SOFTWARE DEVELOPMENT FOR ANALYSIS OF STOCHASTIC PETRI NETS USING TRANSFER FUNCTIONS

A Thesis presented

by

Aman U. Jamwal

to

The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements
for the degree of

Master of Science in Electrical Engineering

New Jersey Institute of Technology
Newark, New Jersey

December 1991

APPROVAL SHEET

Thesis Title: **Software Development for Analysis of Stochastic Petri Nets Using Transfer Functions**

Name of Candidate: Aman Ullah Jamwal
Master of Science in Electrical Engineering

Thesis and Abstract Approved by:

[illegible]

Dr. MengChu Zhou — Date
Assistant Professor
Electrical and Computer Engineering Department

Members of the
Thesis Committee:

Dr. Anthony Robbi — Date
Associate Professor
Electrical and Computer Engineering Department

[illegible]

Dr. Daniel Chao — Date
Assistant Professor
Computer and Information Science Department

VITA

Name	Aman U Jamwal
Degree	M. S. in Electrical Engineering

Secondary Education	St. Jude's High School, Karachi, Pakistan	
Collegiate Institutions	Degree	Date Awarded attended
New Jersey Institute of Technology, USA	M.S.E.E	December 1991
N. E. D. University of Engineering & Technology, Karachi, Pakistan.	B.E.E.E	February, 1989

Dedicated

to

My parents, family
and especially, my wife

CONTENT

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express here my deepest gratitude for my parents, wife and family for their continuous support and encouragement throughout my studies. Their affection and prayers have played an important role for completion of my Master's studies at NJIT.

I acknowledge the consistent guidance and advisement of Dr. MengChu Zhou and Dr. Robbi of NJIT, which has led me to achieve my thesis objectives. I want to mention here that the weekly Petri Net group meetings arranged by Dr. MengChu Zhou, Dr. Robbi and Dr. Daniel Chao have been of a great technical advantage for my thesis work. I would like to thank the members of the thesis committee for reviewing and commenting my thesis report. I am also grateful for the partial support of the Center for Manufacturing Systems of NJIT.

ABSTRACT

This thesis research is an implementation of a closed-form analytical technique for study, evaluation and analysis of Stochastic Petri Nets (SPN). The technique is based on a theorem that an isomorphism exists between an SPN and a Markov Chain. The procedure comprises five main steps: reachability graph generation of the underlying Petri net, transformation of the reachability graph to a state machine Petri net, calculation of transfer functions, computation of equivalent transfer functions via Mason's rule, and computation of performance parameters of the SPN model from the equivalent transfer functions and their derivatives. The software is developed in UNIX using C and applied to various SPN models. Future research includes implementation of Mason's rule for complex cases and symbolic derivation of equivalent transfer functions.

Chapter 1 INTRODUCTION

1.1 MOTIVATION AND OBJECTIVES

Petri nets (PN) are a graphical tool for study, representation and modeling of systems which can be called event-driven, asynchronous, distributed, parallel or discrete event systems. These systems are diversified, for example, computer network systems, computer hardware, computer software, physical systems, social systems, chemical processes, traffic and transportation systems, queueing systems, communication systems, political systems and biological systems. Petri net modeling provides a representation of the system which is graphical and mathematically based.

a) Motivations

Petri net theory, from its beginning in 1962 by Dr. Petri's Doctoral dissertation,has gained popularity in a majority of European countries and in the United States, with major reasons being its simplicity and capabilities. Use of Petri nets is constantly being exploited in many areas, such as, computer science, engineering, transportation, physics and economics. With the acceptance of Petri nets to be a major modeling tool for event-driven systems, it is becoming expected of every computer science or engineering student, engineering professionals and system analyst, to know some basic Petri net theory.

Recently, the inclusion of time as a specification in Petri nets has been done. This has lead to evolution of Time Petri nets (TPN) and Stochastic Petri nets (SPN). The major strength of the SPN is in modeling of systems which involve events occurring in random time.

The inherent features of graphical nature, system state presentation, simple execution and implementation, make Petri nets very suitable for system modeling. Modeling is complemented with analysis. The analysis of a PN can lead to important insights of the system, which may be helpful for study, verification, evaluation and implementation of a system. The performance analysis of systems is an analysis which evaluates the dynamics of the system. A closed-form performance analysis is an approach which is based upon the mathematical model of the system. This thesis is an attempt to perform analysis of systems modeled by Stochastic Petri nets, by using a methodology which is based on the concepts of Markov theory, control systems and symbolic computation methods. It is called the Moment Generating Function (MGF) or the Transfer functions approach. This technique has been recently put forward by Guo, DiCesare and Zhou [Guo 1991; Guo 1992]. The improvement, implementation in software and application, of this technique, is the scope of this research.

b) *Thesis Objectives*

The prime objectives of the thesis are highlighted below:

1) To devise the data structures for Petri net representation in a matrix form. This includes incidence matrix, input and output arcs mapping functions, markings and transition firing distribution parameters.

2) To include stochastic firing delay distributions. At first the exponential and immediate transitions will be used, but the structures will be developed to be adaptable to the future addition of non-expnential transitions.

3) To implement the net execution, produce firing sequence and generate reachability graph of the underlying Petri net and transform it to a state machine Petri net.

4) To compute the moment generating function, branch probability and transfer functions of the transformed state machine Petri net.

5) To allow inhibitor arcs in the structure of Petri nets dealt by the program.

6) To devise data structure for application of Mason's rule to complex and large Petri nets, so that the equivalent transfer functions can be computed by the software.

7) To develop a C program for computation of derivatives of transfer functions.

8) To compute various performance parameters of the net depending upon the system it modeled, for example throughput, fault rate, production rate, steady-state probabilities, etc.

9) To apply the developed software to various SPN models.

1.2 PETRI NET STRUCTURE

1.2.1 Petri Net Graph

More theoretical work on Petri nets is based on the formal definition of PN. However, a graphical representation of PN is much more useful for illustrating the concepts of Petri net theory and its applications. A Petri net graphically consists of following components.

1. Circles (called places) representing conditions or availability of system resources e.g.; machines, data, memory space, etc.
2. Bars (called transitions) representing the initiation or termination of an

event. They may also be used to represent events.

3. Black dots (called tokens) residing in places, representing the availability of the resource represented by the resource place or execution of an operation when residing in an operation place. An infinite number of tokens are permissible in an ordinary Petri net. The presence of tokens in the places controls the execution of the Petri net

The relationship between places and transitions is specified by two functions describing directed arcs from the point of view of transitions: I, the input function, and O, the output function. The arcs can be multiple. A simple Petri net with structural entities is shown in Figure 1.1.

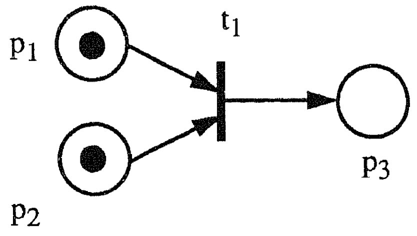


Figure 1.1 A simple Petri net

Formally, an ordinary Petri net (PN) is a five-tuple,

$$Z = (P, T, I, O, m)$$

where

$P = \{p_1, p_2, ..., p_n\}$, $n>0$, and is a set of *n places*;

$T = \{t_1, t_2, ..., t_s\}$, $s>0$, and is a set of *s transitions*;

I is an n $\times$ s matrix indicating the places which are the input of each transition. It is a mapping : $P \times T \rightarrow \{0, 1, 2, ...\}$ corresponding to the set of directed arcs *from places to transitions.*

O is an n $\times$ s matrix indicating the places which are the output of each transition. It is a mapping : $P \times T \rightarrow \{0, 1, 2, ...\}$ corresponding to the set of directed arcs *from transitions to places.*

$m : P \rightarrow N$ and is a marking whose ith component represents the number of tokens in the ith place. An initial marking is denoted by m_0

1.2.2 Marking of a Petri net

Marking of a Petri net is an assignment of tokens to the places of a Petri net. Tokens reside in the places of a Petri net. The number and position of tokens may change during the execution of a Petri net. Any number of tokens can be assigned to the places,

$m = [m(p_1), m(p_2), ..., m(p_n)]^T$ is an n-dimensional, integer valued vector indicating the number of tokens in each place and is known as the marking of the Petri net. For example the Petri net of Figure 1.1 has a marking $m = [\ 1, 1, 0]^T$

1.2.3 Execution Rules

A PN executes by firing transitions. A transition t is enabled if each of its input places has at least as many tokens in it as the number of input arcs from that input place to the transition t. The tokens in the input places which enable a transition are its *enabling tokens*. Firing of a transition means that enabling tokens are removed from the input places of the transition and new tokens are created and distributed to all output places of the transition. The number of tokens put in the each output place of the fired transition equals the number of output arcs between the transition and that output place.

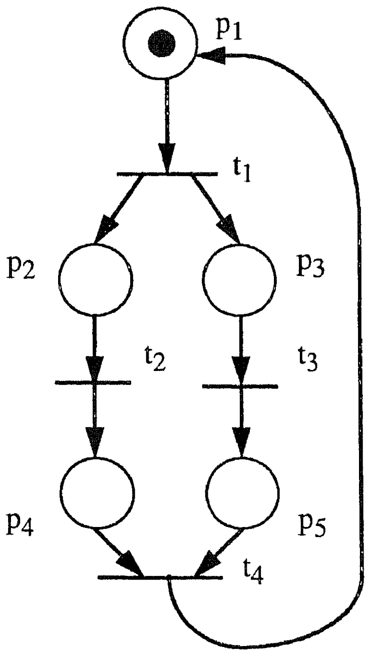
Definition: A transition $t \in T$ in a marked Petri net $Z = (P,T, I, O, m)$ is enabled in a marking m, if,

$$m(p) \geq I(p, t) \ , \quad \forall p \in P$$

A transition fires by removing all of its enabling tokens from its input places and then depositing into each of its output places one token for each arc from the transition to the output place. The resulting new marking of the net is represented as

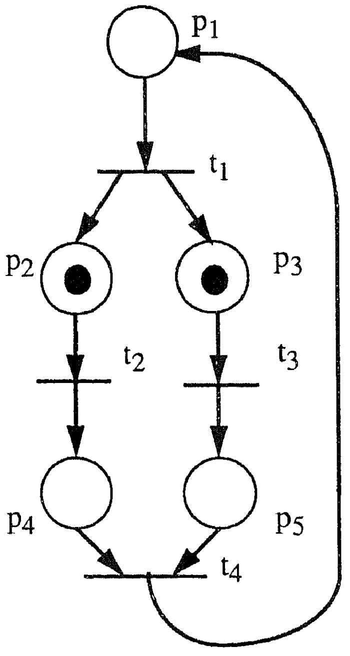
$$m_1(p) = m_0(p) - I(p,t) + O(p,t) \; \forall \; p \in P$$

The firing of transitions and finding of reachability of the net, makes Petri nets capable of modeling dynamic behavior of the modeled system. Execution of a PN is shown in Figure 1.2.(a) through (d).



