NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

ISBN 0 315 56337-0

Performance Prediction Using Timed Petri Nets

By
Gulammahammad Hasanasaheb Masapati

THESIS SUBMITTED
TO THE SCHOOL OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE

at the
UNIVERSITY OF OTTAWA
June 1987

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

Abstract

This thesis investigates system performance prediction using timed Petri nets. Two classes of Petri nets — D-timed and M-timed Petri nets are examined. The formalism of D-timed Petri net is extended to include non-singular nets. A new definition — *guarded due to inheritance place* — is introduced to assist in classifying Petri nets as conflict free nets. Analyses of D-timed and M-timed Petri nets are automated.

A set of reduction rules which permit the systematic reduction of states in a transition graph are presented. In particular, a more general algorithm for vertex folding than those presently existing is developed. It is based on the law of conservation of transition time. To the best of our knowledge, this is the first general vertex folding algorithm. We demonstrate the application of these reduction rules to a version of the stop-and-wait protocol. All these reduction rules are incorporated into a graph reduction software package which can be used in automated performance prediction.

With the help of two examples, we demonstrated the application of timed Petri nets to performance prediction. The first example deals with the analysis of a stop-and-wait protocol. It is modeled as a D-timed Petri net. The model of the stop-and-wait protocol developed in this thesis is quite general — losing messages, messages being damaged and losing acknowledgements are modeled explicitly. The optimum message length and performance of the stop-and-wait protocol are calculated. The second example deals with the analysis of an office copying system. It is modeled as an M-timed Petri net. Various performance measures such as average turnaround time and average waiting time in queue are computed.

Acknowledgements

I would like to express my gratitude to my supervisor, Dr.George M. White, for his advice, guidance and constant encouragement throughout the course of this research. He was kind and patient. It was a pleasure working with him.

I would also like to thank Professor T.I.Ören for his invaluable help during my stay in the Computer Science department.

I would like to sincerely thank Professor R.L.Probert for giving me an opportunity to study in the Computer Science department, Professor L.G.Birta for his generous financial assistance from October 1982 to September 1984, Dr.S.I.Omar for his advice, encouragement and constant support, and Dr.F.Hadziomerovic for his kind words about this research work.

Finally, I am grateful to my colleagues Murat Özmizrak and Nur Özmizrak for their encouragement, help and support.

Contents

Chapter 1

Introduction

1.1 Motivation

Predicting the performance measures such as turnaround time or throughput of computer systems, communication protocols and office information systems, etc., has been actively pursued for many years now. Tools have usually been derived from queueing theory results[REI 82]. New tools such as timed Petri nets[ZUB 80] [ZUB 85a] [ZUB 85b] [ZUB 86a] [ZUB 86b] and finite state machines augmented with time and relative probability for each state transitions[RUD 83] [RUD 84] are also beginning to be used.

In computer-communication systems, formal protocol specification is a fast growing area of research. The formal specification of protocols makes use of modeling techniques such as finite-state machines, Petri nets, and high-level programming languages, although Petri nets are not presently very popular with standards organizations. This has necessiated the automated performance prediction directly from the formal protocol definition[RUD 83]. Using timed Petri net as the formal specification technique, the performance analysis of communication protocols has been investigated by Razouk[RAZ 84] and Zuberek[ZUB 85a] [ZUB 86a] [ZUB 86b].

The objective of the thesis hence is to assist in this goal of automating performance prediction using timed Petri nets.

1.2 Contributions

This thesis deals with system performance analysis using timed Petri nets. The contributions of this research are divided into two categories — *theoretical* and *applied*.

Theoretical

1. Development of an algorithm for vertex folding (Chapter 3).
2. Development of the formalism for non-singular D-timed Petri nets (Chapter 2).
3. Assisting in classifying Petri nets as conflict free nets. We introduce a new definition — *guarded due to inheritance place* (Chapter 2).

Applied

1. Analyses of D-timed and M-timed Petri nets are automated. The set of reachable states of D-timed and M-timed Petri nets shown in Tables 4.2, 4.3 and 4.4 (Chapter 4) are generated by our computer programs.
2. All graph reduction rules (Chapter 3) are incorporated into a graph reduction software package which is used in automated performance prediction.
3. We calculate the optimum message length and performance of the stop-and-wait protocol. The model of the stop-and-wait protocol developed in this thesis is more general than one employed in an earlier study[ZUB 85a]. Losing messages, messages being damaged and losing acknowledgements are modeled explicitly (Chapter 4).
4. We also illustrate (Chapter 4) the use of M-timed Petri nets in analysing the office copying system extracted from office information systems discipline; usually queueing theory is used to analyse such systems.

1.3 Structure of the thesis

The remainder of the thesis is structured as follows. In Chapter 2, we introduce basic concepts and terminology of Petri nets and timed Petri nets. Two classes of timed Petri nets — D-timed and M-timed are examined in detail. We in essence establish a frame work suitable for automated system performance prediction.

In Chapter 3, we present graph reduction rules — vertex reduction, junction removal, decision removal, multi-in multi-out vertex removal, merging parallel self-loops, merging parallel edges, and self-loop removal. These rules help reduce the size of probabilistic transition graphs obtained from analysis of timed Petri nets. These rules can also be used to calculate the running time of computer programs[BEI 70] and to automatically reduce the global-state graphs obtained from reachability analysis of communicating finite-state machines[RUD 83].

We also present the complete algorithm for vertex folding. To the best of our knowledge, this is the first general vertex folding algorithm. All these rules are incorporated into a graph reduction software package which can be used in automated performance prediction. We demonstrate the application of these reduction rules to a version of the stop-and-wait protocol (Chapter 4).

Based on the formalisms developed in Chapters 2 and 3, we illustrate the application of timed Petri nets to performance prediction in Chapter 4. We consider two examples. The first example deals with the analysis of a stop-and-wait protocol. It is modeled as a D-timed Petri net. We calculate the optimum message length and performance of the stop-and-wait protocol.

The second example is extracted from the office information systems discipline. It deals with the analysis of an office copying system. It is modeled as an M-timed Petri net. We calculate various performance measures such as average turnaround time and average waiting time in queue etc.

A brief summary is given in Chapter 5 and is followed by References.

