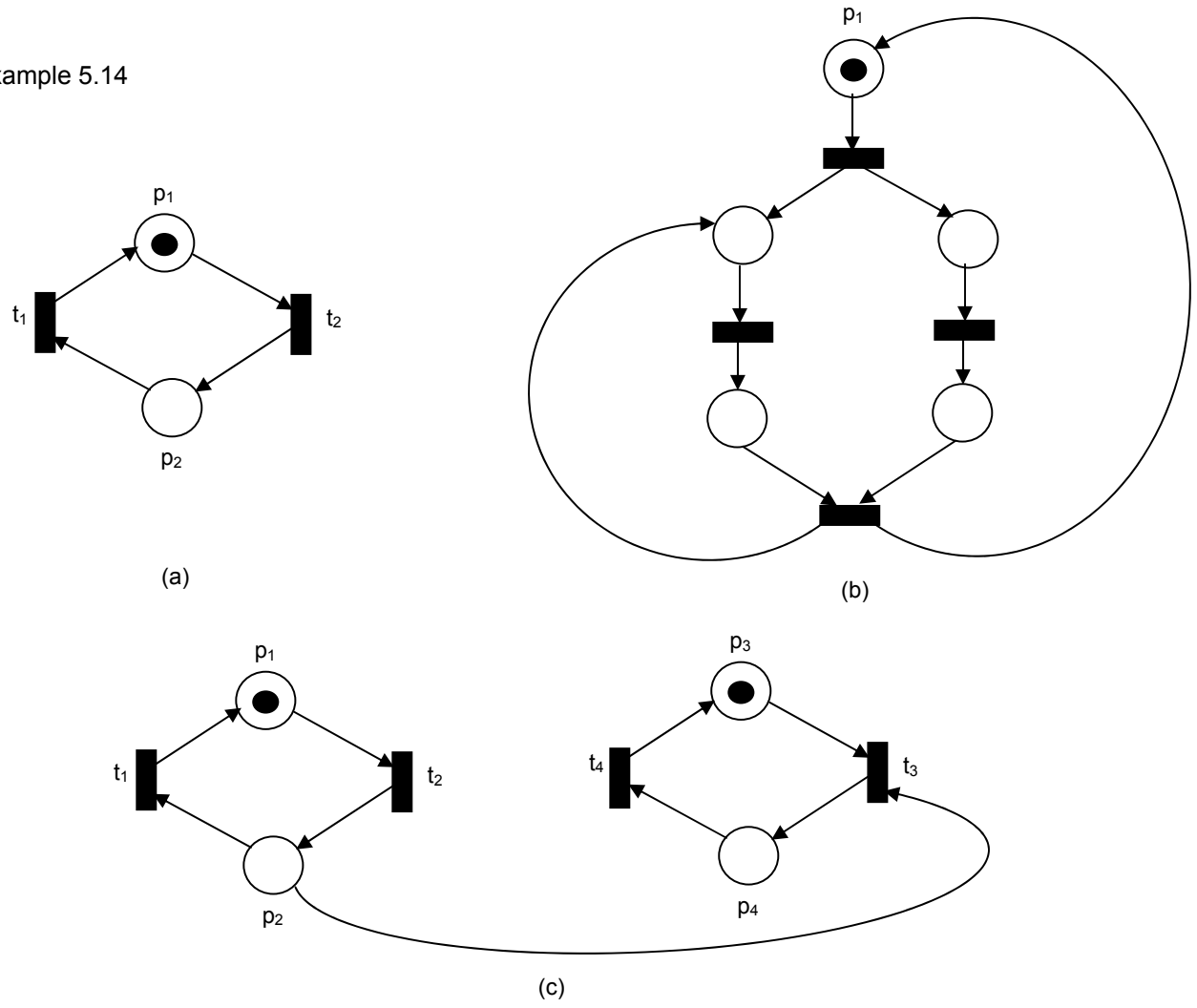


Figure 5.11: (a) B-fair as well as unconditionally fair net, (b) unconditionally fair but not B-fair net, (c) neither B-fair nor unconditionally fair net

Now one can make a comparative study of the behavioral properties whose conclusiveness can be tested using reachability (RT) and coverability tree (CT).

Properties	Conclusiveness of test from	
	Reachability Tree	Coverability Tree
Reachability	✓	
Boundedness	✓	✓
Liveness	✓	
Coverability	✓	
Reversibility	✓	✓
Repetitivity	✓	✓
Persistence	✓	✓
Consistency	✓	✓
Conservation	✓	✓
Synchronic Distance	✓	✓
Fairness	✓	✓

Table 5.1: Behavioral properties and their decidability using RT and/or CT

5.2.2 Structural approach

This approach deals with the structural properties of PN. Structural properties are independent of the initial marking. Behavioral properties are dynamic in nature, whereas structural properties deal with the static structure of the PN. Hence behavioral properties depend on markings and token game simulation but structural properties depend on nodes and arcs only and not on tokens and markings. In the following discussion, structural properties are treated in detail. Then effort has been given to make conclusions about them from matrix based structural analysis.

I. Motivation for Structural analysis

In systems and control theory, system description is given in terms of a set of differential or algebraic equations. Is it possible to give the Petri net system (static and dynamic) description in terms of some equations? That was the spirit that enabled the development of matrix equations for PN analysis. However, these matrix based method has limited power because of two reasons. First reason is the intrinsic non-determinism present in PN models, the second reason is that, unlike conventional control theory concepts, Petri nets pose one additional constraint that solutions must be non-negative integers. These two, togetherly make matrix based analysis somewhat weaker.

In all subsequent analysis, it will be assumed that the PN is pure (self-loop free) and if it is not then it is made pure by adding a dummy transition and a dummy place, before applying any matrix based method. This, of course, increases the matrix dimension.

II. Incidence Matrix

For a PN having n places and m transitions, the Incidence Matrix $A = [a_{ij}]$ is a $(n \times m)$ matrix of integers defined as:

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

where a_{ij}^{+} = weight of the arc from transition j to its output place i

a_{ij}^{-} = weight of the arc to transition j from its input place i

It can be recalled that, i is index for place (general place is p_i) and j is the index for transition (general transition is t_j). Also recall, $i = 1,2,...,n$ and $j = 1,2,...,m$. Physically, a_{ij}^{+}, a_{ij}^{-} and a_{ij}, respectively, signify the number of tokens added, removed and changed in i[th] place due to firing of j[th] transition.

Example 5.15

Consider the PN shown in Fig. 5.14 (a). Now by definition, $n =$ no. of places = no. of rows = 2 and $m =$ no. of transitions = no. of columns = 2. Hence dimension of the incidence matrix A is $(n \times m) = (2 \times 2)$. Now,

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

Since, $a_{11} = a_{11}^{+} - a_{11}^{-} =$ weight of the arc from t_1 to its output place p_1 - weight of the arc to t_1 from its input place $p_1 = 1 - 0 = 1$.

$a_{12} = a_{12}^{+} - a_{12}^{-} =$ weight of the arc from t_2 to its output place p_1 - weight of the arc to t_2 from its input place $p_1 = 0 - 1 = -1$.

$a_{21} = a_{21}^{+} - a_{21}^{-} =$ weight of the arc from t_1 to its output place p_2 - weight of the arc to t_1 from its input place $p_2 = 0 - 1 = -1$.

$a_{22} = a_{22}^{+} - a_{22}^{-} =$ weight of the arc from t_2 to its output place p_2 - weight of the arc to t_2 from its input place $p_2 = 1 - 0 = 1$.

III. State Equation

A state equation, as it is understood in control theory, is an equation, which can predict the next state (say $(k+1)$ [th] state) of the system, having known the present state (say, k [th] state). It will be a nice idea, if the same can be done in PN context.