a) t_1 enabled

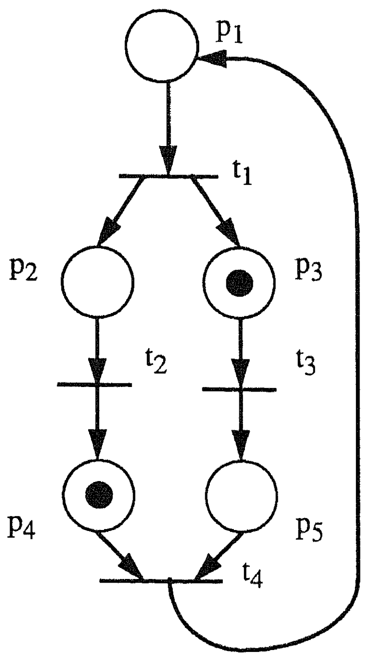
$m_0 = [1\ 0\ 0\ 0\ 0]$



b) t_1 fired, t_2 and t_3 enabled

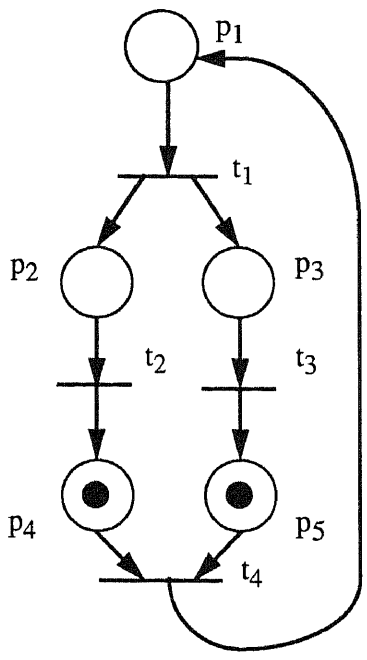
$m_1=[0\ 1\ 1\ 0\ 0]$

Figure 1.2(a) (b) PN Execution



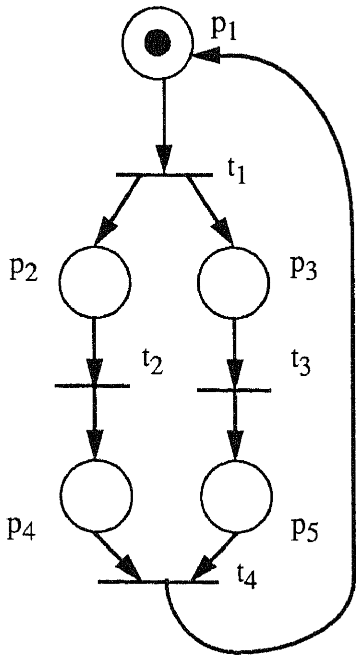
c) t_2 fired randomly

m_2=[0 0 11 0]



d) t_3 fired, t_4 now enabled

m_3=[0 0 0 1 1]



e) t_4 fired.

m4=m0

Figure 1.2 (c), (d) , (e) PN Execution

1.2.4 Firing Sequence

The firing of a transition results in the change of state of PN, i.e., the transformation of the net from an old marking to a new marking. If the new marking has enabled transitions, it may again fire to generate another marking and so on. Thus execution of a Petri net leads to a sequence of transition firings, denoted by,

$$\sigma_s = t_j, t_k, \ldots, t_n$$

which means that transition t_j fires first, then t_k and so on until transition t_n.

A firing sequence can be associated with a firing vector Ns which is an s-dimensional, non-negative integer vector whose jth component corresponds to the number of occurrences of the transition t_j in the sequence.

1.2.5 Incidence Matrix

The formal definition of an ordinary Petri net uses two functions to specify the directed arcs between places and transitions. These functions are the input function I(p,t) and the output function O(p,t). Based on these mapping functions, the matrix description of Petri net is made. I is an n x s matrix indicating the places which are the input of each transition. O is an n x s matrix indicating the number of arcs for all output places of each transition. Combining the information of these I and O matrices, we define a matrix C, known as Incidence matrix. It describes the number of directed arcs between each node. Consider the PN shown in Figure 1.3,

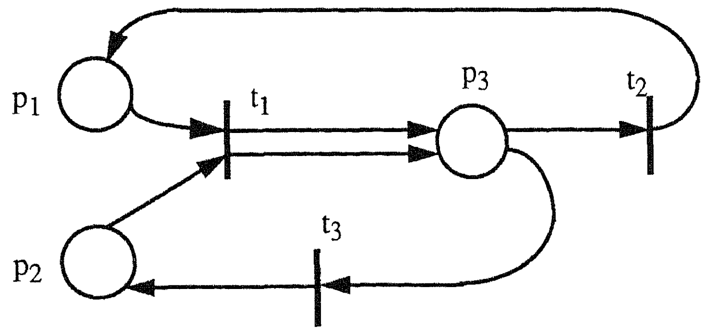


Figure 1.3 Example for matrix representation

The incidence matrix as a combination of input and output arcs matrices are:

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \qquad O = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 2 & 0 & 0 \end{bmatrix} \qquad C = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & -1 \\ -1 & 0 & 0 \end{bmatrix}$$

1.3 MODELING CAPABILITIES OF PN

The application of Petri nets is through modeling. Modeling is an indirect representation, mostly in mathematical terms, of major features of the system. It provides important insights of the modeled system without inconvenience, danger or cost of the actual system. Petri net inherently possess powerful modeling capabilities.

1.3.1 Concept of System State in Petri Net

Every system is composed of various interacting components. A system can be described in terms of its "state" and "state transformations". The interaction of system components produces changes in the state of the system and controls the behavior of the system. The concept of state is inherent in Petri nets. The state of a PN is its marking, which describes the assignment of tokens at places of the net. The state space of a Petri net is the set of its markings, given as N^n. The change in system state, state transformation, is represented by firing of transitions.

1.3.2 Concept of Events and Conditions

The simple Petri net view of a system concentrates on two primitive concepts: events and conditions. Events are actions or activities which take place in the system. Events are controlled by the state of the system. The state of a system can be described as a set of conditions. A condition is a predicate or logical description of the state of the system. A condition may hold or not hold.

1.3.3 Concurrency and Parallelism

In a system, the components may exhibit concurrency or parallelism when activities occur simultaneously with other activities. Consider the example of Petri net with parallel activities, shown in Figure 1.4. which is a Petri net model of computer program containing constructs of PARBEGIN-PAREND, which indicate parallel operation. The transitions t2, t3, t4 are concurrent.

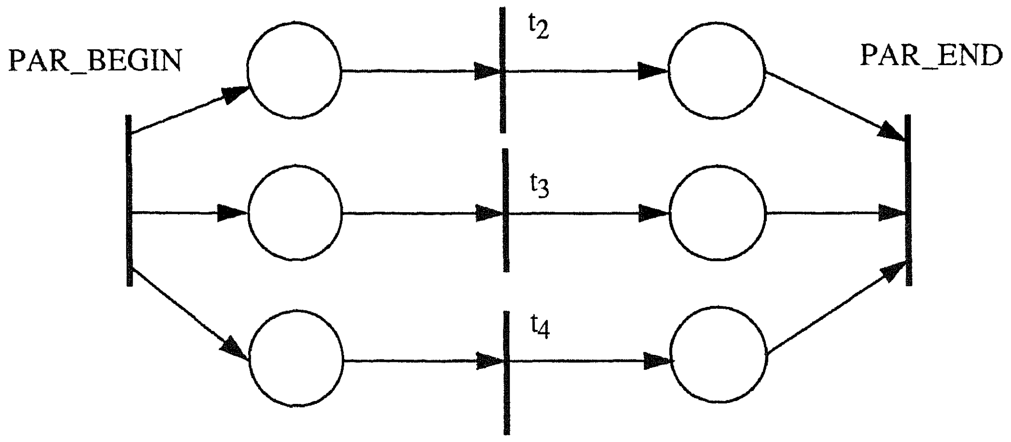


Figure 1.4 PN representing concurrency in a program

1.4 BEHAVIORAL PROPERTIES OF PN

1.4.1 Safeness

A place in a Petri net is safe if the number of tokens in that place never exceed one. A petri net is safe if all of its places are safe. Definition. A place pi P of a PN, given as Z = (P, T, I, O) with initial marking m is safe if for all m' R(Z,m) , if,

$$m(p_i) \leq 1$$

A Petri net is safe if all of each of its place is safe.

1.4.2 Boundedness

Boundedness is a generalization of safeness of a net with the situation that the places can hold i a particular number of tokens. A place is k-bounded, if the number of tokens in that place cannot exceed an integer k. Definition: A place pi P of a Petri net C = (P,T,I,O) with an initial marking m is k-safe or k-bounded if for all reachable markings

$$m(p_i) \leq k$$

Since there are only a finite number of places in Petri net, we can find the k as the maximum of the bounds of each place and define a Petri net to be k-bounded or k-safe if every place in it is k-bounded.

1.4.3 Liveness

In many systems, especially in computer systems, the problem of resource allocations arise. A blockage of resource access or allocation lead to a situation known as deadlock. This situation is modeled by the Petri nets. A deadlock in a Petri net is a transition(or set of transitions) which cannot fire. Conversely, we can define a transition to be live if the transition is not deadlocked and can potentially fire.

Thus if a transition is live, it is always possible to maneuver the Petri net from its current marking to a marking which would allow the transition to fire. A Petri net is live if each of its transition is live.

Chapter 2 STOCHASTIC PETRI NETS

2.1 TIME ACTIVITIES AND PETRI NETS

2.1.1 Non-determinism in PN

The Petri net execution is viewed as occurrence of discrete events in a system. The order of occurrence is not predetermined and the duration of occurrence does not necessarily affect the order of occurrence of future events. This leads to an apparent nondeterminism in Petri net execution. A transition in PN represents occurrence of an event. If, at any time, more than one transition are enabled, then any one of them may be the next one to fire. The choice as to which transition fire is random, based on the randomness involved in the system. Ordinary Petri nets are not fully capable of explicitly representing the time delays, wait and synchronization of events in various real systems.

2.1.2 Time Petri Nets (TPN)

There is no inherent measure of time in the original definition of Petri nets. It considers time only from the logical point of view, in defining the order of occurrence of events. However, in real life, events take variable amount of time. The early Petri net study and application were mostly done in the area of software or protocol verification and the timing considerations were omitted. However, for performance evaluation and scheduling problems of dynamic systems, at present, it is more useful to introduce time as a specification of the system.

Ramachandani [1973] presented timed Petri nets derived from Petri nets by associating a fixed firing time with each transition. He applied this concept in modeling and performance analysis of computer systems. Some researchers has also associated time delays with places. Important work in TPN has also been done by Merlin[1976]. In his model of PN with time, there are two real numbers associated with each transition, representing the minimum and maximum time elapsed for firing a transition, after it has been enabled.

The use of time in Petri nets also allows the determination of some performance measures of the system. However because of the fact that in such models, the transitions must delay a fixed number of steps before it fires. Hence the reachability set of such timed Petri nets may not be the same as of the underlying Petri net. This makes verification and analysis of the PN more difficult. The requirement that a transition "remembers" the number of clock ticks since it was enabled, places severe restriction on TPN analysis. Stochastic activities of dynamic systems are well expressed by allowing transition delay times to be probabilistic.

2.2 STOCHASTIC PETRI NETS

Concept of time in Petri nets is helpful in describing and analyzing the behavior of dynamic systems, such as communication systems, whose behavior is dependent upon the explicit value of time. Moreover, system study also require information about parameters such as, system throughput, or fault rate, or time to reach initial state (cycle time) and so on.

Time considerations are introduced in PN, by assigning firing delay time to each of the transitions. Thus the enabling of a transition, its firing and its completion are considered to take place in a certain amount of time. Some researchers view this time delay as deterministic as in TPN, while some others view time delay of transitions as stochastic. Hence the concept of Stochastic Petri nets (SPN) has evolved.

Molloy [1982] has define SPN as a Petri net in which each transition firing delay is associated with an exponentially distributed random variable.

2.2.1 SPN Definition

Stochastic Petri nets (SPN), are defined as Petri nets which associate a random variable to express the time delay from enabling to firing of a transition to its completion. The variable is mostly considered exponentially distributed.

Stochastic Petri Net is defined as a six-tuple (P, T, I, O, m, F)

where,

1. $P=\{p_1, p_2, ..., p_n\}$, $n > 0$, is a finite set of places.
2. $T=\{t_1, t_2, ..., t_n\}$, $n > 0$, is a finite set of transitions, $P \cup T \neq 0$ and $P \cap T = \phi$
3. $I: P \times T \rightarrow \{0, 1, 2, ...\}$, is an input function that defines the set of directed arcs from P to T

4. O: $P \times T \longrightarrow \{0, 1, 2, \ldots\}$, is an output function that defines the set of directed arcs from P to T.
5. m: $P \longrightarrow \{0, 1, 2, \ldots\}$, is a marking whose ith component represents the number of tokens in the

   i^{th} place. An initial marking is denoted by mo.
6. F : T$\longrightarrow$ R is a firing time delay function with an stochastic distribution function.

2.2.2 Isomorphism of SPN to Markov Chain

Utilizing the memoryless property of the exponential firing distributions, it has been shown [Molloy1982] that the reachability graph of a bounded SPN is isomorphic to a finite Markov Chain. This property of bounded reversible SPN makes it possible to analyze them in terms of their steady-state probabilities and compute important performance parameters.