Chapter 2

Timed Petri Nets

Since its creation in 1962 by **Carl Adam Petri**, the Petri net is well known as an abstract, formal model of systems with interacting, concurrent and asynchronous components[PET 77] [AGE 79] [PET 81]. The spectrum of use of Petri nets in system analysis is very diverse and wide — from analysis of legal systems to performance evaluation of computer systems and communication protocols[PET 81] [ZUB 85b] [ZUB 86a]. In this chapter, we review basic concepts of Petri nets and establish a frame work suitable for system performance analysis.

In sections 2.1–2.3, we present basic Petri nets, modified Petri nets and related concepts. We also introduce for the first time a new definition — *guarded due-to inheritance place* — which assists in classifying Petri nets as conflict free nets. In section 2.4, we present the concepts related to timed Petri nets. In subsection 2.4.1, we introduce D-timed Petri nets. We extend the formalism of D-timed Petri nets to include non-singular nets whereas D-timed Petri nets presented by Zuberek[ZUB 86a] are valid only for singular nets. In subsection 2.4.2, we present M-timed Petri nets. We summarize the results of this chapter in section 2.5.

The notation and terminology used in this chapter closely follows that of Peterson[PET 81] and Zuberek[ZUB 85b] [ZUB 86a]. In this chapter, we occasionally refer to Petri nets as simply nets.

To facilitate the presentation, we list below the notations used in this chapter.

TABLE OF NOTATIONS:

$a(T_j, e)$	describes the number of different ways the choices can be made within T_j [ZUB 85b]
$a_{ij}(t)$	non-negative integer variable
c	choice function $c : T \rightarrow [0,1]$ s.t. $\sum_{t \in O(p)} c(t) = 1$ for each free-choice place p and $c(t) = 1$ for all other transitions
D	set of directed edges in $\mathbf{G(DT)}$ or $\mathbf{G(MT)}$ or $\mathbf{G}_t$
$\mathbf{DT}$	D-timed Petri net
D_a	set of directed edges in $\mathbf{G}_a$
e, e_i	enable or selection functions
f	firing time function, $f : T \rightarrow R^+$
f_i	selection function
$Free(T)$	$\{T_i : T_i = O(p_i) \wedge p_i$ is free-choice$\}$
$\mathbf{G(X)}$	state graph of D or M-timed net $\mathbf{X}$
$\mathbf{G}_a, \mathbf{G}_t$	transition graphs
h	sojourn time function, $h : V \rightarrow R^+$ s.t. $h(v_i) = h_i$ and h_i for D-timed and M-timed nets is given on pages 24 and 26 respectively
$I(O)$	input(output) function
$I(t)(O(t))$	set of all input(output) places of a transition t
$I(p)(O(p))$	set of all input(output) transitions of a place p
$Inh(t)$	set of all inhibitor places of t
$\mathbf{MT}$	M-timed Petri net
N	set of non-negative integers
P	set of places in $\mathbf{PN}$
$\mathbf{PN}$	Petri net
P^∞	bags of places
p_g, p_k, p_{mn}	elements of P or $I(t)$
$p_1, \ldots, p_n$	elements of P or $I(t)$
p, p_a	set of probabilities associated with each directed edge of $\mathbf{G(X)}$, $\mathbf{G}_a$
$\mathbf{R(PN)}$	set of reachable markings of $\mathbf{PN}$ from μ_0
R^+	set of non-negative real numbers
$\mathbf{S(X)}$	set of reachable states of net $\mathbf{X}$ from s_i
$Sel(\mu)$	set of all selection functions of μ
s_i	initial state of D or M-timed net
s, s_j	states of D or M-timed net
$s_j \overset{e_k}{\rightarrow} s_i$	s_j is directly e_k-reachable from s_i

$s_j \xleftarrow{(t_k,e_l)} s_i$	s_j is directly (t_k, e_l)-reachable from s_i
T	set of all transitions in **PN**
$T(\mu)$	set of all transitions enabled by μ
$t, t_i, t_j, t_1, \ldots, t_m$	elements of T or $I(p)$
$t_{i_1}, \ldots, t_{i_n}$	elements of T or $I(p)$
V	set of vertices in $\mathbf{G(X)}$ or $\mathbf{G}_t$
V_a	set of vertices in $\mathbf{G}_a$
μ, μ_i, μ_j	marking functions
$\mu_{i_0}, \ldots, \mu_{i_k}$	marking functions
μ_0	initial marking
$\mu_j \xleftarrow{t} \mu_i$	marking μ_j is directly reachable from μ_i
$\mu_j \xleftarrow{d} \mu_i$	same as $\mu_j \xleftarrow{t} \mu_i$
$\mu_j \leftarrow \mu_i$	μ_j is reachable from μ_i
ν, ν_i, ν_j	firing rank function
γ	denotes remaining-firing-time function for D-timed net and for M-timed net it is firing rate function
γ_i, γ_j	denote remaining-firing-time functions for D-timed net
$\#X$	number of elements in set or bag X
$\#(x, X)$	number of occurances of x in X
$\lfloor X \rfloor$	largest integer $\leq X$
f(t)	false(true)
$\wedge(\vee)$	and(or)
$\sum$	summation
$\cap$	intersection
$\prod$	product function
$\emptyset$	empty set
$\in$	belongs to
$\exists$	there exists
$\forall$	for all
$\notin$	does not belong to
$\nexists$	there does not exist
<=> or iff	if and only if
==>	follows from earlier step or steps
s.t.	such that

2.1 Petri Nets

Definition 2.1
A *Petri net* PN is a five-tuple, $\mathrm{PN} = (P, T, I, O, \mu_0)$

- $P = \{p_1, p_2, \ldots, p_n\}$ is a finite set of *places*, $n \geq 0$.
- $T = \{t_1, t_2, \ldots, t_m\}$ is a finite set of *transitions*, $m \geq 0$. The set of places and the set of transitions are disjoint, $P \cap T = \emptyset$.
- $I : T \longrightarrow P^\infty$ is the *input* function, a mapping from transitions to bags of places. A *bag* is a generalization of a set that allows multiple occurances of an element.
- $O : T \longrightarrow P^\infty$ is the *output* function, a mapping from transitions to bags of places.
- $\mu_0 : P \longrightarrow N$ is the *initial marking* for the net where N is the set of non-negative integers.

A graph is often used to illustrate Petri nets where a *circle* represents a place and *bar* represents a transition. The dynamic aspects of Petri net models are denoted by markings which are assignments of *tokens* to the places of Petri net. A *black dot* inside the places of a Petri net represents a token.