Suppose a PN assumes a state (marking) m_{k+1} resulting from another state (marking) m_k by the k [th] firing (k [th] execution of the net), with $k \geq 0$. Here m_{k+1} is a $(n \times 1)$ column vector where the i [th] entry of m_{k+1} denotes the number of tokens in place i immediately after the k [th] firing. The k [th] firing is expressed as the k [th] *firing vector* or k [th] *elementary firing vector* or k [th] *control vector* u_k, which is an $(m \times 1)$ column vector where the j [th] entry of u_k denotes the number of times transition j fires during the k [th] execution of the net. Since during a particular firing, a particular transition can either not fire or fire only once, hence the elements of the vector u_k can be either 0 or 1; 0 in the position corresponding to the transition not fired and 1 corresponding to the transition fired during k [th] firing. Note that, in general, one can not say that in $(n \times 1)$ dimensional u_k vector, there will be only one non-zero entry 1 and rest $(n-1)$ zeroes, because in concurrent firing (if present) there will be multiple 1s and rest zeroes. Since the j [th] column of A denotes the change of marking as a result of firing transition j, the PN State Equation is given by [6]:

$$m_{k+1} = m_k + Au_k ; \quad k = 0,1,2,...$$

(1)

Since numbers of tokens are non-negative integers, the role of the vector u_k is to make the right hand side of the state equation a $(n \times 1)$ matrix of non-negative integers i.e. to make

$$m_k + Au_k \geq 0 ; \quad \forall k = 0,1,2,...$$

(2)

The above weak inequality can be used to test whether a given firing vector is legal or not, with respect to some marking m_k . In a sense, u_k controls the validity of the next state with respect to present state. It is reminiscent of the control vector in the state equation in control theory. Hence the name *control vector*.

Example 5.16

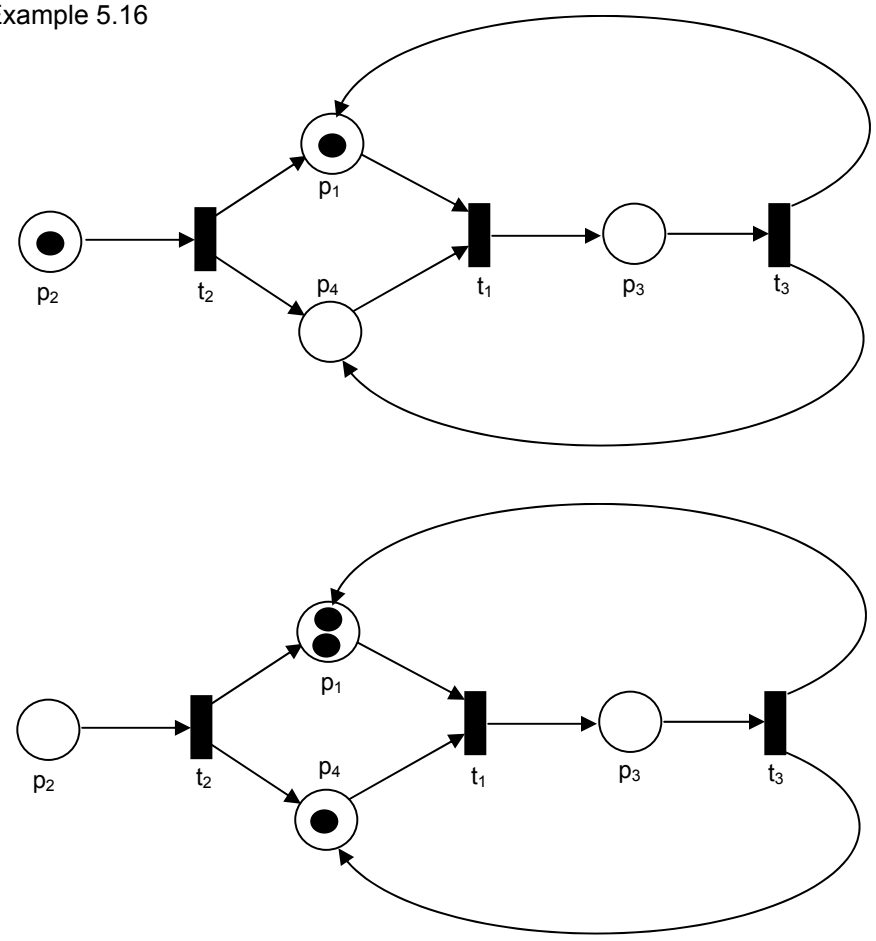


Figure 5.12: (a) A PN with initial marking, (b) that after firing t_2

If Fig. 5.12 (a) corresponds m_0 and (b) corresponds m_1 .This is achieved by firing t_2 only, is represented in the following state equation.

$$m_1 = m_0 + Au_0$$

$$\Rightarrow \begin{bmatrix} 2 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 & 1 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & -1 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

IV. Reachability: Necessary Condition

For reachability, it is desirable to have a formula to test if a given marking m_f (final marking) is reachable from a given initial marking m_0. Suppose a particular firing given by the firing vector u_0 brings the system from m_0 to m_1, u_1 brings m_1 to m_2, u_2 brings m_2 to m_3 ... u_{f-1} brings m_{f-1} to m_f. Mathematically, these are achieved by repeatedly using the state equation:

$$\begin{aligned} m_1 &= m_0 + Au_0 \\ m_2 &= m_1 + Au_1 \\ m_3 &= m_2 + Au_2 \\ &\cdots\cdots\cdots\cdots \\ m_f &= m_{f-1} + Au_{f-1} \end{aligned} \tag{3}$$

In short, final marking mf is reachable from initial marking m_0 through a firing sequence $\{u_0, u_1, u_2, ..., u_{f-1}\}$. Summing all the above equations:

$$\begin{aligned} m_f &= m_0 + A\sum_{k=0}^{f-1} u_k \\ \Rightarrow \Delta m &= A\vec{\sigma} \end{aligned} \tag{4}$$

Where $\Delta m = m_f - m_0$ and $\vec{\sigma} = \sum_{k=0}^{f-1} u_k$. Note that Δm is a $(n \times 1)$ column vector and $\vec{\sigma}$ is the sum of all the individual $(m \times 1)$ firing vectors or control vectors. $\vec{\sigma}$ is called *Firing Count Vector* (already introduced in Art 5.2.1.1), which is a vector of non-negative integers where the j^{th} entry of $\vec{\sigma}$ denotes the number of times transition j would fire in a firing sequence leading from m_0 to m_f.

There should not be any confusion between *Firing Vector* (or *Elementary Firing Vector* or *Control Vector*) and *Firing Count Vector*. Firing Count Vector is the sum of Firing Vectors. The nomenclature is a potential source of misconception. Confusion may arise when using some reference which does not introduce all the terms. The terms *Firing vector, Elementary Firing vector, Control vector* - all are same. But *Firing Count Vector* is different. Firing vector is associated with a single execution, which is an atomic event (hence the name Elementary Firing Vector) but Firing Count Vector is associated with firing sequence (though as mentioned in Art 5.2.1.1, this association is not one-to-one) composed of multiple executions.

The significance of equation (4) is that, for a given PN ($\Rightarrow$A is known), given an initial marking m_0 and a final marking m_f, if it is asked that whether m_f is reachable from m_0, then one

can give an answer. Unfortunately, this answer is not conclusive. If equation (4) results a non-negative integer solution for σ, then the desired marking may or may not be reachable – the test is inconclusive. But if no solution to equation (4) is found then the desired marking is not reachable – the negation is conclusive. Hence, this is a necessary but not a sufficient condition for reachability.

The inconclusive nature for reachability test that results form equation (4) may be due to the following reason. State equation in general and equation (4) in particular, is an equation which relates PN state (marking), which is a more dynamic property with incidence matrix, which is a representation of static PN structure. The state or marking captures both the statics and dynamics of the net whereas incidence matrix captures only the static net structure. Relating the two using a single equation is like using logical AND operator which results the loss of information about the dynamics of the net. That's exactly what has happened in equation (4), it has lost some information about net dynamics. Hence, having a non-negative integral firing count vector as a solution of equation (4) is an inconclusive answer.

V. Controllability

The beauty of Petri nets is that, one can extract essential control-theoretic ideas and adapt them in the context of Petri nets, thus enabling the system engineer to make important conclusions without providing lengthy mathematical manipulations. In control theory, and similarly in PN theory, the term controllability refers to the ability of inputs to change the state of the system. Thus a system is controllable means there exists a sequence of inputs which can steer the system from one state to the other. More formally one can say, a problem domain is completely controllable iff for every two state values in the state space of the problem representation, there exists a finite sequence of inputs (that some planner could produce) which will move the states from one value to the other (one state or marking to the other).