2.2.3 Extensions to SPN Models

a) Inhibitor arcs

A simple extension to Petri nets is inhibitor arcs. An inhibitor arc from a place p inhibits a transition t, under the rule that t is enabled iff,

$m(p) < k$, where k is the number of inhibitor arcs

Graphically, the inhibitor arc has a small circle at its tail and an arrowhead at the transition.

b) Priorities

Priorities can be associated with the transition such that if t_i and t_k are both enabled, then the transition with higher priority will fire first.

c) Generalized SPN (GSPN)

The generalized SPN introduced by Marson et. al. [1986a] allow the use of immediate transitions (with zero time delay) and probabilities associated with arcs. The GSPN model is an abstract formal graph model to represent concurrency, synchronization, and communication operations at a high level of abstraction.

d) Extended Stochastic Petri Net (ESPN)

An Extended stochastic Petri net (ESPN) is defined as a seven-tuple [Guo 1992],

(P, T, I, O, H, m, F)

where (P, T, I, O, m) are the same as in original definition of PN, namely, places, transitions, input arcs, output arcs and initial marking respectively. The additional terms are:

$H : P \times T \rightarrow N$ is an inhibitor function that defines a set of directed arcs from P to T. It is analogous to a NOT gate in logic theory.

$F: T \rightarrow R$ is a firing time delay function with an extended distribution function. The extended distribution function means that it allows generalized distribution functions for non-concurrent transitions.

2.3 EXECUTION POLICIES FOR STOCHASTIC PETRI NETS

Among the execution policies for execution of Stochastic Petri nets, the two most basic are: race execution policy and preselection policy. In race models, there are three categories: resampling, age-memory and enabling-memory policies [Marson 1986]. Their effects on SPN execution are discussed below.

a) Race Execution Policy

In race-resampling policy, the work done by the non-firing transition is always lost. Under age-memory policy, the work done by the non-firing transitions is not lost and resumed once the corresponding transitions are re-enabled. Under the enabling-memory policy, the work done by the non-firing transitions is not lost if they are still enabled in the next marking obtained after firing of any transition.

b) Preselection Policy

In preselection, the transitions to fire are selected according to their probabilities. There are two preselection policies: global and local. The global preselection policy can be used when all activities represented in the net are sequenced allowing only one of the enabled activity to perform.

It is importantly mentioned here that these policies make no difference when firing delay in SPN is exponentially distributed.

Chapter 3 ANALYSIS OF STOCHASTIC PETRI NETS

3.1 MODELING BY SPN

Stochastic Petri nets, evolved in late 1970's as Petri nets with exponential delay distributions. Important research work by Molloy [1982], Marson [1986a;b], Florin [1989] and Dugan [1985] is reverenced in this regard. These researchers have contributed in theory, structural improvement, implementation and application of SPN in various fields as highlighted below.

a) Communication Systems

Communication systems has a main feature of *synchronization.* Florin and Natkin [1989] by using the existing isomorphism of Markov Process and SPN, has modeled synchronous network queues by SPN. They derived ergodic criterion and steady-state solutions for the model.

b) Local Area Networks

Gressier [1985] has shown modeling of Ethernet protocol by SPN. The results are based upon simulation. Marson [1986c] has computed a performance model for Carrier Sense Multiple Access with Collision Detection protocol (CSMA) of a bus LAN

c) Concurrent / Multiprocessor Systems

Much work has been done recently in these areas. SPN modeling for multiprocessor systems [Marson 1986a], interprocess communication, distributed file systems and concurrent task synchronization has been shown. [Dugan 1987].

d) Manufacturing Systems

SPN has also been used for modelling, design and control of manufacturing systems [Guo 1991; Zhou 1989; Viswanadham 1988].

3.2 TRANSFER FUNCTIONS APPROACH

The evaluation and analysis of Stochastic Petri Nets is proposed to be done by using a methodology which is based on the concepts of Markov theory, control systems and symbolic computation methods. It is called the Moment Generating Function (MGF) or the Transfer Functions approach. This technique has been recently put forward by Guo, DiCesare and Zhou [Guo 1989; Guo 1992]. The improvement, implementation in software and application of this technique, is the scope of this research. The implementation of this technique involves five main steps:

a) Reachability Graph Generation and Transformation to State Machine Petri Net

It has been shown by Molloy and others [Molloy 1982], that the reachability graph of a bounded SPN is isomorphic to a finite Markov Chain. Using this theorem, each marking in the reachabilility graph of the underlying PN is considered as a place of a state machine Petri net.

b) Computation of MGF and Transfer functions

Each transitions of the transformed state machine Petri net is assigned a transfer function, which is the product of branch probability and moment generating function. The transfer function depends upon the firing distribution of the transition and number of markings directly reachable from the marking under consideration.

c) Computation of Equivalent Transfer function

The application of Mason's rule or net reduction techniques leads to computation of equivalent transfer functions of the net. The equivalent function is useful for study and evaluation of the net.

d) Computation of Performance Parameters

Finally, computation of various performance parameters of the SPN model is done by computing derivative of equivalent transfer functions. The implementation of this technique can provide important analytical parameters of the modeled system such as fault rate, conflict rate, deadlocks, production rate, cycle time and system throughput.

The main advantages of this technique are:

1) It does not require simulation of transition firing delays for generation of reachability graph. This technique utilizes the reachability graph of the underlying Petri and imparts the timing information while analysis of the graph.

2) Identifies all possible system states by SPN execution and also indicates system parameters.

3) Provides detection of conflicts and deadlocks in the modeled system, for example, resource allocation problem and buffer overflow problem.

4) Implements net reduction techniques to reduce complexity of the net and ease its analysis.

5) Computation of performance indices of the modeled system. For example, steady-state probabilities, system throughput, fault rrm ate, etc.

3.2.1 Definition of MGF

For a random variable t with probability density function f(t), its Moment generating function (MGF) is defined [Howard 1971] as

Discrete case

$$M(s) = \sum_{-\infty}^{\infty} e^{st} f(t) dt$$

Continuous case

$$M(s) = \int_{-\infty}^{\infty} e^{st} f(t)\, dt$$

where s is an arbitrary parameter and f(t) is a probability density function of random variable t. The n-th derivatives of MGF generates n-th moments of the function.

3.2.2 Properties of MGF

a) The k-th moments are computed as

$$E(t^k) = \left. \frac{\partial^k}{\partial s} M(s) \right|_{s=0}$$

b) According to the definition of the pdf as the summation of probabilities, the value of MGF at s=0 equals to unity.

$$M(0) = \int_{-\infty}^{\infty} f(t) dt = 1$$

MGF for Exponential Distributions

The exponential probability density function is given as,

$$f(t) = \lambda \cdot e^{-\lambda t}, t \geq 0$$

The MGF is computed as

$$M(s) = \int_{-\infty}^{\infty} \lambda e^{(s-\lambda)t} dt = \int_{0}^{\infty} \lambda e^{(s-\lambda)t} dt$$

$$M(s) = \frac{\lambda}{\lambda - s}$$

The moments are,

$$E(t) = \frac{1}{\lambda^2} \quad \text{and} \quad E(t^2) = \frac{2}{\lambda^2}$$

3.2.3 Transfer Functions

The concept of transfer functions from control theory is applied in this analytical method. The procedure is that after obtaining the reachability graph, we transform it to a State-Machine Petri net (SMPN) with single output-single input transitions. We define a transfer function for each transition in the transformed SMPN as the product of MGF and the branch probability of firing P(t) of a transition. Thus a transfer function W(s) can be written as

$$W(s) = P(t).M(s)$$

Transfer function depends upon the marking and distribution of concurrent transitions. If the state i leads only to state j, by firing tj and no other concurrent transition exists at that state, then branch probability is 1.

In case when two transitions t_1 and t_2 with exponential distributions λ_1 and λ_2 are enabled concurrently at a marking, then the transfer functions are

$$W_1 = \frac{\lambda_1}{\lambda_2 + \lambda_1 - s} \qquad W_2 = \frac{\lambda_2}{\lambda_2 + \lambda_1 - s}$$

3.2.4 Net Reduction Methods

Reduction techniques, to reduce the structural complexity of the net, are applied on the SPN. These include sequence, parallel and loop structure reductions [Guo 1992].

a) Sequence Structure

$W_E = W_1 . W_2$

b) Parallel

$W_E = W_1 + W_2$

c) Loop

$$W_E(s) = \frac{W_1}{1 - W_1 W_2}$$

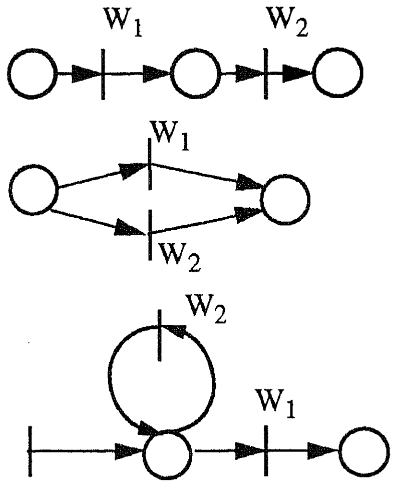


Figure 3.1 Reducible Net Structures

3.3 MASON'S RULE AND ITS APPLICATION TO SPN

Mason's rule [Wang 1991] provides a formula for computation of flow graph gains and relations between system variables. The application of this rule on generated State-Machine Petri net is made because of the fact that a place in the SMPN corresponds to a node in flow graph and the transfer functions of transitions correspond to gain between nodes.

Definition

$$T = \frac{1}{\Delta} \sum_k M_k \Delta_k$$

where,

M_k = path gain of froward path

Δ_k = the value of the part of the block diagram not touching the k-th forward path.

Δ = determinant of the block diagram

Δ = 1 - (sum of loop gains of all loops)
+ (sum of gain products of all possible pairs of nontouching loops)
- (sum of gain of gain products of nontouching loops in triplets)
+ . . .

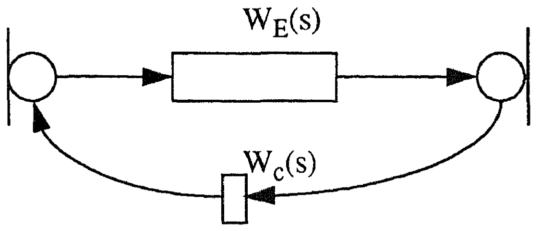


Figure 3.2 Mason's rule example

Using the Mason's rule that the value of loop gain of a closed loop is equal to 1, and using the definition of MGF (eq.) we have,

$$W_E(s)|_{s=0} = P_E . M_E(s)|_{s=0}$$

and $M_E(s)|s=0 = 1$

$$P_E = W_E(s)|s=0 = W_E(0)$$

Hence,

$$M_E(s) = \frac{W_E(s)}{W_E(0)}$$

3.4 COMPUTATION OF PERFORMANCE PARAMETERS

3.4.1 Mean First Passage Time

It is the average time for a system to reach state k, after leaving the state j, In Petri net terms, it is the time in which the marking mj transforms to marking mk, in first execution pass. Computation of this parameter is complex using Markov chain model [Molloy 1989]. However, the computation is simple and direct using the MGF approach.

The mean first passage time from M_0 to Mx is computed as follows:

Step 1) Input arcs to the initial place M_0 are removed.

Step 2) For the modified net, equivalent transfer function is obtained from Mo to Mx, by using Mason's rule.

Step 3) The equivalent MGF of the net is computed and the derivative of this M(s) gives the mean first passage time as:

$$T_{om} = \frac{\partial}{\partial s} M_E(s) \Big|_{s=0}$$

The first passage time variance is also computable from the second moment of the equivalent M(s).

3.4.2 Mean Recurrence Time

It is the expected total time of regeneration of a state. The method for its computation is almost same:

Step 1) Split all loops to a marking Mx by removing input arcs of Mx and directing them to a new introduced place Mx'.

Step 2) For the modified net, equivalent transfer function from Mx' to Mx is obtained, by using Mason's rule.

Step 3) The equivalent MGF of the net is computed and the derivative of this M(s) gives the mean recurrence time.

3.4.3 Mean Sojourn Time

The mean sojourn time [Molloy 1989] of a marking in the SMPN is a special recurrence time. It is the average time for which the system remains at a particular state. It is computed as:

1) Only the output transitions of the considered marking (place in SMPN) are considered timed transitions and all other transitions of the net are considered immediate.
2) The MGF of each transition in the SMPN are computed by retaining the branch probabilities of immediate transitions.
3) The equivalent transfer function and MGF are obtained and the first moment of this $M_E(s)$ gives the sojourn time of the considered place.