Fig. 2.1 shows a Petri net graph corresponding to the Petri net in Fig. 2.2. An arrow from(to) a place to(from) a transition defines the place to be an input(output) to the transition. The set of all input(output) places of a transition, $t \in T$, defines the $I(t)(O(t))$. Similarly, $I(p)(O(p))$ denotes the set of all input(output) transitions of a place, $p \in P$.

Definition 2.2
Marking of a Petri net is a function $\mu : P \longrightarrow N$.
The marking represents the state of the Petri net.

Example 2.1
Referring to Fig.2.1, we find that the marking of the Petri net, μ is equal to

$$\{1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0\}$$

Definition 2.3
A transition is *enabled* $\Longleftrightarrow$ each of its input places contains at least as many tokens as there exists arcs from that place to the transition.

The set of all enabled transitions for a given marking μ of a Petri net is denoted by $T(\mu)$.

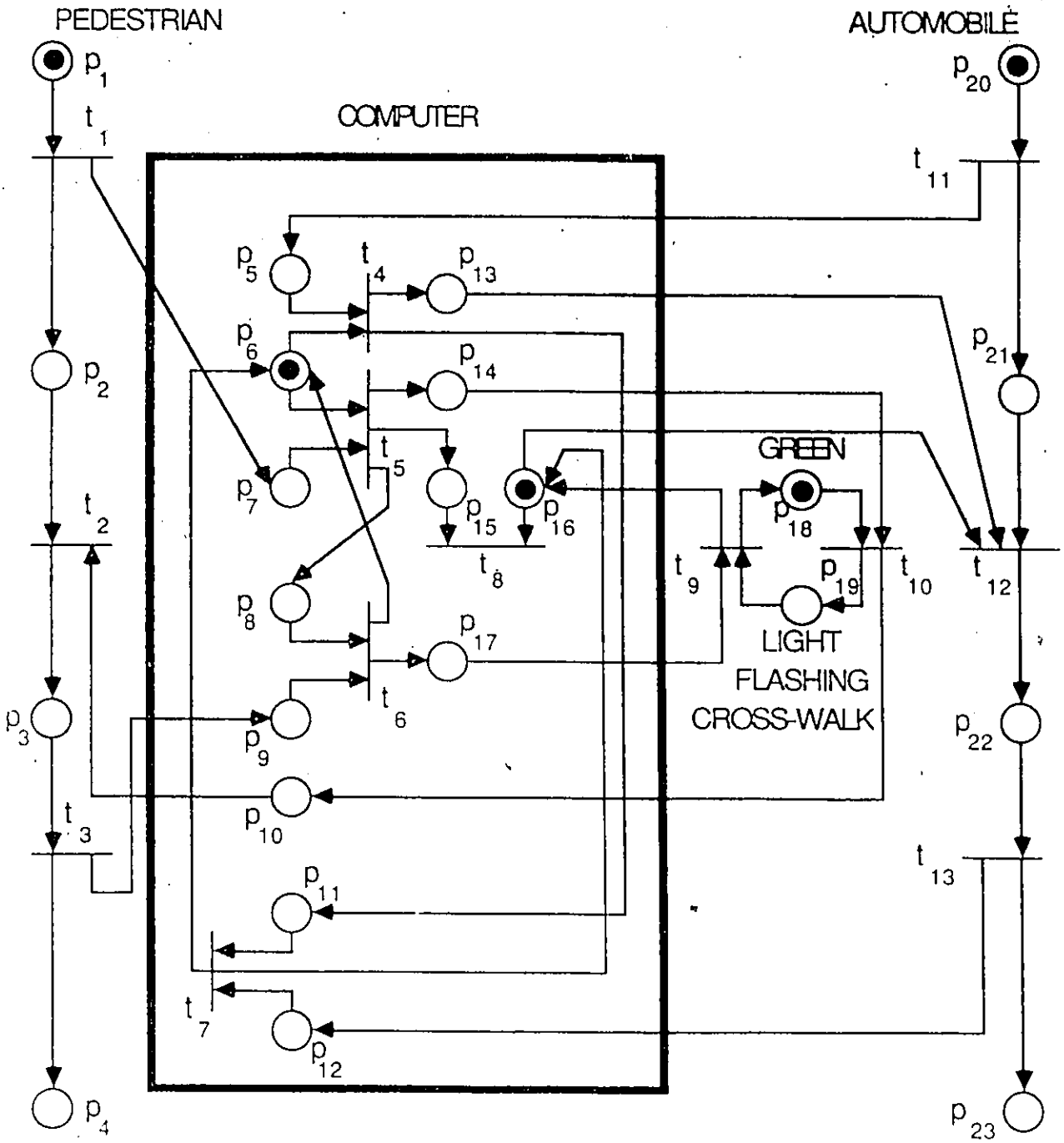

PEDESTRIAN
AUTOMOBILE
COMPUTER
GREEN
LIGHT
FLASHING
CROSS-WALK
p1
p2
p3
p4
p5
p6
p7
p8
p9
p10
p11
p12
p13
p14
p15
p16
p17
p18
p19
p20
p21
p22
p23
t1
t2
t3
t4
t5
t6
t7
t8
t9
t10
t11
t12
t13

Fig.2.1

$$P = \left\{ \begin{array}{l} p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_{12}, p_{13} \\ p_{14}, p_{15}, p_{16}, p_{17}, p_{18}, p_{19}, p_{20}, p_{21}, p_{22}, p_{23} \end{array} \right\}$$

$$T = \{ t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}, t_{13} \}$$

$$\begin{array}{ll} I(t_1) = \{p_1\} & O(t_1) = \{p_2, p_7\} \\ I(t_2) = \{p_2, p_{10}\} & O(t_2) = \{p_3\} \\ I(t_3) = \{p_3\} & O(t_3) = \{p_4, p_9\} \\ I(t_4) = \{p_5, p_6\} & O(t_4) = \{p_{11}, p_{13}\} \\ I(t_5) = \{p_6, p_7\} & O(t_5) = \{p_8, p_{14}, p_{15}\} \\ I(t_6) = \{p_8, p_9\} & O(t_6) = \{p_6, p_{17}\} \\ I(t_7) = \{p_{11}, p_{12}\} & O(t_7) = \{p_6, p_{16}\} \\ I(t_8) = \{p_{15}, p_{16}\} & O(t_8) = \emptyset \\ I(t_9) = \{p_{17}, p_{19}\} & O(t_9) = \{p_{16}, p_{18}\} \\ I(t_{10}) = \{p_{14}, p_{18}\} & O(t_{10}) = \{p_{10}, p_{19}\} \\ I(t_{11}) = \{p_{20}\} & O(t_{11}) = \{p_5, p_{21}\} \\ I(t_{12}) = \{p_{13}, p_{16}, p_{21}\} & O(t_{12}) = \{p_{22}\} \\ I(t_{13}) = \{p_{22}\} & O(t_{13}) = \{p_{12}, p_{23}\} \end{array}$$

$$\mu_0 = \{1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0\}$$

Fig.2.2

Example 2.2

Referring to Fig.2.1, we find that transitions t_1 and t_{11} are enabled.