Equation (4) can also be written in the form of:

$$\Delta m = [AAA.....A]U \tag{5}$$

Where $U = [u_o{}^T u_1{}^T ... u_{f-1}{}^T]^T$; this vector U is a column vector of dimension $(mf \times 1)$ where the column is composed of only 1s and zeroes. In equation (5), the matrix $[AAA.....A]$ is composed of only As and it is a row vector having single row and f As. Since the dimension of each A is $(n \times m)$, the dimension of this matrix $[AAA.....A]$ is $(n \times mf)$. Borrowing ideas from system theory [Appendix B], the significance of equation (5) is that the matrix $[AAA.....A]$ is the controllability matrix in PN context. It can be shown [Appendix B] that, for a PN system to be completely controllable, the rank of the controllability matrix, and hence the rank of the incidence matrix, must be equal to the number of places in the PN.

The above condition for controllability is necessary and sufficient for the existence of a solution U of equation (5) over the field of rational numbers; but is necessary and not sufficient for U to be a vector of zeroes and 1s. Thus, in PN context the above condition for controllability remains only as a necessary condition.

If a problem domain is completely controllable, then for any state there exists a planner that can achieve any specified goal state. Very often complete controllability is not a property of the system, but it may possess a weaker form of controllability. This is the case in PN context too. The condition that rank of the controllability matrix (and hence that of the incidence matrix) equals number of places (in PN context) is as rarely satisfied as rank of the controllability matrix equals the state space dimension (in Control theory context). Therefore, a PN, in general, is not completely controllable just like in Control theory, systems are not completely controllable. This brings forth the concept of weaker controllability.

Moreover, both PN model and the actual plant are susceptible to disturbances. In PN context, disturbances appear as modeling inaccuracies and parameter variations just like in plant domain disturbances appear as noise in actuators, noise in physical system and noise in sensors.

Theorem: A necessary condition that a PN be completely controllable is rank of the incidence matrix equals to number of places in the PN. (Proof: Appendix B)

VI. Invariants

A. Place Invariant

A *Place-Invariant* or *P-Invariant* (also called *S-Invariant*) is defined as a $(n \times 1)$ non-negative integer vector x which satisfies the equation:

$$x^T A = 0 \tag{6}$$

Now it is required to explain the physical meaning of a P-invariant. For this let's pre-multiply both sides of the equation (4) by x^T:

$$\begin{aligned} x^T m_f &= x^T m_0 + x^T A \sigma \\ \Rightarrow x^T m_f &= x^T m_0 \text{ (Since by definition, } x^T A = 0 \text{)} \end{aligned} \tag{7}$$

The expression in equation (7) is nothing but the conservation condition given in equation (1). Both these equations are looking for a vector such that the vector transpose times final (desired) marking equals the vector transpose times initial marking. This clearly reveals that the P-invariant vector x is nothing but the weight vector w associated with places of the PN; hence the name Place-invariant or P-invariant.

B. Transition Invariant

A *Transition-Invariant* or *T-Invariant* is defined as a $(m \times 1)$ non-negative integer vector y which satisfies the equation:

$$Ay = 0 \tag{8}$$

Now it is required to explain the physical meaning of a T-invariant. For this let's post-multiply both sides of the equation (4) by y:

$$\begin{aligned} m_f y &= m_0 y + A y \sigma \\ \Rightarrow m_f &= m_0 \quad \text{(Since by definition, } Ay = 0 \text{)} \end{aligned} \tag{9}$$

The expression in equation (9) gives a definition of T-invariant with physical meaning. A vector y of non-negative integers is a T-invariant iff there exists a marking $m = m_0 = m_f$ and a firing sequence back to m whose firing count vector is y.

C. Invariant Representation

P and T-invariants can be represented in two ways:

(1) Vector representation

P-invariant is represented as a $(n \times 1)$ vector given by $x = [x_{p_1} x_{p_2} ... x_{p_i} ... x_{p_n}]^T$.

T-invariant is represented as a $(m \times 1)$ vector given by $y = [y_{t1} y_{t2} ... y_{tj} ... y_{tm}]^T$.

(2) Set representation

P-invariant is represented as $\| x \| = \{p_1, p_2, ... p_k\}$ where $k \leq n$; this set includes only those places as elements of the set, which have non-zero weights.
T-invariant, in a similar fashion, can be represented as $\| y \| = \{t_1, t_2, ... t_q\}$ where $q \leq m$; this set includes only those transitions as elements of the set, which have non-zero firing occurrences.

D. Minimal or Basic Invariant

A *Minimal* or *Basic Invariant* is one which is not a linear combination of other invariants.

More formally one can define minimal P and T invariants as follows. Let $\| x \| = \{p_1, p_2, ... p_k\}$ be the set representation of a P-invariant of a PN. Then $\| x \|$ is called a *minimal P-invariant* if $\exists$ a $\| x' \|$ such that $\| x' \| \subset \| x \|$, where $\| x' \|$ is another net invariant. Similarly one can define *minimal T-invariant*. Since a linear combination of minimal T-invariants correspond a firing sequence which takes a marking back to itself, *Minimal T-invariants are also called Reproduction Vectors*.

Having defined the minimal or basic invariants, now there are two major questions that remain to answer. First, given a PN, how many basic invariants are possible and secondly, how to determine them. Answer to the second question is relatively easy to perceive. Once the set of solutions of equation (6) or equation (8) (depending on whether one is interested in minimal-P or minimal-T invariants) is obtained, by factoring out the GCD (greatest common divisor) minimal or basic invariants can be computed. However, the answer to the first question is not so obvious.

Theorem: A PN has $(n - r)$ minimal P-invariants and $(m - r)$ minimal T-invariants, where r is the rank of the incidence matrix A.

Proof: Part (1) A PN has $(m - r)$ minimal T-invariants.

T-invariants are defined to be solutions of equation (8): $Ay = 0$. Let A be partitioned as:

$$A = \begin{Bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{Bmatrix}$$

Where, A_{11} is $r \times (m - r)$, A_{12} is $r \times r$, A_{21} is $(n - r) \times (m - r)$ and A_{22} is $(n - r) \times r$. This partition has been done by rearranging the columns of A such that A_{12} has r independent columns of A, i.e. A_{12} is a non-singular square matrix of dimension $(r \times r)$.

Now equation (8) can be written as:

$$A_{11}y_1 + A_{12}y_2 = 0$$
$$A_{21}y_1 + A_{22}y_2 = 0$$

(10)

Where, y_1 is of dimension $(m-r)\times 1$ and y_2 is of dimension $(r\times 1)$. Since A_{12} is nonsingular and hence invertible, the first equation of (10) can be written as:

$$y2 = -A_{12}{}^{-1}A_{11}y_1$$

(11)

Substituting (11) in the second equation of (10):

$$(A_{21} - A_{22}A_{12}{}^{-1}A_{11})y_1 = 0$$

(12)

The co-efficient of y_1 in equation (12) is nothing but the schur complement of A_{21} [Appendix A]. Now from equation (11) it is evident that, the solution of equation (8) and hence the system of equations (10) is given by:

$$y = \begin{Bmatrix} I \\ -\mathrm{A}_{12}{}^{-1}\,\mathrm{A}_{11} \end{Bmatrix}$$

(13)

Where, I is an $(m-r)$ dimensional identity matrix. Clearly, $y^T = [I, -A^T{}_{11}(A^T{}_{12})^{-1}] = B_t$ (say). Generally the basic T-invariants are given in terms of B_t, which actually is the transpose of basic T-invariant. One can observe that B_t has $(m-r)$ rows. Hence the PN contains $(m-r)$ basic T-invariants.

Check:

$$Ay = AB_t{}^T = \begin{Bmatrix} A11 & A12 \\ A21 & A22 \end{Bmatrix} \begin{Bmatrix} I \\ -\mathrm{A}_{12}{}^{-1}\,\mathrm{A}_{11} \end{Bmatrix} = \begin{Bmatrix} A_{11} - A_{12}A^{-1}{}_{12}A_{11} \\ \mathrm{A}_{21} - \mathrm{A}_{22}\mathrm{A}^{-1}{}_{12}\mathrm{A}_{11} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}$$ (using equation (12))

Hence the solution is verified. Note that A and B_t are orthogonal to each other.