3.4.4 Steady-State Probabilities

Computation of steady-state printabilities of states of a PN can be done by Markov chain model for SPN. That technique involve solving of transition parameter matrix and becomes very complicated as the size of net increases [Murata 1989]. By using the MGF approach, we compute the steady-state probability P(m) of a marking (place in SMPN) is computed in terms of recurrence time TR and sojourn time TS as,

$$\mathrm{Prob}(m) = \frac{TS}{TR}$$

Thus, computation of MGF, transfer function and equivalent transfer functions, leads to determination of various performance parameters of the SPN. For example, the time taken to reach a particular state or marking, expected number of tokens in a place, mean number of firings in a unit time and probability of a particular condition may be computed.

Chapter 4 SOFTWARE DEVELOPMENT FOR SPN ANALYSIS

4.1 DESIGN SPECIFICATIONS

The SPN analysis program is developed to meet the thesis objectives, outlined in Section 1.1. It is required that the program should be able to implement the transfer functions approach to various SPN models involving exponential firing distributions of transitions. The program is supposed to carrry out the five main steps involved in this analytical technique of SPN analysis. Those steps are: reachability graph generation of the underlying PN, transformation of reachability graph to state machine PN, computation of transfer functions, application of Mason's rule to find equivalent transfer function and computation of performance parameters. The execution policy of race-resampling is to be followed, such that, in case of conflict transition with minimum firing delay fires first. The program is supposed to handle reasonably complicated Petri nets, including inhibitor arcs and loop structures.

With these specifications in view, the software is developed in UNIX using C language. It is modular and structured. The data structures for the five major steps are devised. The program modules are linked which automates the execution of the program.

4.1.1 Selection of Programming Language

The selection of C language as for code generation of the SPN analysis software was made. C is a modern language having all desirable control features for programmers especially UNIX programmers. C has a wealth of operators, library functions, variety of basic and derived data types, intelligent control structures and powerful pointer system due to which it easily deals with advanced topics like linked lists, arrays, data structures, casts, file operations, classes etc. The design of C also makes it natural to use top-down planning, structured and modular programming. C is not only dominant in UNIX but also being used on PCs.

4.1.2 UNIX Advantages

The main advantages of developing the program on UNIX are following:

1) Availability at the New Jersey Institute of Technology capmus,
2) Powerful incorporated library functions,
3) Easiness of editing, compiling and linking of C and C++ programs on it,
4) Computation power and speed,
5) Possible further enhancement in the program is possible,
6) Use for teaching purpose over the network.

4.2 DATA STRUCTURE FOR SPN AND UNDERLYING PN

Petri nets are defined in terms of their structural entities (places, transitions & arcs) and initial marking. The dynamic behavior of the net is the movement of tokens, firing sequences, change of system states and occurrence of conflicts or deadlocks in the net. In case of Stochastic Petri nets, additional feature is random firing times with a distribution function, associated with transitions. SPN make modeling more explicit in terms of timings involved with events and processes of the modeled system. The software implementation for SPN, necessarily involve development of data structure which should provide following information and properties of the net.

a) Static

1) Description of PN entities - places, transitions and their interconnections in terms of input and output arcs.
2) Initial system state or marking
3) Transition delay parameters.

b) Dynamic

1) Enabling of transitions in a manner, which comply to their time delay specifications.
2) Firing of enabled transitions in accordance with the specified execution policy

3) Conflicts detection

4) Reachability graph generation.

The above objectives are met by the proposed data structure in this thesis.

4.2.1 Data Structure for Net Representation

The Petri net consists of three main entities; places, tokens and transitions. For successful implementation, the data structure should provide complete information about these entities and the static and dynamic behavior of PN. This software is a combination of linked lists, dynamic arrays and arrays of pointers. There are three main structures in the main function; described as follows,

a) Structure for Places

Each place structure has following fields:

Place id - An integer array containing place identification number, which is a unique number for each place in the net.

Tokens - A linked lists of tokens marking each place.

Level - It is linked with the tokens and contains the information of the firing level at which that marking exists

b) Structure for Arcs

Directed arcs of the PN are shown with a positive integer. The value of the integer is the number of arcs. The concept of input and output arcs mapping functions of PN theory is used in this structure.

Input arcs array - It is a two dimensionsal array. First field is the place id form which the arc is originated. The second field of this array is the number of arcs.

Output arcs array - It contains output arcs of each transition. It is also a two dimensional array containing the output place and number of arcs of each transitions.

c) Inhibitor Arcs

The inhibitor arcs, as defined in SPN theory, are directed from places to transitions. These are also included in the data structure for arcs. A two dimensional array, denoted as H is used, which contains the associated place id and transition id of the inhibitor arc.

d) Data Structure for Transitions

All input and output arcs are defined from the point of view of transitions. The structure for transitions therefore contains the I, O and H arrays. It also has a variable "enable" which provides the information about the status of the transition at any execution step. If the transition is enabled the value of the variable is 1 and 0 if it is disabled.

4.2.2 Firing Levels and Sequences

As the execution of the underlying Petri net occurs, there are transitions enabled and are fired one by one. In the developed program, a record of fired transitions at each step is important because the firing of a transition in a marking m, enables other transitions in the next marking m'. The program keeps track of all enabled transitions by assigning a firing level to each marking. The transitions which are enabled at the initial marking are at level 0. One of them fires, and the resulting enabled transitions are at level 1 and so on. The firing level information is used to record firing sequences

4.2.3 History Structure

A data structure named "History" is used to keep record of all firing sequences, enabled transitions and system state (marking), at each execution step. It maintains the following information:

Fired transition - as variable, "trans"

Enabled transitions - in integer array " his[step].enabled[max_p]"

Marking of the net - in " mark[max_p]

Firing level as "lvl"

Execution step count as "step"

4.3 PROGRAM MODULES

4.3.1 Input Module

The developed SPN analysis program is composed of several linked modules. The net structure and delay parameters are user defined a nd are handled by the program input module. The program inputs are:

a) Input arc matrix d) Initial marking

b) Output arc matrix e) Transition delays

c) Inhibitor arcs matrix

The read_net function is used to read all of the above described inputs to the program. A typical input file is shown below:

```
5 4       /* total places are 5 and total transitions are 4*/
1 0 0 0   /* Input arcs  matrix, rows corresponds to places and columns to transitions */
0 1 0 0
0 0 1 0
0 0 0 1
0 0 0 1
1 0 0 0   /* Output arcs matrix*/
0 1 0 0
0 0 1 0
0 0 0 1
0 0 0 0
0 1 0 0   /* Inhibitor Arcs Matrix
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0 0
1 0 0 0 0 /* initial marking*/
1 2 3 4 /* transition firing rates */
```

The function also gives error messages in case of any discrepancies in the net information. The error messages are detailed in appendix.

4.3.2 Function To Check Enabled Transitions

The rule for transition enabling, as described by Murata [1989] and others.

Transition Enable Rule

A transition t_k is said to be enabled if each of its input place p_i, is marked with at least $w(p_i, t_k)$ tokens, where $w(p_i, t_k)$ is the number of arcs from an input place p_i to transition t_k. Mathematically, a transition t_k is enabled if,

$$m(p_i) \geq I(p_i, t_k) \qquad i = 1, 2, ..., |P|$$

Inhibitor arcs case

The inclusion of innhibitor arcs necessitates another condition to be added to the enable rule. A transition with inhibitor arcs is enabled if,

$$m(p_i) < H(p_i, t_k) \ , \ i = 1, 2, ..., |P|$$

chk_enabled_trans( )

A vsriable "enable"' is defined for status of transition at every step. An enabled transition is indicated as 1, while a disabled as 0. The logic used in this function for checking of transitions is that at first all transitions are initialized as enables. Then the transition enable rule and inhibitor arcs condition are applied and those which donot conform to the conditions are tagged as disabled. The parameter passed to the function is the firing level. The function cehcks all enabled transitions, updates the linked lists of transitions and also returns the total number of enabled transitions.

The logical flow of the function is given below.

```
LOOP1) For each transition t of the incidence matrix do loop1:
        1.1) Scan the transition input arcs array to find all input places of
        the transition t
    LOOP2) If there is any input place to t, do following, otherwise go
            to step 1.3.
            2.2.1) For a transition t find the number of input arcs between an
            input place p to the transition t.
```

2.2.3) Is tokens in theplace p are less than the number of input arcs existing between p and t. If not, go to step 2.2.7.

2.2.4) If yes, Switch value of array 'trans_enabled', from 1 to 0, indicating that t is disabled in this marking.

2.2.5) For a transition t find the number of inhibitor arcs between an input place p to the transition t.

2.2.6) Continue loop2 until all input places of t are checked.

2.2.7) Disable the transition.

1.2) Check for next transition and continue loop1 until all transitions are checked.

1.3) Terminate function

The return value of the function will be the id numbers of enabled transitions marked as 1 in the "t[i]. enable" array and their sum.

4.3.3 Firing an Enabled Transition

A PN executes by firing transitions. A transition may fires if it is enabled. The tokens in the input places which enable a transition are its enabling tokens. Firing of a transition is a two step process,

a) Removal of tokens

Enabling tokens are removed from each of the input places of the firing transition. The number of tokens removed equals the number of input arcs to the fired transition.

b) Addition of tokens

New tokens are distributed to all output places of the fired transition. The number of tokens put in the each output place of the fired transition equals the number of output arcs between the transition and the output place. The function "*move_tokens( )*" is used in the program to do this job.

4.3.4 Conflict Detection by Chk_Conflict( )

A conflict in PN is defined as a situation when, at a given marking m, firing of a transition disables any other transition which was also enabled at m. The developed C program has a conflict detection function, called internally while executing the PN. The parameter passed to this function is the execution step of the program. The function utilizes the information of firing level of enabled transitions, maintained in a linked array of transitions. The logic flow of the function is described below.

1) At initial marking m_0, all enabled transitions are known after use of "trans_enable" function. Additional information of firing level and token assignment is also linked with it.
2) One of the enabled transition is fired and new marking m' is obtained.
3) The "trans_enable" function is again called and the transitions enabled in the marking m' are compared to transitions enabled in previous marking m. If any of the previously enabled transition is now disabled in the marking m', it is a conflict situation.
4) All of the transitions which are in conflict are tagged accordingly.

4.3.5 Reachability Graph Algorithm and Implementation by Function Firing( )

The Reachability graph of a PN is a set of all reachable markings (states) from an initial marking m_0 (initial state). Given a PN, we can obtain as many new markings as the number of enabled transitions. From each new marking, we can reach more markings. This process results in a tree representation of the markings, which is known as the *Reachability Graph.* The generation of reachability graph in the program is done by the function firing( ). It is one of the key functions of the program and calls several other functions during execution. It also calls itslef for next firings until it termintes upon some conditions.

Unfortunately, no algorithm was found which deal exactly with bounded PN and SPN with inhibitor arcs. Thus in this thesis a modified algorithm based on the the algorithms described by Murata[1989] and others is proposed in this thesis. The following are the details the implementation of the algorithm follows.

Step 1) Label the initial marking as the root and tag it "m_0" and initialize execution step i=0 and level l=0.

Step 2) Find all enabled transitions at marking *m* and level *l*. If none exists tag it dead and go to termination step. If there are any enabled transitions, continue.

Step 2.1) Fire an enabled transition *t* to find a new marking *m* store it as a new marking at level *l +1*. and increment land execution step count.

Step 2.2) Compare if *m* is identical to any previous marking. If yes, then tag *m* as "old". If not, go to next step

Step 2.3) If the marking at level *l+1* disabled any ofthe transitions enabled at the marking at level *l*, then store it as a conflict situation. If not go to next step.

Step 2.4) Continue to fire the transitions along the depth of the tree, increasing firing level each time. Tag them as new and old accordingly.

Step 2.5) If no new markings are being found along the depth (increasing level) of the tree, then stop execution by announcing a non-terminating loop.

Step 3) Consider all of the conflict situtuations, starting from the first one, and repeat all substeps of step 2.

Step4) Terminate the algorithm when all conflict situations at all levels have been checked and no new markings can be found further.

4.3.6 Transfer Functions Generation

After the reachability graph of the net is obtained. It is considered as a state machine PN with each marking as a place. Transfer functions for each branch of the state machine PN are computed. Transfer function is the product of the branch probability and MGF of the transition. The program checks the branchings in the SMPN, generates the transfer functions for each branch, indicating the origin and desstination place (marking) and sends output to a log file.