Definition 2.4

A transition is *firable* if it is enabled.

When an enabled transition $t \in T$ fires:

- $\forall p \in I(t)$, $\#(p, I(t))$ tokens are removed from p
- $\forall p \in O(t)$, $\#(p, O(t))$ tokens are deposited into p

This results in a new marking. Transition firings continue as long as there exists at least one enabled transition.

Example 2.3

Fig. 2.3 is the result of firing an enabled transition t_1 in Fig.2.1.

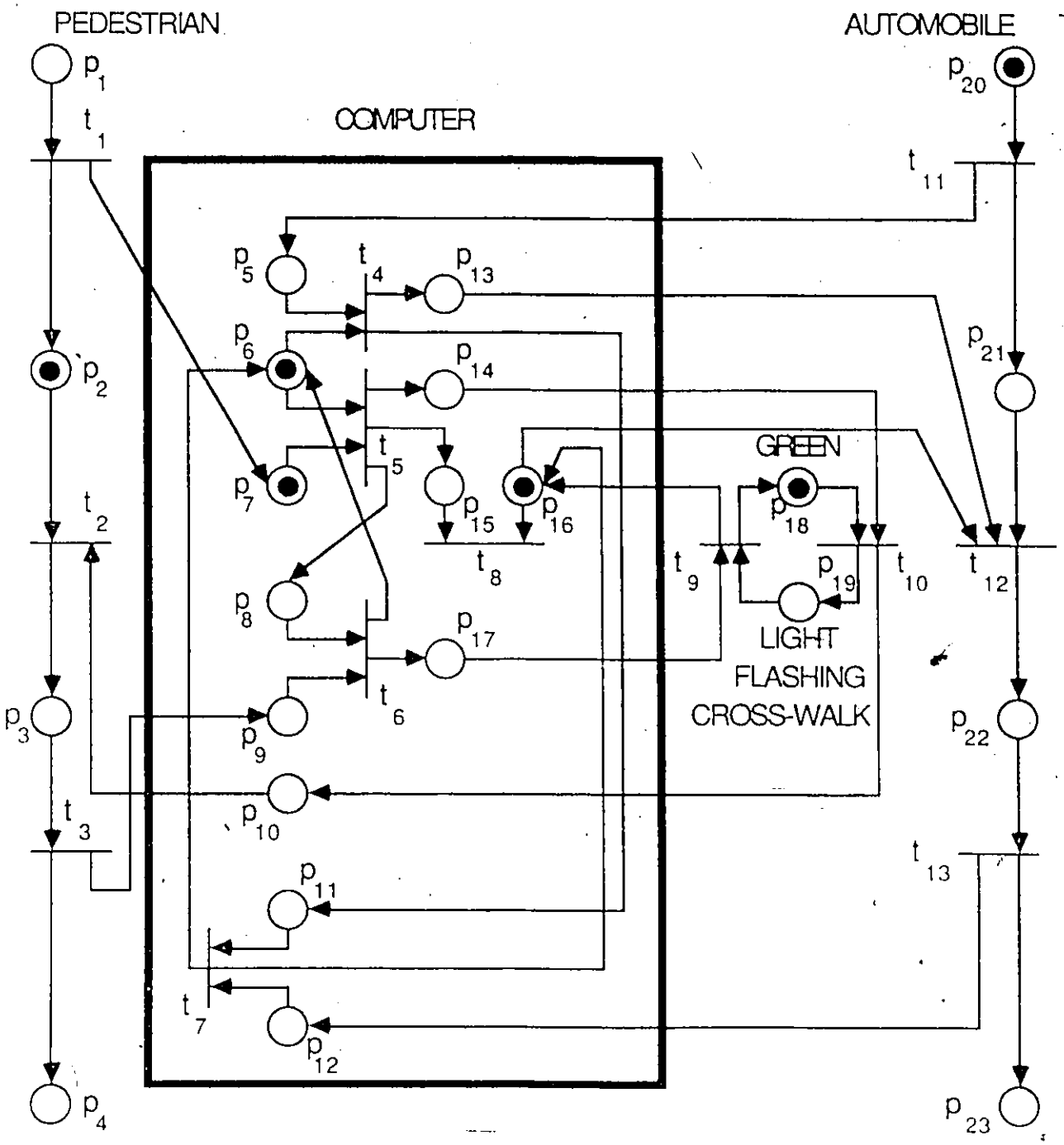

PEDESTRIAN
AUTOMOBILE
COMPUTER
p1
t1
p2
t2
p3
t3
p4
p20
t11
p21
p5
t4
p13
p6
p14
t5
p7
p15
p16
t8
p8
p17
t6
p9
p10
p11
t7
p12
GREEN
p18
t9
p19
t10
t12
LIGHT
FLASHING
CROSS-WALK
p22
t13
p23

Fig.2.3

Interpretation of Petri net graph(Fig. 2.1)

In Petri net models, places represent conditions and transitions represent events. Fig. 2.1 can be interpreted as a model of a simple cross-walk signalling system. Four components of the system — the pedestrian, the computer or the controlling device, the cross-walk light and the automobile — are modeled.

Places p_1, p_2, p_3 and p_4 represent the different conditions that can hold for the pedestrian. The conditions respectively are: approaching, ready-to-cross, crossing and past the crossing. Similar conditions can also hold for the automobile and are represented by p_{20}, p_{21}, p_{22} and p_{23} respectively. The states of the cross-walk light are represented by two places — p_{18}(the light is green) and p_{19}(the light is flashing). Place p_{10} represents the indicator for the computer that the cross-walk light is green. Place p_6 represents the condition that the computer is ready to receive input signals — pedestrian pressed the button and sensors detected the automobile approaching.