Part (2) A PN has $(n-r)$ minimal P-invariants.

P-invariants are defined to be solutions of equation (6): $x^T A = 0 \Rightarrow A^T x = 0$. As it was done in case of T-invariants, similar partitioning yields:

$$A^T{}_{11}y_1 + A^T{}_{12}x_2 = 0$$
$$A^T{}_{21}x_1 + A^T{}_{22}x_2 = 0$$

(14)

Since $A^T{}_{12}$ is invertible,

$$x1 = -(A^T{}_{12})^{-1}A^T{}_{22}x_2$$

(15)

Consequently,

$$(A^T{}_{21} - A^T{}_{11}(A^T{}_{12})^{-1}A^T{}_{22})x_2 = 0$$

(16)

The co-efficient of x_2 in equation (16) is nothing but the schur complement of $A^T{}_{21}$ [Appendix A]. Now from equation (15) it is evident that, the solution of equation (6) and hence the system of equations (14) is given by:

$$x = \begin{Bmatrix} -(A^T{}_{12})^{-1} A^T{}_{22} \\ I \end{Bmatrix}$$

(17)

Where, I is an $(n-r)$ dimensional identity matrix. Clearly, $xT = [-A_{22}A^{-1}{}_{12}, I] = B_p$ (say). Generally the basic P-invariants are given in terms of B_p, which actually is the transpose of basic P-invariant. One can observe that B_p has $(n-r)$ rows. Hence the PN contains $(n-r)$ basic P-invariants.

Check:

$$x^T A = A^T x = A^T B_p{}^T$$

$$= \begin{Bmatrix} A^T{}_{11} & A^T{}_{21} \\ A^T{}_{12} & A^T{}_{22} \end{Bmatrix} \begin{Bmatrix} -(A^T{}_{12})^{-1} A^T{}_{22} \\ I \end{Bmatrix} = \begin{Bmatrix} A^T{}_{21} - A^T{}_{11}(A^{-1}{}_{12})^T A^T{}_{22} \\ 0 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}$$

(Using equation (16)). Hence the solution is verified.

Note that, the state equation (equation (4)) is given by: $A\vec{\sigma} = \Delta m$, which is an inhomogeneous system of equations. It is a well-known fact in linear algebra that this inhomogeneous system has a solution $\vec{\sigma}$ iff Δm is orthogonal to every solution x of the homogeneous system $A^T x = 0 \Rightarrow x^T A = 0$. Therefore, the existence of a solution for $\vec{\sigma}$ demands $x^T \Delta m = 0 \Rightarrow B_p \Delta m = 0$ .This gives an alternative statement (alternative to equation (4)) for necessary condition of reachability.

E. Trivial invariant

An invariant vector with all its elements equal to zero (or the null set, in the set representation) is called a *trivial invariant*.

F. Non-trivial invariant

An invariant is said to be *non-trivial* if at-least one element of the vector is non-zero (a non-empty set).

G. Purely non-trivial invariant

An invariant in which all the elements of the vector are non-zero (in set representation, all the places for a place invariant and all the transitions for a transition invariant are included in the set) is said to be a *purely non-trivial invariant* [5].

H. Computation of Invariants

In the above section the fact that a PN has $(n-r)$ minimal or basic P-invariants and $(m-r)$ minimal or basic T-invariants has been established. Now the question is how to calculate the invariants. As per the definition of P and T-invariants, calculating invariants means

solving equation (6) and (8). These two seemingly innocent equations are not so easy to solve because of the constraint in the invariant definition that, the elements of solution vectors are non-negative integers.

At first glance, readers may get the misconception that without going for solving the equation (6) and (8) with constraint, why not directly partition A and get B_p and B_t in terms of partitioned matrices. It should be clearly understood that partitioning A and calculating B_p and B_t, in general, does not result in solution vector whose elements are non-negative integers. In fact, generally such a practice yields B_p and B_t having negative and/or fractional elements. This is clearly unacceptable. In a nutshell, calculating B_p and B_t may be used as a short-cut method to find invariants. If someone is fortunate enough, then one may get an invariant having all non-negative elements. Otherwise, in the general case, one has to use one of the following algorithms. The objective of computation is to find the minimal set of invariants – those invariants which are linearly independent.

Martinez-Silva Algorithm

Martinez and Silva [10] have given a Gauss elimination-like algorithm to calculate the invariants of a PN. This section describes the algorithm for finding minimal set of P-invariants. How the same can be used to find the minimal set of T-invariants will be mentioned subsequently. It may be helpful to recollect the fact that a PN has $(n-r)$ minimal P-invariants and $(m-r)$ minimal T-invariants.

Algorithm:

Recall that the basic equation for finding P-invariants is $x^T A = 0$.

Step 1: Append an $(n \times n)$ identity matrix to A to generate $[A : I]$.

Step 2: Nullify the i^{th} column of $[A : I]$ by adding any two rows of $[A : I]$.

Step 3: Iterate for $j = 1,2,...,m$ (stop when first m columns are nullified).

Step 4: Take out the un-nullified columns (starting at the $(m+1)^{\text{th}}$ column, leaving first m nullified columns).

Step 5: Rank of this resultant matrix is $(n-r)$. Hence $(n-r)$ linearly independent rows are minimal P-invariants.

Note that, there is no guarantee that this algorithm will find only the minimal invariants. Hence no one can guarantee that the resultant matrix has only $(n-r)$ rows. Even if it has more than $(n-r)$ rows then by inspection one can find $(n-r)$ linearly independent rows. This is where, knowing apriori the fact that there will be $(n-r)$ minimal P-invariants, help.

When it is desired to find minimal T-invariants then the basic equation to be solved is $Ay = 0 \Rightarrow y^T A^T = 0$. With this form, it is very much similar to the basic equation for P-invariants except the incidence matrix is now got transposed. Thus one can now apply Martinez-Silva algorithm as stated above. Only at Step 3, this time one has to iterate for $i = 1,2,...,n$ (nullify the i^{th} column). Rest of the algorithm is similar to the above discussed P-invariant case.

Example 5.17

Consider the PN of example 5.15. The incidence matrix is $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$. Now it is required to apply Martinez-Silva algorithm.

Step 1: $[A:I] = \begin{bmatrix} 1 & -1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{bmatrix}$.

Step 2 and
Step 3: $\begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix}$.

Step 4: $\begin{bmatrix} 1 & 1 \end{bmatrix}$.

Step 5: Rank of this resultant matrix is $(2-1)=1$. Hence only one linearly independent row is minimal P-invariant.

Hence the minimal P-invariant is given by $\{x\}_{2\times 1} = \begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$. Similarly one can find minimal T-invariants.

Computation of suitable invariants that help in providing various properties of the PN, and hence of the physical system are of particular interest. However, finding invariant is nothing but solving a set of simultaneous equations under inequality constraint, which is a tedious job, especially when the number of equations is quite large. Also in most practical situations, hardly one can find a unique set of invariants. The identification of the desired set of invariants from all possible ones is a complicated task [5]. The Martinez-Silva algorithm discussed above suffers from the disadvantage that it considers the entire set of equations and hence the entire net to compute the invariants. This may become a daunting task when the system is very complex and the PN is of very large size with complex interactions. Moreover, Martinez-Silva algorithm obtains minimal invariants. But minimal invariants may not be the desired invariants for proving properties of the PN. Hence it is still a challenging task to compute what combination of minimal invariants will give desired invariants. V. K. Agrawal [5] has proposed a simpler technique to find the desired invariants. In this approach, the invariant is found as described below:

Step 1: Define a subnet called Restricted P-Subnet (RPSN) (as defined in Art 4.2.11).
Step 2: Select a RPSN such that the selected RPSN can have an invariant which is also an invariant of the original PN.
Step 3: The selected RPSN is reduced to a smaller one using the proposed reduction rules [5] and then the reduced subnet is analyzed.

However, the selection of the subnet is governed by the properties to be proved. Since the subnet, in general, contains a subset of the entire set of places and transitions, the number of nodes required to deal with is much smaller. Then it is much easier to find the invariant for this subnet either by solving simultaneous equations under inequality constraint or by the algorithm proposed in [5]. The proposed technique is also useful for finding the spanning set of invariants for the entire net which covers all the places of the net. This report does not deal with the detailed intricacies of this technique. The above discussion serves only as an overview of how an alternate method can address the invariant computation problem. For a detailed discussion, one can refer to [5].