4.3.7 Program Output Function

The program generates the following:

a) All new states of the net

b) Reachability graph of the net, display it and also store it in a log file

c) All firing sequences

d) Transfer functions, currently for exponential and immediate transitions

e) Predefined performance parameters from the derivative of the equivalent transfer function

It is done by the ouput function called *out( )*. The outputs are stored in separate log files.

4.3.8 The Derivative Program

The derivative program is a simpler program which computes the first order partial derivative of the equivalent transfer function. The module of Mason rule implementation is not linked with it yet so the program at the moment, requires the equivalent transfer function to be made input to it. The program reads in the values of transition distribution parameters (lambdas), the individual transfer functions and the equivalent function. The derivative of the equivalent transfer function is computed as,

$$\frac{\partial}{\partial s}W_E(s) = \frac{W_E(\Delta s) - W_E(0)}{\Delta s}$$

The program is written temporarily until the module for Mason rule implementation is available. The Mason rule module will automate the generation of equivalent transfer function and its derivative and the above described derivative module will be eliminated.

Chapter 5 APPLICATIONS OF DEVELOPED SOFTWARE

5.1 EXAMPLE FROM COMPUTER SYSTEMS

The computer systems with shared resources often have the problems of resource sharing conflicts and deadlocks. The problem of mutual exclusion and buffer overflows are also common. These situations are easily be modeled by Petri nets. SPN also provide a powerful mathematical model to deal with such problems. Consider the SPN model [Molloy 1982] of a computer system, as shown in Figure 5.1.

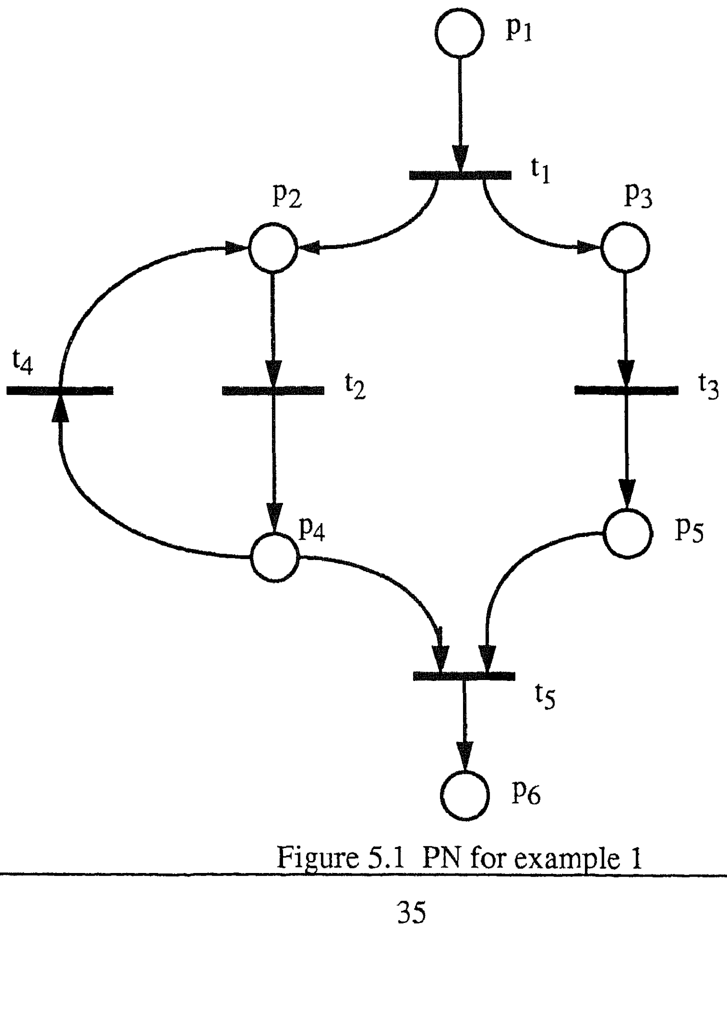


Figure 5.1 PN for example 1

This SPN displays :

sequential operation - t_5 and t_1

parallel operation - t_2 and t_3

forking - t_1

joining - t_5

contention - t_4 and t_5.

Assume the initial marking of only one token in place p1 and no tokens in the remaining places. The net input file is created as "e4.in". It contains the input and output arcs matrices, initial marking and transition delay parameters.

5.1.1 Reachability Graph generation

Running the program input file "e4.in" on the developed software, generates reachability graph of the net in figure 5.2. It has five reachable states (M_0 through M_4) as shown in table 1 below.

Table 1: Markings reachable from M_0

Marking	p_1	p_2	p_3	p_4	p_5
M_0	1	0	0	0	0
M_1	0	1	1	0	0
M_2	0	0	1	1	0
M_3	0	0	0	1	1
M_4	0	1	0	0	1

The reachability graph generated by the program is shown in matrix form in the appendix.

Transition Parameters

t_1: 2

t_2, t_3: 1

t_4: 3

t_5: 2

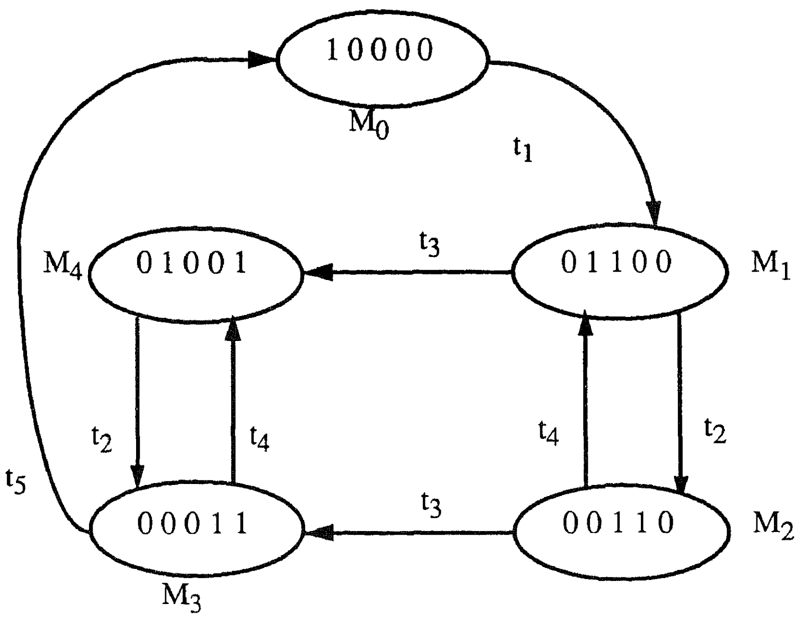


Figure 5.2 Reachability Graph of Molloy's Example

5.1.2 Computation of Transfer Functions

The reachability graph is transformed to a state machiune Petri net by adding transitions between each marking of the graph. Next, the transfer functions of each transition in the SMPN are computed by the program and are stored in a transfer functions log file (shown in apppendix). The individual transfer functions of the SMPN are computed by the program. The final SMPN is shown in Figure 5.3.

$$W_1 = \frac{\lambda_1}{\lambda_1 - s}$$

$$W_2 = \frac{\lambda_2}{\lambda_2 + \lambda_3 - s}$$

$$W_3 = \frac{\lambda_3}{\lambda_3 + \lambda_2 - s}$$

$$W_4 = \frac{\lambda_2}{\lambda_2 - s}$$

$$W_5 = \frac{\lambda_4}{\lambda_4 + \lambda_5 - s}$$

$$W_6 = \frac{\lambda_3}{\lambda_3 + \lambda_4 - s}$$

$$W_7 = \frac{\lambda_4}{\lambda_4 + \lambda_3 - s}$$

$$W_8 = \frac{\lambda_5}{\lambda_5 + \lambda_4 - s}$$

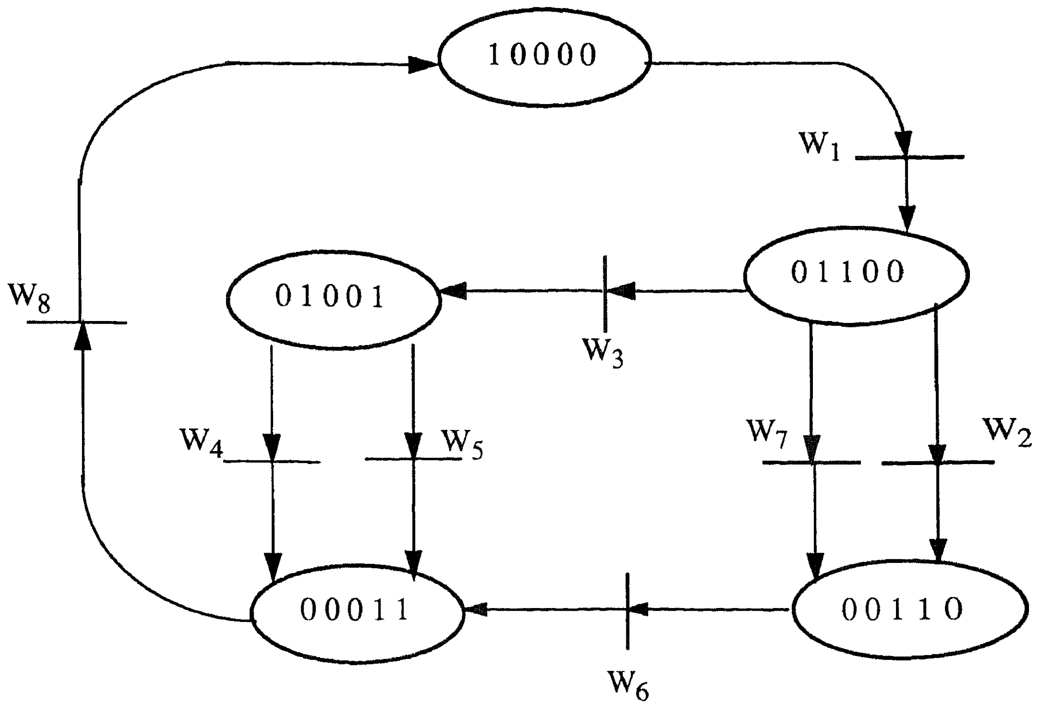


Figure 5.3 State Machine Petri Net Transformation

5.1.3 Computation of Recurrence time of a marking

Consider M_0 in the SPN reachability graph. Using the procedure (described in section 3.5.2), the input arcs to M_0 are detached from it and directed into an assumed marking $M_{0'}$. The Masons' rule is used to compute the equivalent transfer function.

a) Forward paths

path 1 = M_0-M_1-M_2-M_3-$M_{0'}$	pathgain 1 = $g_1 = W_1.W_2.W_6.W_8$
path 2 = M_0-M_1-M_4-M_3-$M_{0'}$	pathgain 2 = $g_2 = W_1.W_3.W_4.W_8$

b) Loops

loop1 = M_1-M_2-M_1 loop gain 1 = $l_1 = W_2.W_7$

loop2 = M_3-M_4-M_3 loop gain 2 = $l_2 = W_5.W_4$

We have

Δi = 1 - (loops not touching path i)

$\Delta_1 = 1 - 0 = 1$

$\Delta_2 = 1 - 0 = 1$ and

c) Denominator

Δ = 1 - (sum of loop gains) + (product of any two nontoucinhg loops) - (product of any three nontouching loops) - ...

$= 1 - (l_1 + l_2) + (l_1.l_2)$

$= 1 - (W_2.W_7 + W_5.W_4) + (W_2.W_7.W_5.W_4)$

d) Equivalent transfer function

Derived from M_0 to M_0' is,

$$W_E(s) = \frac{g_1\Delta_1 + g_2\Delta_2}{\Delta} = \frac{W_1W_2W_6W_8 + W_1W_3W_4W_8}{1 - ((W_2W_7 + W_5W_4) + (W_2W_7W_4W_5))}$$

e) Computing the value of the equivalent transfer function at s=0 we get,

$W_E(0) = 1$

Next, the recurrence time of M_0 is computed as

$$TR_{00} = \frac{\partial}{\partial s} W_E(s) \Big|_{s=0}$$

We use an approximate method to find the derivative of the equivalent function, which has been implemented as a separate program module called "drvt", shown in Appendix. Using module "drvt", we get the derivative values for all markings in the net.