Transitions t_1,t_2 and t_3 denote events signalling — the pedestrian arrived, the pedestrian entering the cross-walk and the pedestrain's departure. Similarly, t_{11},t_{12} and t_{13} represent the events signalling — the automobile approaching, the automobile is entering the cross-walk and the automobile's departure.

Places and transitions inside the large box represent states and responses of the computer making the cross-walk system safe in the sense of hazards involved in such real-time systems. Leveson and Stolzy[LEV 87] have analysed such properties as safety and fault-tolerance of safety-critical real-time systems using Petri nets.

Definition 2.5

A marking μ_j is *directly reachable* from marking μ_i in a Petri net **PN**, $\mu_j \xleftarrow{t} \mu_i$, $<-> $ a transition $t \in T(\mu)$ such that

$$\mu_j(p) = \mu_i(p) - \#(p, I(t)) + \#(p, O(t)) \quad \forall p \in P$$

Example 2.4

Let μ_i and μ_j denote the markings of Petri net graphs shown in Fig.2.1 and Fig.2.3 respectively. Then, we have,

$$\mu_i = \{1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0\}$$

$$\mu_j = \{0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0\}$$

$$T(\mu_i) = \{t_1, t_{11}\}$$

We now show that $\mu_j \xleftarrow{t_1} \mu_i$.

Given: $I(t_1) = \{p_1\}, O(t_1) = \{p_2, p_7\}$.

Hence,

$$\#(p, I(t_1)) \quad \begin{cases} 1 & \text{if } p = p_1 \\ 0 & \text{otherwise} \end{cases}$$

$$\#(p, O(t_1)) = \begin{cases} 1 & \text{if } p = p_2 \vee p = p_7 \\ 0 & \text{otherwise} \end{cases}$$

Case 1. $p = p_1$

$$\begin{aligned} & \mu_j(p_1) = 0 \\ & \mu_i(p_1) - \#(p_1, I(t_1)) + \#(p_1, O(t_1)) = 1 - 1 - 0 = 0 \\ \Rightarrow \quad & \mu_j(p_1) = \mu_i(p_1) - \#(p_1, I(t_1)) + \#(p_1, O(t_1)) \end{aligned}$$

Case 2. $p = p_2$

$$\begin{aligned} & \mu_j(p_2) = 1 \\ & \mu_i(p_2) - \#(p_2, I(t_1)) + \#(p_2, O(t_1)) = 0 - 0 + 1 = 1 \\ \Rightarrow \quad & \mu_j(p_2) = \mu_i(p_2) - \#(p_2, I(t_1)) + \#(p_2, O(t_1)) \end{aligned}$$

Case 3. $p = p_7$

$$\begin{aligned} & \mu_j(p_7) = 1 \\ & \mu_i(p_7) - \#(p_7, I(t_1)) + \#(p_7, O(t_1)) = 0 - 0 + 1 = 1 \\ \Rightarrow \quad & \mu_j(p_7) = \mu_i(p_7) - \#(p_7, I(t_1)) + \#(p_7, O(t_1)) \end{aligned}$$

Case 4. $\forall p : p \notin \{p_1, p_2, p_7\}$

Since $\#(p, I(t_1)) = 0 \wedge \#(p, O(t_1)) = 0$, then we have to only show that $\mu_j(p) = \mu_i(p)$. Examining μ_i and μ_j this is indeed true.

Combining cases — 1,2,3 and 4, we have,

$$\mu_j(p) = \mu_i(p) - \#(p, I(t_1)) + \#(p, O(t_1)) \qquad \forall p \in P$$

Hence, by Definition 2.5, we have, $\mu_j \overset{t_1}{\leftarrow} \mu_i$.

Definition 2.6

A marking μ_j is *reachable* from a marking μ_i in a Petri net PN, $\mu_j \leftarrow \mu_i$, if

- $\exists$ a sequence of markings $\{\mu_{i_0}, \mu_{i_1}, \ldots, \mu_{i_k}\}$ such that $\mu_{i_0} = \mu_i \wedge \mu_{i_k} = \mu_j$ and
- $\mu_{i_n} \overset{d}{\leftarrow} \mu_{i_{n-1}} \qquad \forall n = 1, 2, \ldots, k$

Definition 2.7

The set of *reachable markings* $\mathbf{R(PN)}$ of a Petri net PN is the set of all markings which are reachable from the initial marking μ_0. That is,

$$\mathbf{R(PN)} = \{\mu : \mu \leftarrow \mu_0\}$$

2.2 Petri nets with Inhibitor arcs

Referring to Fig.2.1, we find that $T(\mu_0) = \{t_1, t_{11}\}$. Firing transitions t_1 and t_{11}, we obtain a Petri net graph as shown in Fig.2.4. Now, transitions t_4 and t_5 are both enabled. However, firing either one of them disables the other since $\mu(p_6) = 1$, $p_6 \in I(t_4)$ and $p_6 \in I(t_5)$. If we know that the event $t_4(t_5)$ has priority over the event $t_5(t_4)$ then we could model such priority by including the special type of arcs called *inhibitor arcs*[PET 81] in the Petri net graph. Inhibitor arcs are represented by arcs with a *small circle* rather than an arrowhead at the transition. Inhibitor arc (p_5, t_5) is shown in Fig.2.5.

Definition 2.8

A place $p \in P$ is an *inhibitor place* of a transition $t \in T$ if $\exists$ an inhibitor arc (p, t).

The set of all inhibitor places of a transition t is denoted by $Inh(t)$.

Example 2.5

Referring to Fig.2.5, we find that the place p_5 is an inhibitor place.

For Petri nets with inhibitor arcs, the Definition 2.3 for enabled transitions is extended as follows.

Definition 2.9

A transition $t \in T$ is *enabled* $\Longleftrightarrow$

- $\mu(p) \geq \#(p, I(t)) \qquad \forall p \in I(t)$
- $\mu(p) = 0 \qquad \forall p \in Inh(t)$

Example 2.6

Referring to Fig.2.5, we find that the transition t_5 will be enabled if

- $\mu(p_6) > 0, \mu(p_7) > 0$
- $\mu(p_5) = 0$

Observe that whenever $\mu(p_5) > 0$, $\mu(p_6) > 0$ and $\mu(p_7) > 0$, transition t_4 is selected to fire — thus modeling the priority of the event t_4 over t_5. That is, if both the events — the pedestrian has pressed the cross-walk button and the sensors have detected the approaching automobile — are completed and both the pedestrain and the automobile are ready-to-cross, then the automobile is the first to go past the crossing. The firing rule for Petri nets with inhibitor arcs is same as for Petri nets without inhibitor arcs except that no tokens are removed from inhibitor places.