I. Support of an Invariant

The *Support of an invariant* is a set of nodes (places or transitions depending on support of P-invariant or support of T-invariant) whose corresponding component in the invariant is positive.

The support of a P-invariant x is denoted by $\langle x \rangle$ which is a set of places whose corresponding components in the vector x is positive. Similarly, the support of a T-invariant y is denoted by $\langle y \rangle$ which is a set of transitions whose corresponding components in the vector y are positive.

J. Minimal Support

The support of an invariant is called *minimal* iff it does not contain the support of another invariant but itself and the empty set [9].

Example 5.18

The minimal support for the P-invariant x in example 5.17 is given by: $\langle x \rangle = \{p_1, p_2\}$.

Theorem: Let z_1 and z_2 be two invariants of same kind (either both P-invariant or both T-invariant) then:

(1) For non-negative integers a and b, $(az_1 + bz_2)$ is an invariant.

(2) If $(z_1 - z_2)$ has no non-negative elements, then it is an invariant.

(3) $\langle z_1 + z_2 \rangle = \langle z_1 \rangle + \langle z_2 \rangle$.

The first statement, in the context of T-invariants, means that the firing sequence corresponding to a support resulting from a linear combination of minimal T-invariants (which by this theorem is also a T-invariant) will result a marking reproducing itself (a firing sequence starting a marking back to itself). Hence minimal T-invariants are also called Reproduction Vectors [9].

Theorem: Let a pure PN is bounded with respect to an initial marking m_0. Let the place p_i belong to the support of a P-invariant. Let $l_1, l_2, ..., l_j$ be the minimal supports containing p_i and let $x_1, x_2, ..., x_j$ be the minimal P-invariants associated with the minimal supports. Then for any reachable marking $m \in R(m_0)$, $m(p_i)$ is upper-bounded by:

$$m(p_i) \leq \min_{l=1,2,...,j}[(m^T{}_0 x_l)/(x_l(p_i))]$$

The above statement is important from PN boundedness property point of view. This theorem enables one to determine the k-boundedness of a PN from the P-invariants. It can be noted that if a net is covered by P-invariants and m0 is bounded, then the net is bounded.

VII. Structural Properties

This section will describe the structural properties and how to determine them using incidence matrix.

5.2.2.1 Structural Boundedness

A PN is said to be *Structurally Bounded* if it is bounded by any finite initial marking.

Thus structural boundedness is more general case compared to behavioral boundedness which is defined with respect to a particular marking. Thus if one can show a PN is structurally bounded, it implies the PN has behavioral boundedness.

Example 5.19

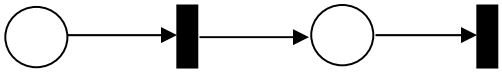


Figure 5.13: The PN shown above is structurally bounded

Theorem: A PN is structurally bounded iff $\exists$ a $(n \times 1)$ vector x of positive integers such that $x^T A \leq 0$. (Proof can be found at [2])

A place p in a PN is said to be *structurally unbounded* if $\exists$ a marking m_0 and a firing sequence σ from m_0 such that unbounded.

Corollary: A place p in a PN is structurally unbounded iff $\exists$ an m -vector y of non-negative integers such that $Ay = \Delta m >\neq 0$, where $x >\neq y$ means $x \geq y$ and $x_i \neq y_i$ for some i .

5.2.2.2 Structural Liveness

A PN is Structurally Live if it is live for any initial marking m_0 .

Example 5.20

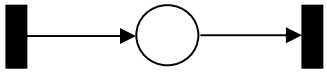


Figure 5.14: The PN shown above is structurally live but the one in Fig. 5.13 is structurally not live

5.2.2.3 Structural Conservation

A PN is Structurally Conservative if it is conservative for any initial marking m_0 .

Theorem: A PN is structurally conservative (partially structurally conservative) iff $\exists$ a $(n \times 1)$ vector x of positive (non-negative) integers such that $x^T A = 0$.

Example 5.21

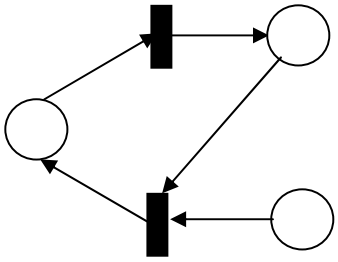


Figure 5.15: The PN shown in Fig. 5.14 (a) is structurally conservative.
But the one shown above is partially structurally conservative but not structurally conservative

Partial structural conservativeness condition demands the existence of a P-invariant. Hence a partial structural conservative PN is also called a P-invariant net or S-invariant net. It can be noted that structural conservativeness is a special case of structural boundedness.

5.2.2.4 Structural Repetitivity

A PN is *Structurally (partially) Repetitive* if it is repetitive for at least one finite initial marking.

Theorem: A PN is (partially) structurally repetitive iff $\exists$ a $(m \times 1)$ vector y of positive (non-negative) integers such that $Ay \geq 0$ and vice versa. (Proof can be found at [2])

Example 5.22

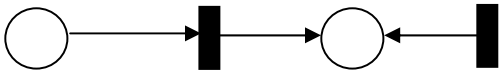


Figure 5.16: The PN shown above is partially structurally repetitive but not structurally repetitive

5.2.2.5 Structural Consistency

A PN is *Structurally (partially) consistent* if it is consistent for at least one finite initial marking.

Formally a PN is said to be structurally (partially) consistent if $\exists$ a finite initial marking m_0 and a firing sequence σ from m_0 back to m_0 such that every (some) transition occurs at least once in σ.

Example 5.23

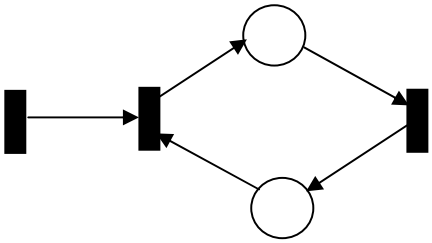


Figure 5.17: The PN shown above is partially structurally consistent but not structurally consistent

The net shown in Fig 5.14 is structurally consistent but the one shown above is structurally partially consistent but not structurally consistent.

Theorem: A PN is (partially) structurally consistent iff $\exists$ a $(m \times 1)$ vector y of positive (non-negative) integers such that $Ay = 0$. (Proof can be found at [2])

Partially structurally consistent nets are also called T-invariant nets. It can be noted that consistency is a special case of repetitiveness.

Theorem: If a PN is structurally bounded and structurally live then it is both structurally conservative and structurally consistent.

5.2.2.6 Complete Controllability

A PN is said to be Completely Controllable if from any state (marking), any initial state (marking) can be reached. Controllability is discussed in detail in Art 5.2.2 V.

Example 5.24

The net shown in Fig 5.14 is completely controllable.

5.2.2.7 Structural B-Fairness

Two transitions are said to be in a *Structural B-Fair relation* if they are in a B-fair relation for any initial marking.

A PN is said to be *structurally B-fair* if it is a B-fair net for any initial marking.

Example 5.25

(a) (b)

Figure 5.16: (a) Structurally B-fair PN, (b) Not Structurally B-fair PN.

Murata and Silva [11] have given the following results on structural B-fairness.

i. A structural B-fair relation (as well as a B-fair relation) on the set of transitions T is an equivalence relation and thus partitions T into equivalence classes.
ii. Structural B-fairness implies B-fairness but the converse is not true [2].
iii. A structurally bounded net is structurally B-fair iff
Either it is consistent and there is only one reproduction vector (minimal non-negative T-invariant $y \neq 0$.
Or it is not consistent and there is no reproduction vector.

Conclusion

This report described the basic Petri net theory in five chapters. Chapter 1 served as an introduction. Chapter 2 introduces the basic nomenclature and representation symbols in the context of Petri net and formally defines Petri nets and its various fundamentals. Chapter 3 is dedicated exclusively for modeling. Chapter 4 defines various sub-structures of Petri net which may ease the modeling and analysis of a complex net. Chapter 5 addresses the analysis aspect. Various analysis approaches and corresponding mathematical formulations are given. However for the sake of conciseness and less popularity among engineering fraternity, the reduction method of analysis is not discussed in Chapter 5.