5.1.4 Sojourn Time

The sojourn time of a state is the time for which the system remains in that state before transiting to another state. It is computed (detailed in sub-section 3.5.3) by assuming the outgoing transitions of the marking M_7 as timed transitions and the other transitions are assumed immediate. Once again the equivalent function and of the marking is computed by steps (3.a) thru (3.e). The derivative of the equivalent transfer function at s=0, gives the mean sojourn time of the marking.

$$TS_{00} = \frac{\partial}{\partial s}\overline{W_E(s)}\Big|_{s=0}$$

and is computed by derivative program module. The computed value is,

$$TS_{00} = 3.000$$

5.1.5 Steady-State Probability

The steady-state probability is give as the ratio of sojourn time and recurrence time of a marking. For M_0,

$$P(M_0) = \frac{TS_{00}}{TR_{00}}$$

The steady state probabilities for other markings $M_1$to M_4 are also computed in the similar manner.

The computed results and the published results of Molloy [1982] are compared below.

Table 2: Comparison of Computed Probabilities

Marking	Computed results	Molloy's results
P[1 0 0 0 0]	0.1128	0.1163
P[0 1 1 0 0]	0.1848	0.1860
P[0 0 1 1 0]	0.0488	0.0465
P[0 0 0 1 1]	0.1124	0.1163
P[0 1 0 0 1]	0.5231	0.5349

The results are found to be within 5% deviation to the results in Molloy [1982].

5.2 EXAMPLE FROM FLEXIBLE MANUFACTURING SYSTEMS

Flexible manufacturing systems (FMS) are systems with automated machines, interconnected by automated material handling. The design, operation and control of these systems have to take into account numerous interactions occurring between concurrent and non-deterministic activities. The system is categorized as a discrete-event dynamic system. In these systems, the important performance parameters are; machine utilizations, production rates, average queue sizes and waiting times.

Consider a FMS, which has two work-stations and two robots [Zhou 1991]. The workstations do identical job, which is, assembly of parts with the help of both robots. To provide mutual exclusion and description simple, we assume that each workstation acquires its right robot first and then the robot on its left, to assemble the parts. All these activities take stochastic time, which is exponentially distributed.

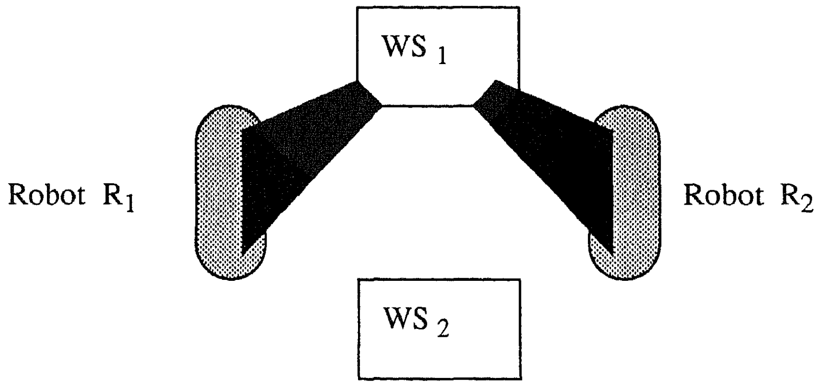


Figure 5.4 Example of a Two Robot -Two Workstations FMS system

An inherent deadlock is present in the system, because each workstation will acquire a robot and wait forever to acquire the other. This deadlock is resolved by the addition of a transition to move the system from deadlock state. The firing rate of that transition will indicate the deadlock rate of the system.

The system specifications and their modeling by SPN is done as below.

a) When a WS is ready to assemble the parts, it requests the right robot and, if it is available, acquires it and then requests its left robot.

b) Each WS takes exponentially distributed time to assemble parts.

c) The job, once started on a WS cannot be interrupted or halted.

d) After performing the assembly of parts, the WS releases the robot. All activities, such as, robot request, robot access, part assembly, robot release etc. take independently and exponentially distributed time.

5.2.1 SPN Model

The SPN model of the above described FMS example is made. The general synthesis procedure is used.

a) Identify system resources and all dynamic activities or operations of the system. Generally each such activity or event is represented as a place. For the given system we have System Resources

Workstation1 = WS_1 Workstation2 = WS_2 Robot 1 = R_1 Robot 2 = R_2

Events/Operations

p_1: WS_1 requests R_1

p_2: WS_1 requests R_2

p_3: WS_1 has both R_1 and R_2

p_4: WS_1 ready to release Robots

p_5: WS_2 requests R_2

p_6: WS_2 requests R_1

p_7: WS_2 has both Robots

p_8: WS_2 ready to release Robots

p_9: R_1 available

p_{10}: R_2 available

b) The start or stop of an activity can be represented by a transition.

Transitions

t_1: WS_1 acquires R_1

t_2: WS_2 acquires R_2

t_3: Assembling parts at WS_1

t_4: Releasing robots from WS_1

t_5: WS_2 acquires R_2

t_6: WS_2 acquires R_1

t_7: Assembling parts at WS_2

t_8: Releasing robots from WS_2

c) Identify the relationships involved like mutual exclusion or concurrency of events involved with resources.

d) Draw input and output arcs, depending upon the precedence relations and dependence among elements of the system.

e) Specify the initial state (initial marking) of the system.

Using the steps a) through e) we form the Petri net for this system as drawn below.

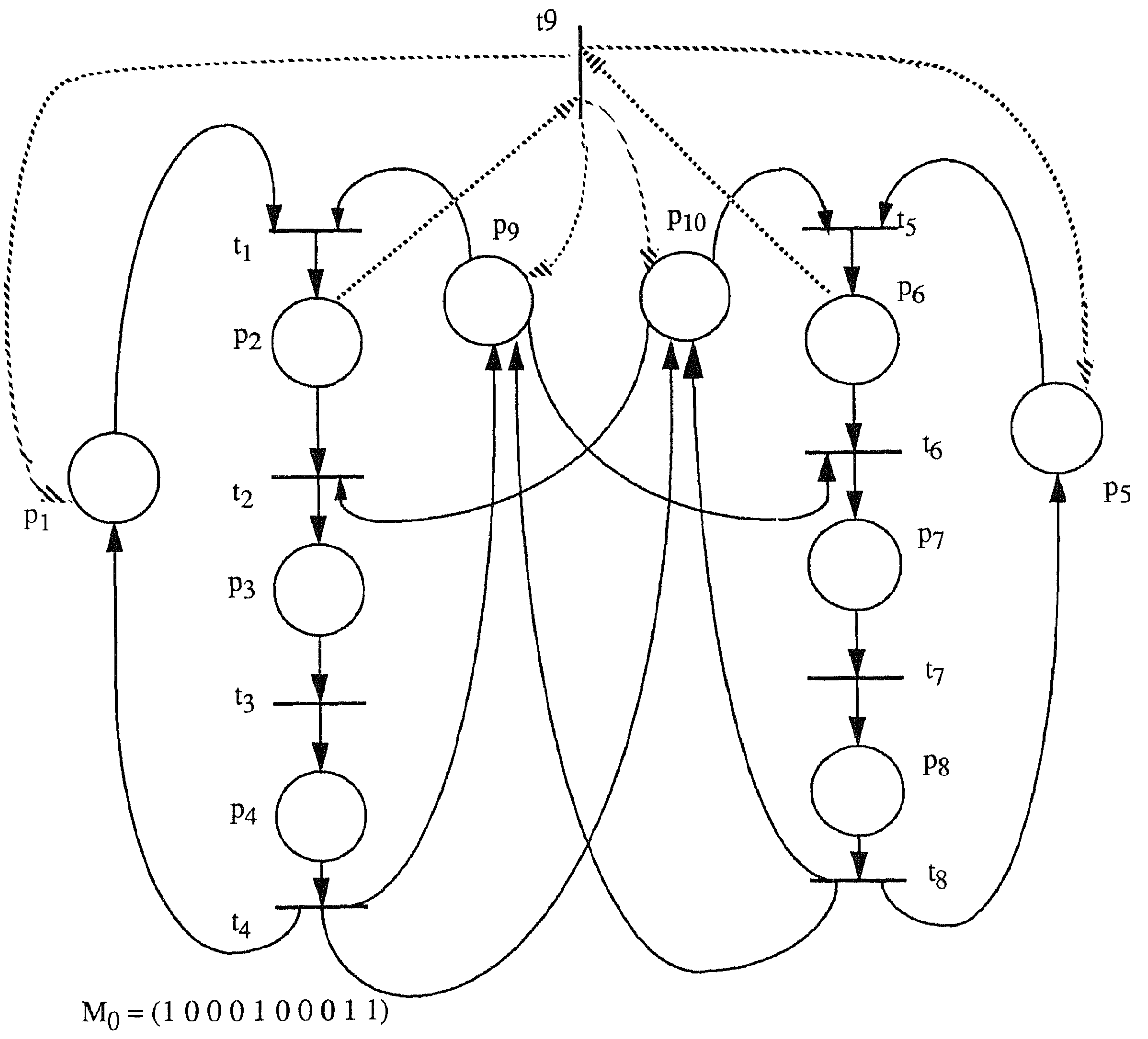


Figure 5. 5 The SPN model of FMS example
with deadlock of the system resolved by adding t_9 and dotted arcs

The incidence matrix, transition rates and initial marking of the net in Figure 5.5 made input to the SPN analysis program as file "zhu.in", shown in the appendix. The program identifies 7 new reachable markings of the net and the generated reachability graph is shown in the Figure 5.6.

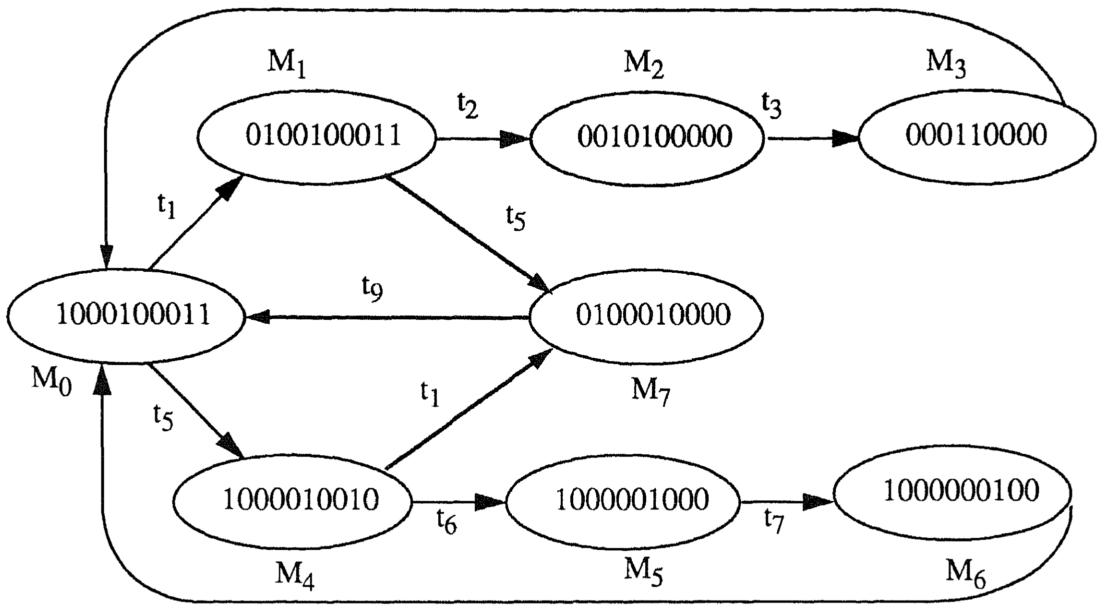


Figure 5.6 Reachability Graph of FMS example

In this example, we are interested to find the deadlock rate of the system, which will be the firing rate of transition t_9. We also need to find the production rate of both workstations if the deadlock resolution, by addition of t_9, is used.

5.2.2 Computation of Transfer Functions

The reachability graph of Figure 5.6 is converted to state machine Petri net and the transfer functions for each transitions are also generated by the program. It is shown in Figure 5.7.