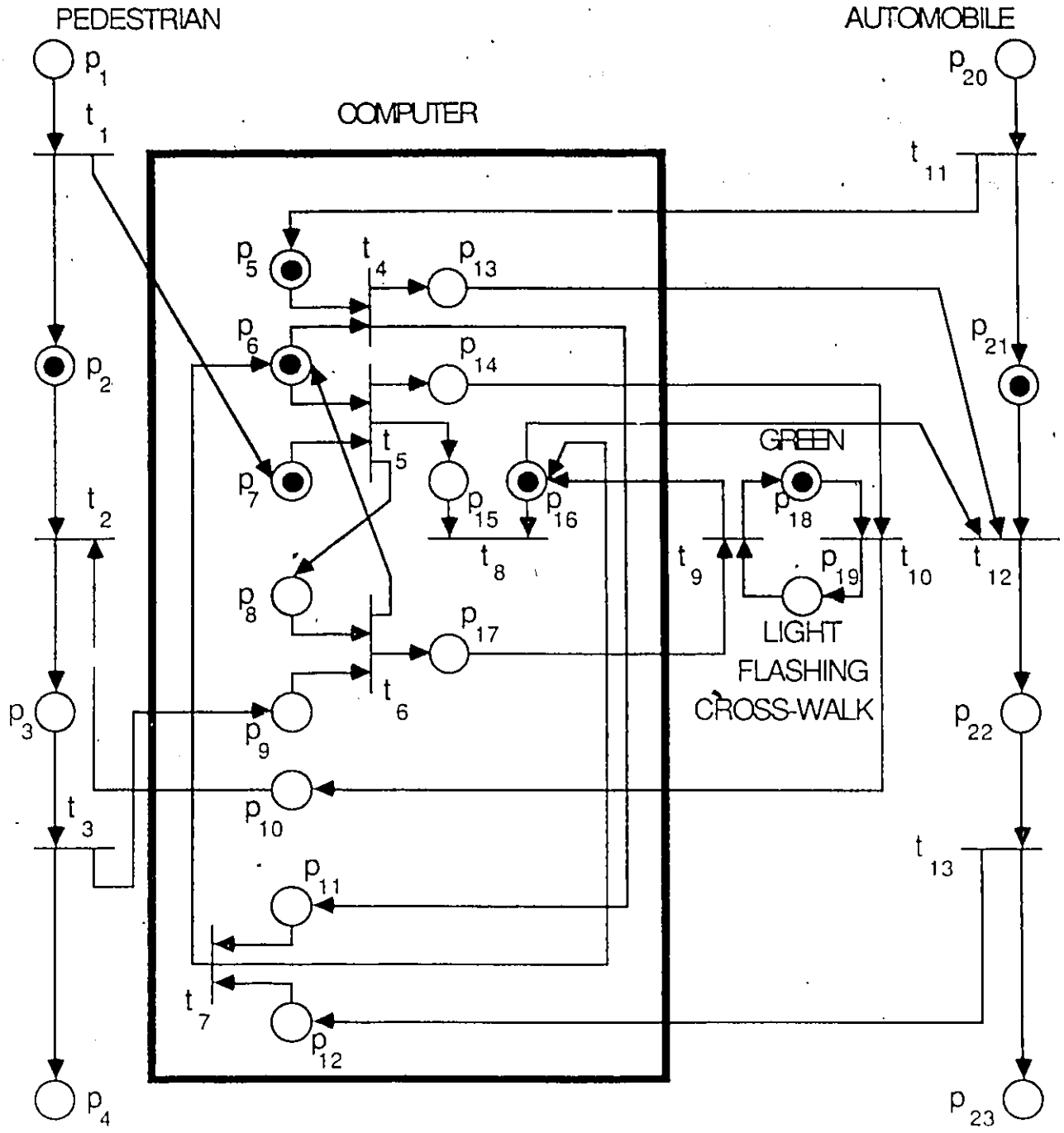

PEDESTRIAN
AUTOMOBILE
COMPUTER
p1
t1
p2
t2
p3
t3
p4
p5
t4
p13
p6
p14
t5
p7
p15
p16
t8
p8
p17
t6
p9
p10
p11
t7
p12
p20
t11
p21
GREEN
p18
t9
p19
t10
t12
LIGHT
FLASHING
CROSS-WALK
p22
t13
p23