It should be emphatically mentioned that there are many other aspects of Petri net which has not been addressed in this report. Two such important topics are subclasses of Petri net and various extensions of Petri net.

A vast formalism like Petri net theory demands attention to many aspects. Survey report on such a topic, henceforth, has to be either extensive or exhaustive. This report has been made with more emphasis on the depth of the matter rather than attempting to cover all aspects of the Petri net theory in a single run. This is the reason why the report has more vertical growth compared to lateral growth. However, it can be mentioned that, this is only a preliminary survey report made within the time limit of two months. The author hopes that the next tier report will be able to address the subclasses, various extensions, simulation (tools and limitations) and applications of Petri net. Once these basic concepts are addressed, then only one will be able to address interesting questions like applicability of optimal control theory to Petri nets, asynchronous decentralized control system modeling and analysis, hybrid system modeling, simulation, analysis and performance evaluation etc. In this sense this report, though looks very narrow-based at first glance, is rather open-ended.

A P P E N D I X

A

Algebra of Partitioned Matrices

A1. Definition of Partition Matrix or Block Matrix

A *Partition Matrix* or *Block Matrix* is a matrix whose elements are themselves matrices, called *blocks* (not necessarily square blocks), such that in each row all the blocks have same number of rows and in each column all the blocks have same number of columns. Thus block matrix is a matrix of matrices. One can suitably partition a large dimensional matrix into smaller blocks to obtain the partition matrix or block matrix.

Example

Let the matrix $X = \begin{bmatrix} 0 & 2 & 3 & 3 & 3 \\ 2 & 0 & 3 & 3 & 3 \\ 4 & 4 & 5 & 0 & 5 \\ 4 & 4 & 0 & 5 & 0 \\ 4 & 4 & 5 & 0 & 5 \end{bmatrix}$. It is a (5×5) matrix. Let X be partitioned as :

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}.$$

where

$$A = \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \end{bmatrix}$$

$$C = \begin{bmatrix} 4 & 4 \\ 4 & 4 \\ 4 & 4 \end{bmatrix}$$

$$D = \begin{bmatrix} 5 & 0 & 5 \\ 0 & 5 & 0 \\ 5 & 0 & 5 \end{bmatrix}$$

Note that the blocks A and B have same no. of rows. Similarly blocks C and D have same no. of rows.
Again blocks A and C have same no. of columns. Similarly blocks B and D have same no. of columns.

A2. Partitioning a large dimensional matrix makes computation easier : An Illustrative Example

Often it is needed to compute the product of two large dimensional matrices. If we directly multiply the two matrices then it is very time consuming and sometimes impossible to calculate the product, since the memory space required to store such huge matrices in computer, is very large. This problem can be circumvented by partitioning the two matrices into smaller dimensional blocks. Then the product of two block matrices can be stored as the multiplication of smaller dimensional blocks.

Let

$$A = \begin{bmatrix} a_{11} & a_{12} & \Lambda & a_{1\ np} \\ a_{21} & a_{22} & \Lambda & a_{2\ np} \\ \mathrm{M} & \mathrm{M} & \mathrm{O} & \mathrm{M} \\ a_{mq\ 1} & a_{mq\ 2} & \Lambda & a_{mq\ np} \end{bmatrix}$$

be a $(mq \times np)$ matrix and

$$B = \begin{bmatrix} b_{11} & b_{12} & \Lambda & b_{1\ sr} \\ b_{21} & b_{22} & \Lambda & b_{2\ sr} \\ \mathrm{M} & \mathrm{M} & \mathrm{O} & \mathrm{M} \\ b_{np\ 1} & b_{np\ 2} & \Lambda & b_{np\ sr} \end{bmatrix}$$

be a $(np \times sr)$ matrix. We can partition the two matrices by

$$A = \begin{bmatrix} A_{11} & A_{12} & \Lambda & A_{1\ p} \\ A_{21} & A_{22} & \Lambda & A_{2\ p} \\ \mathrm{M} & \mathrm{M} & \mathrm{O} & \mathrm{M} \\ A_{q\,1} & A_{q\,2} & \Lambda & A_{qp} \end{bmatrix}$$

and

$$B = \begin{bmatrix} B_{11} & B_{12} & \Lambda & B_{1\,r} \\ B_{21} & B_{22} & \Lambda & B_{2\,r} \\ \mathrm{M} & \mathrm{M} & \mathrm{O} & \mathrm{M} \\ B_{p\,1} & B_{p\,2} & \Lambda & B_{pr} \end{bmatrix},$$

where A_{ij} are $(m \times n)$ matrices and B_{jk} are $(n \times s)$ matrices**,**

$$i = 1,2,\mathrm{K}\ ,q;\ j = 1,2,\mathrm{K}\ ,p,\ k = 1,2,\mathrm{K}\ ,r.$$

Then,

$$AB = \begin{bmatrix} A_{11} & A_{12} & \Lambda & A_{1p} \\ A_{21} & A_{22} & \Lambda & A_{2p} \\ \mathrm{M} & \mathrm{M} & \mathrm{O} & \mathrm{M} \\ A_{q1} & A_{q2} & \Lambda & A_{qp} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} & \Lambda & B_{1r} \\ B_{21} & B_{22} & \Lambda & B_{2r} \\ \mathrm{M} & \mathrm{M} & \mathrm{O} & \mathrm{M} \\ B_{p1} & B_{p2} & \Lambda & B_{pr} \end{bmatrix}$$

$$= \begin{bmatrix} C_{11} & C_{12} & \Lambda & C_{1r} \\ C_{21} & C_{22} & \Lambda & C_{2r} \\ \mathrm{M} & \mathrm{M} & \mathrm{O} & \mathrm{M} \\ C_{q1} & C_{q2} & \Lambda & C_{qr} \end{bmatrix} = C_{mq \times sr}$$

where

$$C_{ij} = \sum_{k=1}^{p} A_{ik} B_{kj} = A_{i1} B_{1j} + A_{i2} B_{2j} + \Lambda + A_{ip} B_{pj},\ i = 1,2,\mathrm{K},q;\ j = 1,2,\mathrm{K},r,$$

are $(m \times s)$ matrices**.**

So instead of storing $(mq \times sr)$ elements, now it is enough to store $(m \times s)$ elements. Thus simple partitioning can significantly reduce memory space requirement.

A3. Definition of Block Diagonal Matrix

A *Block Diagonal Matrix* is a partition or block matrix whose off-diagonal blocks are zero matrices and diagonal blocks are square matrices.

If D is a block matrix of the form $D = \begin{bmatrix} A_{11} & 0 & 0 \\ 0 & A_{22} & 0 \\ 0 & 0 & A_{33} \end{bmatrix}$ where A_{ij} are matrices, then we write $D = diag(A_{11}, A_{22}, A_{33})$ and call D as *Block Diagonal Matrix*.

Furthermore,

$$| D | = | A_{11} | . | A_{22} | . | A_{33} | \qquad \text{(a)}$$

and if $D \neq 0$, then

$$D^{-1} = diag(A^{-1}{}_{11}, A^{-1}{}_{22}, A^{-1}{}_{33}) \qquad \text{(b)}$$

A4. Definition of Jordan Block

A square matrix is called *Jordan Block* if

i. each element along the diagonal consists of a single number λ,
ii. each element along the superdiagonal consists of 1,
iii. all other elements of the matrix are zero.

Thus a Jordan block of order q is a $(q \times q)$ matrix denoted as J_q given by

$$J_q = \begin{bmatrix} \lambda & 1 & 0 & \ldots & 0 & 0 \\ 0 & \lambda & 1 & \ldots & 0 & 0 \\ 0 & 0 & \lambda & \ldots & 0 & 0 \\ 0 & 0 & 0 & \ldots & 1 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & 1 \\ 0 & 0 & 0 & \ldots & 0 & \lambda \end{bmatrix}$$

It can be noted that the degenerate case of (1×1) matrix is considered as a Jordan block, even though it lacks a superdiagonal to be filled with 1s [15].
Instead of taking 1s along the superdiagonal, sometimes they are taken along the subdiagonal [16].