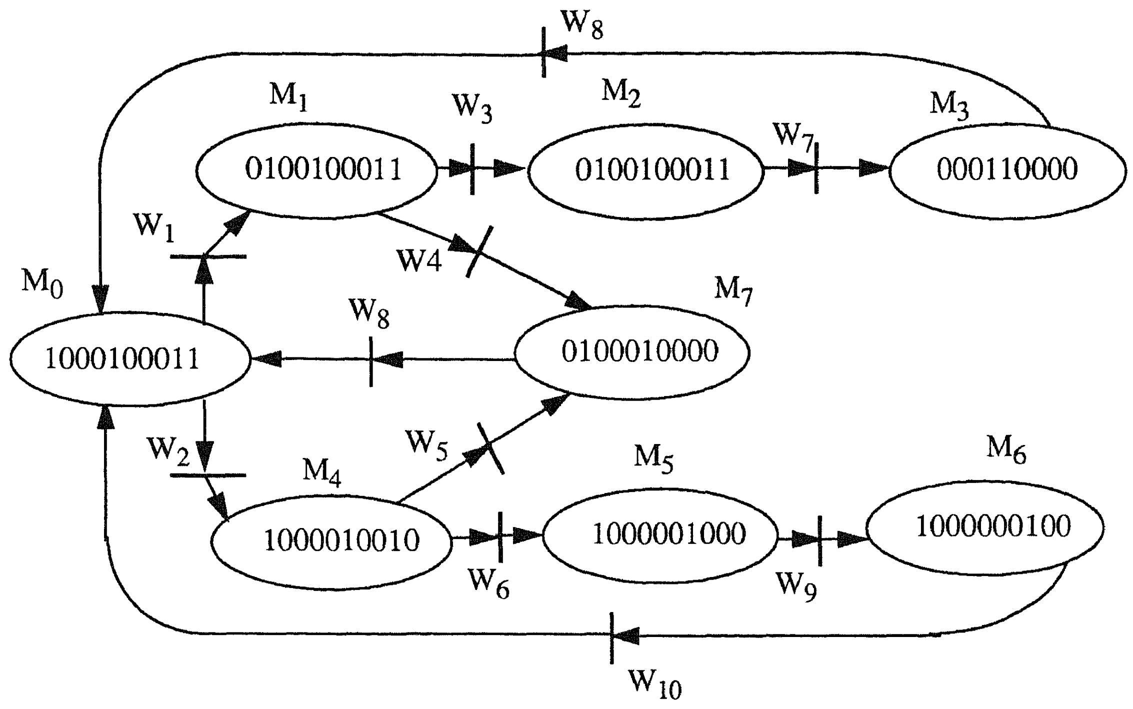


Figure 5.7 The tranformation of SPN to state machine PN.

$$W_1 = \frac{\lambda_1}{\lambda_1 + \mu_1 - s} \qquad W_2 = \frac{\mu_1}{\lambda_1 + \mu_1 - s} \qquad W_3 = \frac{\lambda_2}{\lambda_2 + \mu_1 - s}$$

$$W_4 = \frac{\mu_1}{\lambda_2 + \mu_1 - s} \qquad W_5 = \frac{\lambda_1}{\lambda_1 + \mu_2 - s} \qquad W_6 = \frac{\mu_2}{\lambda_1 + \mu_2 -}$$

$$W_7 = \frac{\lambda_3}{\lambda_3 - s} \qquad W_8 = e^{c_1 s} \qquad W_9 = \frac{\mu_3}{\mu_3 - s}$$

$$W_{10} = e^{c_2 s} \qquad W_{11} = e^{c_3 s}$$

5.2.3 Computation of Recurrence Time

The recurrence time of each marking (place in SMPN) can be computed from the equivalent transfer function. The net reductions of sequence and loop structures in the net are also possible as in [Guo 1990]. The recurrence time and sojourn time computation is required to find the deadlock rate of the system. To find recurrence time of M_7, cut the flow from transition t_9 (W_{11})
and add a sink place M_0'. The net is shown in Figure 5.8 below.

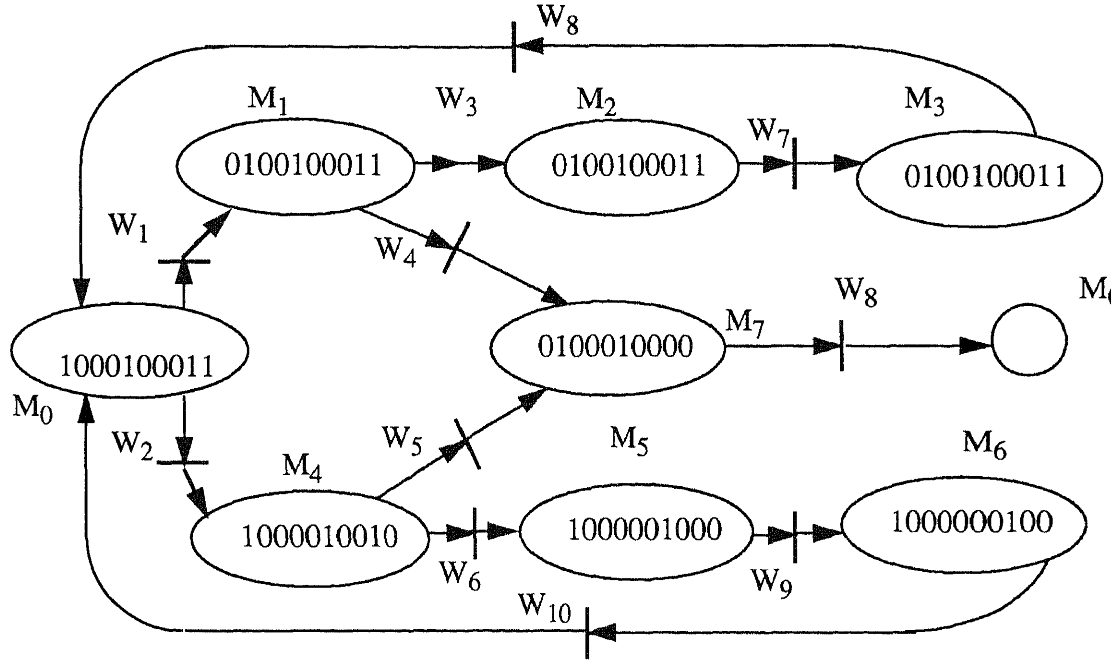


Figure 5. 8 Net for solution of deadlock rate

The equivalent transfer function from M_0 to M_0' is computed by Mason rule,

$$W_E(s) = \frac{(W_1W_4+W_2W_5)W_{11}}{1-W_1W_3W_7W_8-W_2W_6W_9W_{10}}$$

The mean recurrence time of the system is computed as the derivative of the equivalent transfer function at s=0.

$$'R_{00} = \frac{\partial}{\partial s}M_E(s)\Big|_{s=0} = 8.628$$

Computation of Sojourn time

The Sojourn time is computed by considering the transition w_{11} as timed and all others as immediate. The equivalent function is computed and its derivative at s=0, gives the value of sojourn time.

$$TS_{00} = \frac{\partial}{\partial s}\overline{M_E(s)}\Big|_{s=0} = 3.000$$

Computation of Deadlock Rate

The deadlock rate R_d is the ratio of mean sojourn time and mean recurrence time of t_9.

$$R_d = \frac{TS_{00}}{TR_{00}} = 0.3477$$

Throughputs of Workstations

Using the same procedure, we can compute the production rates of the two workstations in the example. The production rate of each Workstation are computed as the ratio of their sojourn time to recurrence time.

Table 3 Summary of Computed Results for FMS Example

Associated Marking	$W_E(0)$	Recurrence Time	Sojourn Time
M_8	1	8.628	3.000
M_4	1	15.680	2.000
M_7	1	8.407	2.000

Table 4 Performance Parameters of FMS Example

Performance Parameter	Computed Value
Deadlock Rate	0.3477
Production Rate of WS_1	0.127
Production Rate of WS_2	0.2378

Chapter 6 CONCLUSION AND FUTURE RESEARCH

6.1 FEATURES OF THE DEVELOPED SOFTWARE

The analytical software for SPN analysis using the transfer function approach is developed. The main features of the software are higlighted below.

a) Reachability Graph Generation

The program identifies all reachable states of the modeled system, from a given initial state and records all firing sequences for a bounded PN.

b) Conflict and Deadlock Detection

Possible deadlocks and conflicts of shared resources are detected. The results computed by the program can be used effectively for possible resolution of such situations.

c) Inhibitor Arcs

An important design aspect of the developed software is that it allows inclusion of inhibitor arcs in SPN. The inhibitor arcs are used to effectively model failure or error situations, such as, machine breakdowns in manufacturing systems or error in message transmission in communications. Inhibitor arcs can also model mutual exclusion of operations or processes.

d) Transfer Functions

The program computes individual transfer functions of the transformed SMPN, as the product of branch probability and moment generating function.

e) Performance Parameters

The MGF technique uses transfer functions to compute sojourn time, recurrence time and steady-state probabilities of system states. These parameters can then be interpreted as the performance parameters of the modeled system, for example, production time and deadlock rate.

6.2 PROGRAM LIMITATIONS

Most of the program limitations are a reflection of the limitations of the implemented technique and are indicated below.

a) State Explosion

The generation of a large number of system states in the reachability graph, is a major hindrance in study and evaluation of Petri nets and SPN [Molloy 1989; Murata 1989]. Reduction techniques or approximate approaches are required in the analysis of a complicated SPN.

b) Unbounded Petri Nets

In case of unbounded Petri nets execution, an infinite number of states would be generated. Analysis of such a reachability graph by MGF approach could only be done theoretically and it may not lead to exact solutions for SPN.

c) Execution Policies

The present software is based on the execution policy of race-resampling. The execution policies have important effect on semantics of extended models of SPN [Marson 1989]. Their addition to the existing software will become necessary when enhancements in the software are made for inclusion of non-exponential transitions.

6.3 FUTURE RESEARCH DIRECTIONS

a) GSPN and ESPN

These extended models of SPN also include some distributions other than exponential. The inclusion of such models will enhance capacity of the software and would make it more general. But this will require an extensions to the data structure and transfer function generation methodology, because of the complex effects of execution policies and corresponding Markov process.

b) Net Reduction Techniques

The state explosion problem of Petri nets necessitates further research in reduction techniques or approximate approaches in evaluating an SPN. With the help of transfer functions and net reduction techniques , the SPN analysis can be simplified [Guo 1991].

c) Unbounded SPN analysis

The implemented MGF technique can be improved further to deal with unbounded SPN. The use of the algorithm published in the survey of Murata [1989], can lead to generation of reachability graph of infinite cases, with concept of representing infinite markings as "ω". But research is required for extracting information leading to performance analysis of the SPN.

d) Mason's Rule Implementation for General Cases

The implementation of Mason's rule for complex loop structures has not been carried out at the moment, due to data structure complexity factors. This general implementation is only worthwhile when more sophisticated net reductions are required for very complex and large reachability graphs.

e) Effect of Execution Policies

The effect of execution policies on SPN analysis becomes important when non-exponential distributions are considered. The implications of execution policies for different SPN models need to be further studied and methods for deriving the transfer functions need to be devised. The analysis of non-exponential firing distributions of transitions will entail more complexities.

6.4 CONCLUSION

The research objectives of development of a software tool for study, evaluation and analysis of Stochastic Petri nets have been partially completed. The developed software implements a closed-form analytical technique for bounded Stochastic Petri nets, known as the *transfer function approach.*

Data structures for Petri net representation, reachablility graph, state machine Petri net transformation and transfer functions generation are devised. Equivalent transfer functions are computed by application of the Mason's rule. Performance parameters are computed from equivalent transfer functions and their derivatives. The software is developed in UNIX, using the C language and deals with bounded SPN of reasonable size. It is modular and expandable. A version of this software in C++ has also been developed for its use in a PC environment

The analysis of SPN by the developed program has the advantages of direct computation of system states, detection of resource deadlocks, calculation of transfer functions and the computation of performance indices of the system modeled by the SPN. It is expected that the developed software will facilitate modeling, analysis and evaluation of SPN for academic scholars, engineers and researchers.