Fig.2.4

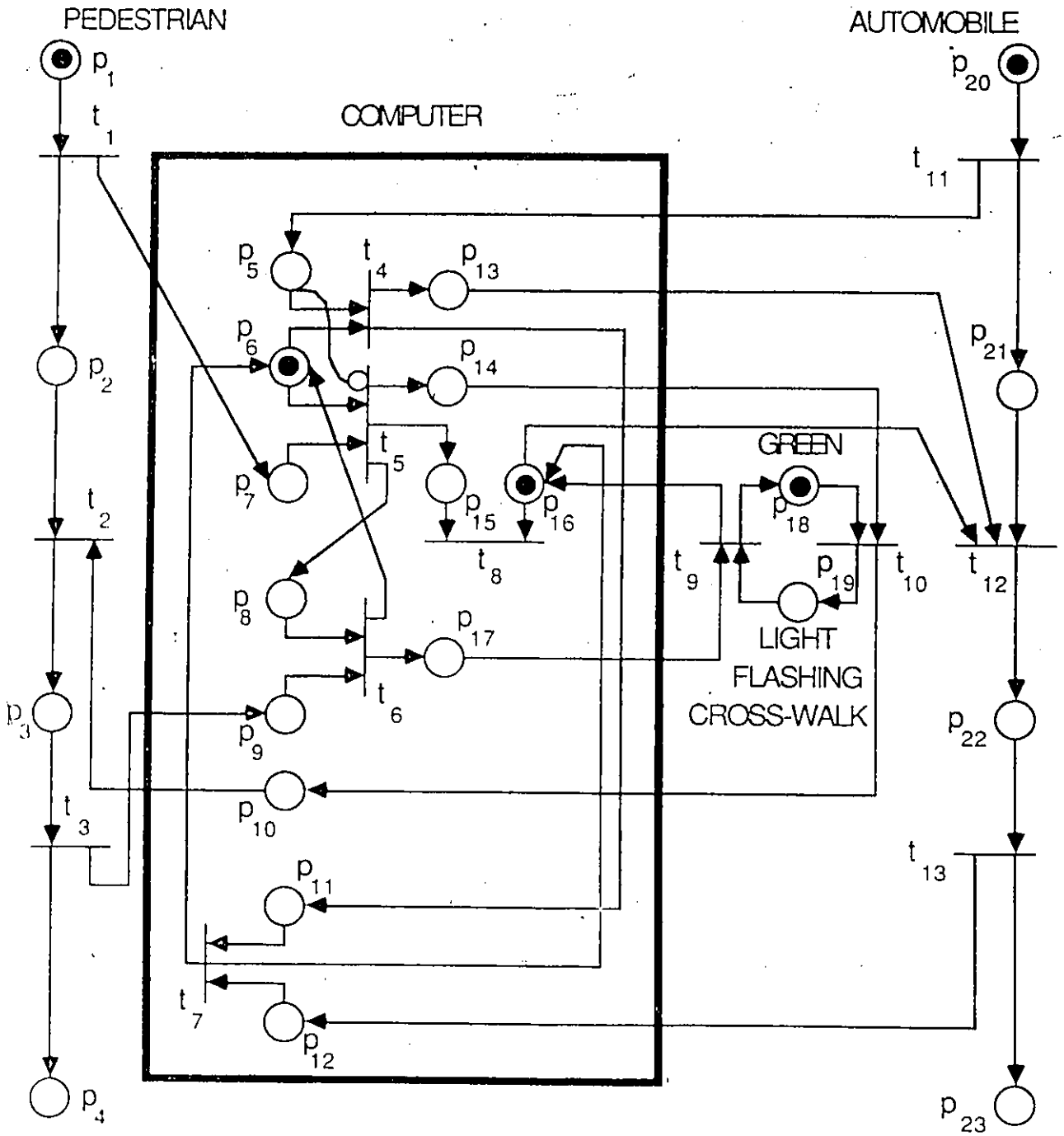

PEDESTRIAN
AUTOMOBILE
COMPUTER
GREEN
LIGHT
FLASHING
CROSS-WALK
p1
p2
p3
p4
p5
p6
p7
p8
p9
p10
p11
p12
p13
p14
p15
p16
p17
p18
p19
p20
p21
p22
p23
t1
t2
t3
t4
t5
t6
t7
t8
t9
t10
t11
t12
t13

Fig.2.5

2.3 Some More Definitions

Definition 2.10

A place $p \in P$ is *shared* if $\exists t_i, t_j \in T : p \in I(t_i) \wedge p \in I(t_j)$.

Example 2.7

Referring to Fig.2.1, we find that the place p_6 is shared by transitions t_4 and t_5. Also, p_{16} is shared by t_8 and t_{12}.

Definition 2.11

A Petri net PN is *conflict-free* if $\nexists p \in P : p$ is shared.

Example 2.8

Petri net graph shown in Fig.2.6 is conflict-free.

Definition 2.12

A shared place $p \in P$ is *free-choice* if

$$\forall [t_i, t_j \in O(p)] \quad I(t_i) = I(t_j) \wedge Inh(t_i) = Inh(t_j)$$

Example 2.9

Referring to Fig.2.7, we find that places p_5, p_6 and p_7 are free-choice places.

$$\begin{array}{lll} I(t_6) = I(t_8) = \{p_5\} & \wedge & Inh(t_6) = Inh(t_8) = \emptyset \\ I(t_9) = I(t_{10}) = \{p_7\} & \wedge & Inh(t_9) = Inh(t_{10}) = \emptyset \\ I(t_5) = I(t_7) = \{p_6\} & \wedge & Inh(t_5) = Inh(t_7) = \emptyset \end{array}$$

Observe that places p_3 and p_4 are not free-choice places.

$$\begin{array}{ll} I(t_3) = \{p_3\} & I(t_4) = \{p_3, p_4\} \\ Inh(t_3) = \{p_4\} & Inh(t_4) = \emptyset \end{array}$$

Definition 2.13

A shared place p is *guarded* if for each two (different) transitions t_i and t_j sharing p there is another place p_k such that

$$p_k \in I(t_i)(I(t_j)) \wedge p_k \in Inh(t_j)(Inh(t_i))$$

That is, no two transitions from $O(p)$ can be enabled by the same marking.

Example 2.10

Referring to Fig.2.7, we find that the place p_3 is guarded since $p_4 \in I(t_4) \wedge p_4 \in Inh(t_3) \wedge O(p_3) = \{t_3, t_4\}$.

Also referring to Fig.2.5, place p_6 is guarded since $p_5 \in I(t_4) \wedge p_5 \in Inh(t_5) \wedge O(p_6) = \{t_4, t_5\}$.

Definition 2.14

A shared place p is *guarded due to inheritance* if

$$\forall[t \in O(p)] \quad \exists p_k \neq p \;:\; p_k \in I(t) \wedge p_k \in O(O(p_g)) \wedge p_g \neq p_k \neq p \wedge p_g \text{ is guarded}$$

Example 2.11

Referring to Fig.2.5, we find that the place p_{16} is guarded due to inheritance.

$O(p_{16}) = \{t_8, t_{12}\}$
$I(t_8) = \{p_{15}, p_{16}\}$
$I(t_{12}) = \{p_{13}, p_{16}, p_{21}\}$

1. For $t = t_8$, choose $p_k = p_{15}$

$p_{15} \neq p_{16}$
$p_{15} \in I(t_8)$
$p_{15} \in O(t_5) \wedge t_5 \in O(p_6) \wedge p_6$ is guarded

2. For $t = t_{12}$, choose $p_k = p_{13}$

$p_{13} \neq p_{16}$
$p_{13} \in I(t_{12})$
$p_{13} \in O(t_4) \wedge t_4 \in O(p_6) \wedge p_6$ is guarded

Now, we can extend the definition of conflict-free Petri nets to include some more shared places as follows:

Definition 2.15

A Petri net **PN** is *conflict-free* if

either $\nexists p \in P : p$ is shared
or every shared place is either guarded or guarded due to inheritance

Example 2.12

Petri net graph shown in Fig.2.5 is conflict-free since shared places p_6 and p_{16} are guarded and guarded due to inheritance respectively.