A5. Definition of Jordan Canonical Form

Jordan Canonical Form is a special kind of block diagonal matrix where each diagonal block is a Jordan block with possibly differing constants λ_i.
Thus a Jordan Canonical Form J may be given as

$$J = \begin{bmatrix} J_{q1} & 0 & 0 & 0 & 0 & 0 \\ 0 & J_{q2} & 0 & 0 & 0 & 0 \\ 0 & 0 & J_{q3} & 0 & 0 & 0 \\ 0 & 0 & 0 & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & 0 \\ 0 & 0 & 0 & \ldots & 0 & J_{qn} \end{bmatrix}$$

Example

$$J = \begin{bmatrix} 7.9 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 7.9 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 7.9 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7.9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1-2i & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1-2i & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1-2i \end{bmatrix} = \begin{bmatrix} J_4 & 0 \\ 0 & J_3 \end{bmatrix}$$

A6. Usefulness of Jordan Canonical Form

Most of the linear algebra problems involving linear system of equations are easily manageable if the coefficient matrix is diagonalizable. However, for a non-diagonalizable matrix A , it is difficult to compute quantities like A^k or e^A. Hence, finding the general solution of the form $\vec{x}(t) = \vec{C}e^{At}$ of a system of linear differential equations $\dot{\vec{x}}(t) = A\vec{x}(t)$ may not be easy. Jordan Canonical Form provides a way to handle such non-diagonalizable matrix A and hence to compute A^k or e^A.

A7. Definition of Upper Block Triangular and Lower Block Triangular Matrix

If $D = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A_{22} & A_{23} \\ 0 & 0 & A_{33} \end{bmatrix}$ where A_{ij} are matrices, then D is *Upper Block Triangular* matrix and (a) still holds. (c)

Lower Block Triangular matrices have the form of the transpose of (c).

A8. Schur Complement and Matrix Inversion Lemma

If A is a partitioned or block matrix of the form

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \tag{d}$$

then we define *Schur Complement* of A_{22} as

$$D_{22} = A_{22} - A_{21}A^{-1}{}_{11}A_{12} \tag{e}$$

and the *Schur Complement* of A_{11} as

$$D_{11} = A_{11} - A_{12}A^{-1}{}_{22}A_{21} \tag{f}$$

The inverse of A can be written

$$A^{-1} = \begin{bmatrix} A^{-1}{}_{11} + A^{-1}{}_{11}A_{12}D^{-1}{}_{22}A_{21}A^{-1}{}_{11} & -A^{-1}{}_{11}A_{12}D^{-1}{}_{22} \\ -D^{-1}{}_{22}A_{21}A^{-1}{}_{11} & D^{-1}{}_{22} \end{bmatrix} \tag{g}$$

$$A^{-1} = \begin{bmatrix} D^{-1}{}_{11} & -D^{-1}{}_{11}A_{12}A^{-1}{}_{22} \\ -A^{-1}{}_{22}A_{21}D^{-1}{}_{11} & A^{-1}{}_{22} + A^{-1}{}_{22}A_{21}D^{-1}{}_{11}A_{12}A^{-1}{}_{22} \end{bmatrix} \tag{h}$$

or $$A^{-1} = \begin{bmatrix} D^{-1}{}_{11} & -A^{-1}{}_{11}A_{12}D^{-1}{}_{22} \\ -A^{-1}{}_{22}A_{21}D^{-1}{}_{11} & D^{-1}{}_{22} \end{bmatrix} \tag{i}$$

depending, of course, on whether $|A_{11}| \neq 0$, $|A_{22}| \neq 0$, or both. These can be verified by checking that $AA^{-1} = A^{-1}A = I$. By comparing these various forms, we obtain the well known *Matrix Inversion Lemma*

$$(A^{-1}{}_{11} + A_{12}A_{22}A_{21})^{-1} = A_{11} - A_{11}A_{12}(A_{21}A_{11}A_{12} + A^{-1}{}_{22})A_{21}A_{11} \tag{j}$$

The Schur Complement arises naturally in the solution of linear simultaneous equations, for if

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} 0 \\ Z \end{bmatrix} \tag{k}$$

then from the first equation

$$X = -A^{-1}{}_{11}A_{12}Y \tag{l}$$

and using (l) in the second equation yields

$$(A_{22} - A_{21}A^{-1}{}_{11}A_{12})Y = Z \tag{m}$$

If A is given by (d), then

$$|A| = |A_{11}| \cdot |A_{22} - A_{21}A^{-1}{}_{11}A_{12}| = |A_{12}| \cdot |A_{11} - A_{12}A^{-1}{}_{22}A_{21}| \tag{n}$$

Hence the determinant of A is the product of the determinant of A_{11} (or A_{22}) and the determinant of Schur Complement of A_{22} (or A_{11}).

A P P E N D I X

B

Controllability

Linear System Theory for Petri Nets

B1. Two Important Points

1. In the literature, a state is said to be *controllable* if there exists a sequence of inputs which can transfer that state to zero state or nominal state. Again, a state is said to be *reachable* if there exists a sequence of inputs which can transfer zero state or nominal state to that state. Many books on system theory do not make any distinction between these two definitions and controllability is defined in both the ways dropping the term reachability. However, in PN context, the term reachability has got a wide significance. Hence it is imperative in PN context, to closely follow the classical definition of controllability to avoid any confusion.

2. The terms '*completely controllable*' and '*controllable*' are often taken synonymously in the literature. In PN context, as will be proved soon, complete controllability condition is hardly satisfied but weak controllability is not so rare.

B2. Condition of Controllability for a Linear System

To avoid mathematical complexity, we will consider linear time invariant (LTI) system. This will serve as a representative case for deriving condition of controllability in linear system theoretic framework. Once condition of controllability form LTI system is established then we will make an attempt to make a parallel study of controllability in PN context and in the process, we will end up with a corresponding condition of controllability (i.e. complete controllability) in PN context.

For LTI systems, the n-dimensional linear state equation is given by

$$\dot{\vec{x}}(t) = A\vec{x}(t) + B\vec{u}(t) \qquad \text{(a)}$$

where
$\vec{x}(t)$ is $(n \times 1)$ state vector,
$\vec{u}(t)$ is $(p \times 1)$ input vector,
A is $(n \times n)$, B is $(n \times p)$ — Coefficient matrices.

Theorem: The n-dimensional linear time-invariant state equation is controllable iff any of the following equivalent conditions are satisfied.

i. All rows of $e^{-At}B$ (and consequently of $e^{At}B$) are linearly independent on $[0, \infty)$ over C, the field of complex numbers.

ii. The controllability grammian

$$W = \int_0^t e^{A\tau} BB^* e^{A^*\tau} d\tau \quad \text{is nonsingular for any } t > 0 .$$

iii. The $(n \times np)$ controllability matrix

$$U \quad [A : AB : A^2B : \ldots : A^{n-1}B] \text{ has rank } n.$$

Proof: We will prove here that the controllability conditions, as listed above, are equivalent. The first attempt is to prove that conditions (i) and (ii) are equivalent. Then we will prove that (iii) is equivalent to condition (i) (and hence also to condition (ii)).

Proving Equivalence of Condition (i) and (ii)

Theorem: Let $\overset{\omega}{f_i}$ for $i = 1,2,\ldots,n$ be $(1 \times p)$ complex-valued continuous functions defined on $[t_1, t_2]$. Let F be the $(n \times p)$ matrix with $\overset{\omega}{f_i}$ as its i^{th} row. Define

$$W(t_1, t_2) = \int_{t_1}^{t_2} F(t)F^*(t)dt$$

Then $\overset{\omega}{f_1}$, $\overset{\omega}{f_2}$, … , $\overset{\omega}{f_n}$ are linearly independent on $[t_1, t_2]$ iff the $(n \times n)$ constant matrix $W(t_1, t_2)$ is non-singular[†].

Proof:

Proof of necessity of the theorem

Let's assume that $\overset{\omega}{f_i}$s are linearly independent on $[t_1, t_2]$ but $W(t_1, t_2)$ is singular.