REFERENCES

[Chiola 1985]	G. Chiola, "A Software Package for the Analysis of Generalized Stochastic Petri Net Models," *IEEE Proceedings International Workshop on Timed Petri Nets*, Torino, Italy, July 1985.
[Dugan 1985]	J. B. Dugan, A. Bobbio, A. Ciardo and K. S. Trivedi, "The Design of a Unified Package for Solution of Stochastic Petri Net Models," *International Workshop on Timed Petri Nets*, Torino, Italy, July 1985.
[Dugan 1987]	J. B. Dugan, A. Bobbio, A. Ciardo, " Stochastic Petri Net Analysis of a Replicated File System," *IEEE-CS press, Int. Workshop on Petri Nets and Performance Models*, Wisconsin, August 1987.
[Florin 1989]	G. Florin, S. Natkin, " Necessary and Sufficient Ergodicity Condition for open synchronized queuing networks," *IEEETransactions on Software Engineering*, vol. 15, no. 4, pp. 367-380, 1989.
[Gressier 1985]	E. Gressier, " A Stochastic Petri Net Model for Ethernet," Proceedings of Int. Workshop on Timed Petri Nets, Torino, Italy, July 1985.
[Guo 1991]	D. L. Guo, F. DiCesare, M. C. Zhou, " Moment Generating Function Approach to Performance Analysis of Extended Stochastic Petri Nets," *Proceedings of IEEE Int. Conference on Robotics and Automation*, pp. 1309-1314, Sacramento, CA 1991.

[Guo 1992]	D. L. Guo, F. DiCesare, M. C. Zhou, " A Moment Generating Function Approach for evaluating extended Stochastic Petri Nets," *To appear in, IEEE Transactions on Automatic Control,* November 1992.
[Ho 1987]	Y. C. Ho, "Performance Evaluation and Perturbation Analysis of Discrete Event Dynamic Systems," *IEEE Trans. on Automatic Control*, Vol. AC-32, No. 7, pp. 563-572, 1987
[Howard 71]	R. A. Howard, *Dynamic Probabilistic Systems*, John Wiley, New York, NY 1971.
[Lee 1987]	K. H. Lee, J. Favrel, and P. Baptiste, " Generalized Petri net Reduction method," *IEEE Trans. Systems, Man, and Cybernetics*, SMC-17, No. 2, pp. 297-303, 1987.
[Marson 1984]	M. A. Marson and G. Chiola, "A Class of Generalized Stochastic Petri Nets for the Performance Analysis of multiprocessor Systems," *ACM Trans. on Computer Systems*, vol. 2, no. 2. 1984, pp. 93-122.
[Marson 1986 a]	M. A. Marson, G. Balbo, A. Bobbio, G. Chiola, G. Conte, A. Cumani, " The Effects of Execution Policies on the Semantics and Analysis of Stochastic Petri Nets," *IEEE Transactions on Software Engineering*, vol. 15, no. 7, pp. 832-846, 1986.
[Marson 1986b]	M. A. Marson, G. Chiola and A. Fumagalli, "An Accurate Performance Model of CSMA/CD bus LAN," *Proc. of* 7th *European Workshop on Application and Theory of Petri Nets,* Oxford, England, June 1986.

[Merlin 1976] P. Merlin, "A Methodology for the Design and Implementation of CommunicationsProtocols,"*IEEE Transactions on Communictions,* pp. 614-621, 1976.

[Molloy 1982] M. K. Molloy, " Performance Analysis using Stochastic Petri Nets," *IEEE Transactions on Computers*, vol. C-3, no. 9, pp. 913-917, 1982.

[Murata 1989] T. Murata, "Petri Nets: Properties, Analysis and Application," *Proceedings of the IEEE,* vol. 77, no. 4, pp. 541-579, 1989.

[Molloy 1985] M. K. Molloy, "Discrete Time Stochastic petri Nets," IEEE Trans. on Software Engineering vol. SE-11, no.4, pp. 417-423, April 1985.

[Patel 1991] P. Patel, *Petri Net Modeling and Analysis of a FMS Cell,* Master's thesis in Manufacturing Engineering, New Jersey Institute of Technology, 1991.

[Peterson 1977] J. L. Peterson, "Petri nets", *ACM Computing Surveys*, Vol. 9, pp. 223-252, September 1977

[Peterson 1981] J. L. Peterson, *Petri Net Theory and Modeling of Systems.* Englewood, Cliffs, NJ, ,Prentice-hall, 1981.

[Ramachandani 1974] C. Ramchandani, "Analysis of Asynchronous Concurrent Systems by Timed Petri Nets," Ph.D. dissertation, MIT, Cambridge, Project MAC Rep. MAC-TR-120, 1974.

[Sifakis 1977] J. Sifakis, "Petri Nets for Performance Evaluation," *Measuring, modeling and evaluating computer systems, Proc. 3rd Int. Symp. IFIP Working Group 7.3,* H. Beilner and E. Gelenbe, Eds., North-

Holland Publishing Company, pp. 75-93, 1977.

[Viswanadham 1988] N. Viswanadham, Y. Narahari, "Stochastic Petri Net Models for Performance Evaluation of Automated Manufacturing Systems," *Information and Decision Technologies*, 14, North-Holland, pp. 125-142, 1989.

[Wang 1991] Chi-Hsu Wang, "Computer-Aided Manipulation of Mason's Formula and Its Applications," Proc. of IEEE Int. Conference on System, Man and Cybernetics,pp. 493-500, Charlotteville, VA 1991.

[Zhou 1989] M. C. Zhou and F. DiCesare, "Adaptive Design of Petri Net Controllers for Error Recovery in Automated Manufacturing Systems," *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-19, 5, pp. 963-973, 1989.

[Zhou 1991] M. C. Zhou, D. L. Guo and F. DiCesare, "Integration of Petri Nets and Moment Generating Function Approaches for System Performance Evaluation," Working paper, New Jersey Institute of Technology, 1991.

APPENDIX

/* INPUT FILE "e4.in" for Example in Section 5.1 */

```
5 5  /* max places and transitions*/
/* Input Arcs Matrix*
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 1
0 0 0 0 1
/* Output Arcs Matrix*/
0 0 0 0 1
1 0 0 1 0
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
/* Inhibiotr Arcs Matrix  */
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
/* Initial marking*/
1
0
0
0
0
/* Transition parameters */
2
1
1
3
2
```

/* **Output File For e4.in (ref. Section 5.1)** */

```
Data read from input file:
Input Arcs    :
   1   0   0   0   0
   0   1   0   0   0
   0   0   1   0   0
   0   0   0   1   1
   0   0   0   0   1
Output Arcs :
   0   0   0   0   1
   1   0   0   1   0
   1   0   0   0   0
   0   1   0   0   0
   0   0   1   0   0
Inhibitor Arcs :
   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0
Initial markings
       1
       0
       0
       0
       0
lamda
2.000000
1.000000
1.000000
3.000000
2.000000
A loop found.
A loop found.
A loop found.
A loop found.

Firing Sequences

Fire/enabled/l=0  1  2  3  4  5  6  7  8  9  0
t0/ 1          1  0  0  0  0 New m0
t1/ 2 3           0  1  1  0  0 New m1
```

```
t2/ 3 4              0   0   1   1   0 New m2
t3/ 4 5                  0   0   0   1   1 Conflict New m3
t4/ 2                        0   1   0   0   1 New m4
t2/                              0   0   0   1   1 m3
t5/                          1   0   0   0   0 m0
t4/                      0   1   1   0   0 m1
t3/                  0   1   0   0   1 m4

Reachability Graph

     m0  m1  m2  m3  m4
m 0   0   1   0   0   0
m 1   0   0   2   0   3
m 2   0   4   0   3   0
m 3   5   0   0   0   4
m 4   0   0   0   2   0

Transfer functions
m0->m1: 2.000000/(2.000000-s)
m1->m2: 1.000000/(2.000000-s)
m1->m4: 1.000000/(2.000000-s)
m2->m1: 3.000000/(4.000000-s)
m2->m3: 1.000000/(4.000000-s)
m3->m0: 2.000000/(5.000000-s)
m3->m4: 3.000000/(5.000000-s)
m4->m3: 1.000000/(1.000000-s)

Total fired times of each transition
t1: 1
t2: 2
t3: 2
t4: 2
t5: 1
```

/* **INPUT FILE FOR EXAMPLE FROM MANUFACTURING SYSTEMS**
(refer Section 5.2) */

```
10 9              /* Maximum places and Transitions */
1 0 0 0 0 0 0 0 0/* INPUT ARCS MATRIX */
0 1 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0
0 0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0 1
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1 0
1 0 0 0 0 1 0 0 0
0 1 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0 1  /* OPUTPUT ARCS MATRIX  */
1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1
0 0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 1 1
0 0 0 1 0 0 0 1 1
0 0 0 0 0 0 0 0 0   /* INHIBITOR ARCS MATRIX  */
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
1 /* INITIAL MARKING  */
0
0
0
1
0
0
0
```

```
1
1
1
1        /* Transitions Parameters */
0.5
1
1
0.8
2
2
3
```

/*OTUPUT FILE FOR MANUFACTURING SYSTEM EXAMPLE OF SECTION 5.2*/

```
Data read from input file:
Input Arcs :
    1    0    0    0    0    0    0    0    0
    0    1    0    0    0    0    0    0    1
    0    0    1    0    0    0    0    0    0
    0    0    0    1    0    0    0    0    0
    0    0    0    0    1    0    0    0    0
    0    0    0    0    0    1    0    0    1
    0    0    0    0    0    0    1    0    0
    0    0    0    0    0    0    0    1    0
    1    0    0    0    0    1    0    0    0
    0    1    0    0    1    0    0    0    0
Output Arcs :
    0    0    0    1    0    0    0    0    1
    1    0    0    0    0    0    0    0    0
    0    1    0    0    0    0    0    0    0
    0    0    1    0    0    0    0    0    0
    0    0    0    0    0    0    0    1    1
    0    0    0    0    1    0    0    0    0
    0    0    0    0    0    1    0    0    0
    0    0    0    0    0    0    1    0    0
    0    0    0    1    0    0    0    1    1
    0    0    0    1    0    0    0    1    1
Inhibitor Arcs :
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0
Initial markings
          1
          0
          0
          0
          1
          0
          0
```

```
       0
       1
       1


lamda
1.000000
1.000000
0.500000
1.000000
1.000000
0.800000
2.000000
2.000000
3.000000

A loop found.
A loop found.
A loop found.
A loop found.

Firing Sequences
Fire/enabled/l=0  1  2  3  4  5  6  7  8  9  0
t0/ 1 5                 1  0  0  0  1  0  0  0  1  1 New m0
t1/ 2 5                 0  1  0  0  1  0  0  0  0  1 Conflict New m1
t2/ 3                        0  0  1  0  1  0  0  0  0  0 New m2
t3/ 4                          0  0  0  1  1  0  0  0  0  0 New m3
t4/                               1  0  0  0  1  0  0  0  1  1 m0
t5/ 9                        0  1  0  0  0  1  0  0  0  0 New m4
t9/                             1  0  0  0  1  0  0  0  1  1 m0
t5/ 1 6                 1  0  0  0  0  1  0  0  1  0 Conflict New m5
t1/                          0  1  0  0  0  1  0  0  0  0 m4
t6/ 7                        1  0  0  0  0  0  1  0  0  0 New m6
t7/ 8                          1  0  0  0  0  0  0  1  0  0 New m7
t8/                               1  0  0  0  1  0  0  0  1  1 m0


Reachability graph
                                   1  1  1  1  1  1  1  1  1  1  2
    m0 m1 m2 m3 m4 m5 m6 m7 m8 m9 m0 m1 m2 m3 m4 m5 m6 m7 m8 m9 m0
m 0  0  1  0  0  0  5  0  0
m 1  0  0  2  0  5  0  0  0
m 2  0  0  0  3  0  0  0  0
m 3  4  0  0  0  0  0  0  0
m 4  9  0  0  0  0  0  0  0
m 5  0  0  0  0  1  0  6  0
```

```
m 6   0   0   0   0   0   0   0   7
m 7   8   0   0   0   0   0   0   0
```

```
Transfer functions
m0->m1: 1.000000/(2.000000-s)
m0->m5: 1.000000/(2.000000-s)
m1->m2: 1.000000/(2.000000-s)
m1->m4: 1.000000/(2.000000-s)
m2->m3: 0.500000/(0.500000-s)
m3->m0: 1.000000/(1.000000-s)
m4->m0: 3.000000/(3.000000-s)
m5->m4: 1.000000/(1.800000-s)
m5->m6: 0.800000/(1.800000-s)
m6->m7: 2.000000/(2.000000-s)
m7->m0: 2.000000/(2.000000-s)
```

```
Total fired times of each transition
t1: 2
t2: 1
t3: 1
t4: 1
t5: 2
t6: 1
t7: 1
t8: 1
t9: 1
```