Then $\exists$ a non-zero $(1 \times n)$ row vector $\overset{\omega}{\alpha}$ such that $\overset{\omega}{\alpha} W(t_1, t_2) = \overset{\omega}{0}$. This implies

$$\alpha W(t_1, t_2)\alpha^* = 0 \Rightarrow \alpha W(t_1, t_2)\alpha^* = \int_{t_1}^{t_2} (\alpha F(t))(\alpha F(t))^* dt = 0 \qquad \text{(b)}$$

Since the integrand $(\alpha F(t))(\alpha F(t))^*$ is a continuous function and is non-negative $\forall t \in [t_1, t_2]$, equation (b) implies

$$\alpha F(t) = 0 \qquad \forall t \in [t_1, t_2].$$

This contradicts the linear independence assumption of the set $\overset{\omega}{f_i}$, $i = 1,2,\ldots,n$. Hence if the $\overset{\omega}{f_i}$s are linearly independent on $[t_1, t_2]$, then $\det . W(t_1, t_2) \neq 0$.

Proof of sufficiency of the theorem

Suppose $W(t_1, t_2)$ is non-singular. But $\overset{\omega}{f_i}$s are linearly dependent on $[t_1, t_2]$. Then by definition, $\exists$ a non-zero constant $(1 \times n)$ row vector $\overset{\omega}{\alpha}$ such that $\overset{\omega}{\alpha} F(t) = \overset{\omega}{0}$ $\forall t \in [t_1, t_2]$. Consequently we have

$$\alpha W(t_1, t_2) = \int_{t_1}^{t_2} \alpha F(t)F^*(t)dt = 0$$

[†] In fact, the matrix $W(t_1, t_2)$, called the *Grammian matrix* is positive definite. The determinant of $W(t_1, t_2)$ is called the *Gram determinant* of $\overset{\omega}{f_i}$s.

which contradicts the assumption that $W(t_1,t_2)$ is non-singular. Hence, if $W(t_1,t_2)$ is non-singular, then the $\vec{f}_i$s are linearly independent on $[t_1,t_2]$. *(Proved)*

Now we note that $e^{-At}B$ (and consequently $e^{At}B$) are real valued continuous functions, a subset of complex valued continuous functions. Hence the above theorem, when applied to $e^{-At}B$ (and consequently to $e^{At}B$) directly proves the equivalence of condition (i) and condition (ii).

Proving Equivalence of Condition (i) and (iii)

Theorem: Assume that for each i, $\vec{f}_i$ is analytic on $[t_1,t_2]$. Let F be the $(n\times p)$ matrix with $\vec{f}_i$ as its i^{th} row and let $F^{(k)}$ be the k^{th} derivative of F. Let t_0 be any fixed point in $[t_1,t_2]$. Then the $\vec{f}_i$s are linearly independent on $[t_1,t_2]$ iff

$$\rho[F(t_0):F^{(1)}(t_0):F^{(2)}(t_0):\ldots:F^{(n-1)}(t_0):\ldots]=n.$$

Proof:

Similar to the proof of previous theorem using method of contradiction.
Since the entries of $e^{-At}B$ are analytic functions, the above stated theorem implies that the rows of $e^{-At}B$ are linearly independent on $[0,\infty)$ iff

$$\rho[e^{-At}B:-e^{-At}AB:e^{-At}A^2B:\ldots:(-1)^{(n-1)}e^{-At}A^{(n-1)}B:\ldots]=n$$

$\forall t\in[0,\infty)$.
Now let $t=0$; then this equation reduces to

$$\rho[B:-AB:A^2B:\ldots:(-1)^{(n-1)}A^{(n-1)}B:(-1)^nA^nB:\ldots]=n$$

From Cayley-Hamilton theorem, we know that A^m with $m\geq n$ can be written as a linear combination of $I,A,\ldots,A^{n-1}$. Hence the columns of A^mB with $m\geq n$ are linearly dependent on the columns of $B,AB,\ldots,A^{(n-1)}B$. Consequently

$$\rho[B:-AB:A^2B:\ldots:(-1)^{(n-1)}A^{(n-1)}B:\ldots]=\rho[B:-AB:A^2B:\ldots:(-1)^{(n-1)}A^{(n-1)}B]$$

Since changing the sign does not change linear independence, we conclude that the rows of $e^{-At}B$ are linearly independent iff

$$\rho[B:AB:A^2B:\ldots:A^{(n-1)}B]=n. \qquad \text{(c)}$$

The matrix $[B:AB:A^2B:\ldots:A^{(n-1)}B]$ is called the Controllability matrix of the LTI system. This completes the proof of the equivalence of statements (i) and (iii). *(Proved)*

B3. Condition of Controllability for a PN

The state equation of a PN, when compared with the state equation of LTI system, reveals the fact that the identity matrix in PN state equation is analogous to the state vector coefficient matrix A in LTI system and the incidence matrix in PN state equation is analogous to the control vector coefficient matrix B in LTI system. Thus in equation (c), replacing A by identity matrix and B by incidence matrix, we obtain that the controllability matrix for PN becomes $[A : A : A : \ldots : A\ ]$ and the condition for controllability becomes

$$\rho[A : A : A : \ldots : A\ ] = n$$
$$\Rightarrow \rho[A] = n$$

Hence the controllability requirement in PN context is that the rank of the incidence matrix must be equal to the no. of places in the PN.

References

[1] Leslie Lamport, “Time, clocks and the ordering of events in a distributed system”, *Communications of the ACM*, Vol. 21, no. 7, July, 1978, pp. 558 – 565.

[2] Tadao Murata, “Petri nets: properties, analysis and applications”, *Proc. IEEE*, Vol. 77, no. 4, April, 1989, pp. 541 – 580.

[3] Alessandro Giua and Frank DiCesare, “Petri net structural analysis for supervisory control”, *IEEE Trans. Robotics and Automation*, Vol. 10, no. 2, April, 1994, pp. 185 – 195.

[4] James F. Watson, III and Alan A. Desrochers, “State-space size estimation of Petri nets: a bottom-up perspective”, *IEEE Trans. Robotics and Automation*, Vol. 10, no. 4, August, 1994, pp. 555 - 560.

[5] V. K. Agrawal, “Formal tools for specification driven protocol design of distributed computing systems”, PhD Thesis, Dept. of Computer Science and Automation, Indian Institute of Science, Bangalore, March, 1986.

[6] Tadao Murata, “State equation, controllability and maximal matchings of Petri nets”, *IEEE Trans. Automatic Control*, Vol. AC-22, NO. 3, June, 1977, pp. 412 – 416.

[7] J. L. Peterson, *Petri net theory and modeling of systems*, Englewood Cliffs, NJ: Printice-Hall, 1981.

[8] L. Ferrarini, M. Narduzzi and M. Tassan-Solet, *IEEE Trans. Robotics and Automation*, Vol. 10, no. 2, April, 1994, pp. 169 – 184.

[9] Alan A. Desrochers and Robert Al-Jaar, *Applications of Petri nets in manufacturing systems: modeling, control and performance analysis*, IEEE Press, Piscataway, NJ: 1994.

[10] J. Martinez and M. Silva, “A simple and fast algorithm to obtain all invariants of a generalized Petri net”, 2nd European Workshop on the Application and Theory of Petri nets, Bad Honnef (FRG), September, 1981, pp. 411 – 422.

[11] M. Silva and T. Murata, “B-fairness and structural B-fairness in Petri net models of concurrent systems”, Technical Report no. UIC-EECS-86-10, University of Illinois at Chicago, June, 1986.

[12] David E. Johnson and Johnny R. Johnson, *Graph theory with engineering applications*, The Ronald Press Company, New York, 1972.

[13] G. Memmi and G. Roucairol, “Linear algebra in net theory”, *Lecture notes in computer science (LNCS)*, Vol. 84, no. 15, 1980, pp. 213 – 223.

[14] Javier Campos, Giovanni Chiola and Manuel Silva, “Ergodicity and throughput bounds of Petri nets with unique consistent firing count vector”, *IEEE Trans. Software Engineering*, Vol. 17, no. 2, February, 1991, pp. 117 – 125.

[15] G. Strang, *Linear Algebra and Its Applications*, 3rd Edition, Philadelphia, PA : Saunders, 1988.

[16] V. N. Faddeeva, *Computational Methods of Linear Algebra*, New York : Dover, 1958.

[17] C. T. Chen, Linear Systems Theory and Design, 3rd Edition, Oxford University Press, 1998.