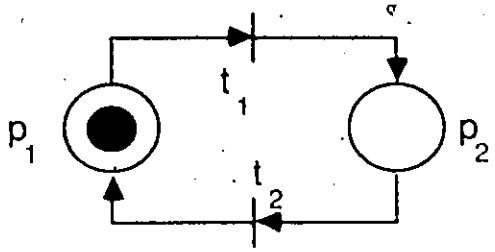


Fig.2.6

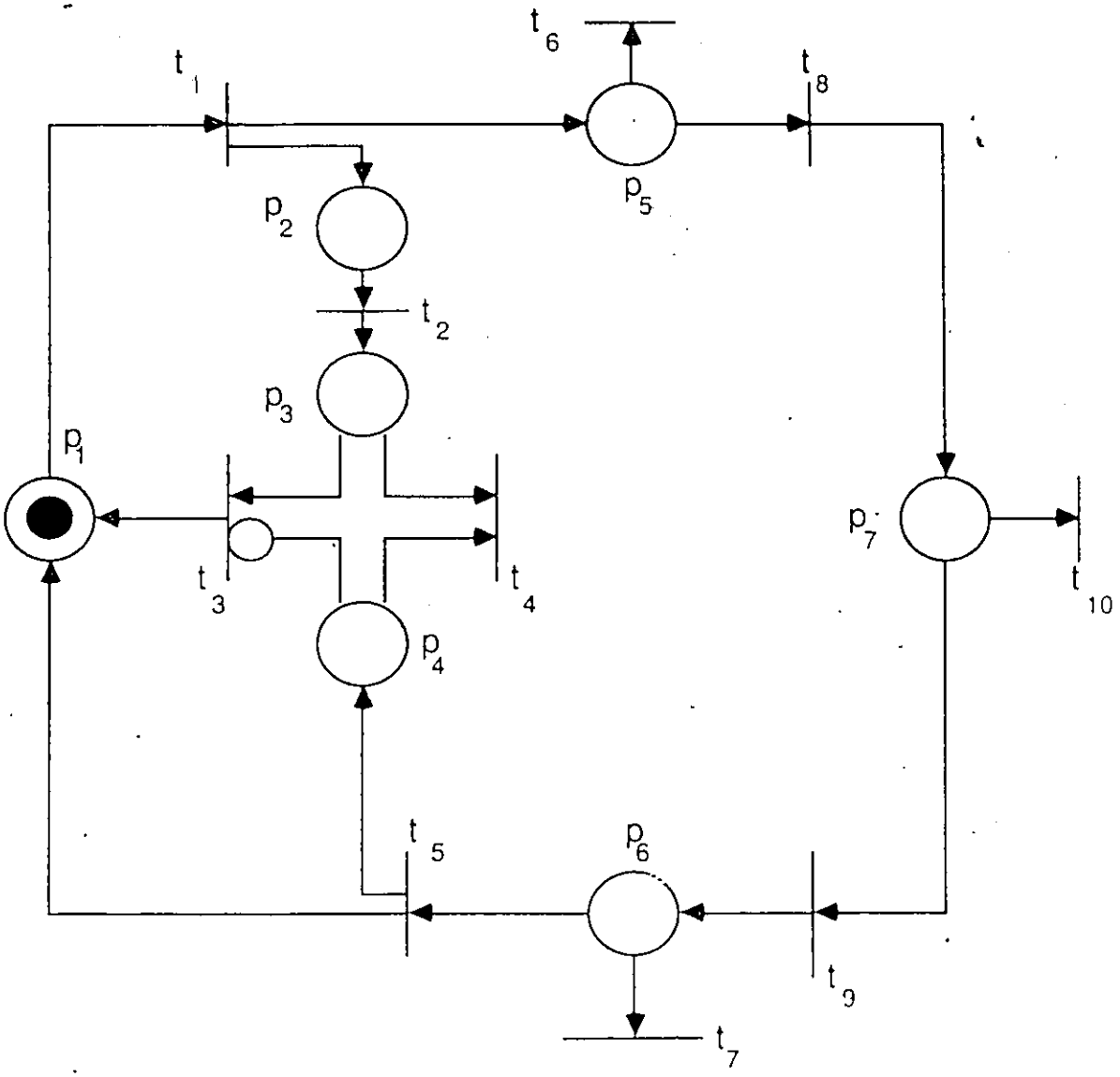


Fig.2.7

Definition 2.16

An *enable function* of a marking μ in a conflict-free Petri net, **PN**, without inhibitor arcs is a function $e : T \longrightarrow N$ defined as follows:

$\forall t \in T$
  Let $p_{min} = \{p_m \in I(t) : \mu(p_m) = \min_{p \in I(t)}[\mu(p)]\}$
  Let $p_{maxmin} \in p_{min} : \#(p_{maxmin}, I(t)) = \max_{p_m \in p_{min}}[\#(p_m, I(t))]$
  If $\mu(p_{maxmin}) - \#(p_{maxmin}, I(t)) < 0$
    Then $e(t) \leftarrow 0$
    Else $e(t) \leftarrow \lfloor \mu(p_{maxmin}) / \#(p_{maxmin}, I(t)) \rfloor$
  Endif
End $\forall t \in T$

Definition 2.17

An *enable function* of a marking μ in a conflict-free Petri net, **PN**, with inhibitor arcs is a function $e : T \longrightarrow N$ defined as follows:

$\forall t \in T$
  Case $(Inh(t) \neq \emptyset, \sum_{p \in Inh(t)} \mu(p) > 0)$ Of
    (t,t) : $e(t) \leftarrow 0$
    (t,f) : Case $(I(t) = \emptyset)$ Of
      t : $e(t) \leftarrow 1$
      f : Find $e(t)$ from Definition 2.16
    End {Case (t,f)}
    (f,*) : Find $e(t)$ from Definition 2.16
  End Case
End $\forall t \in T$

Note: * in (f,*) indicates "DO NOT CARE" condition. As a matter of fact, when $Inh(t) = \emptyset$, the question of computing $\sum_{p \in Inh(t)} \mu(p)$ does not arise.

Let $Free(T) = \{T_i : T_i$ is a set of output transitions of Free-choice place $p_i\}$. Then, referring to Example 2.9, we find that places p_5, p_6 and p_7 in Fig.2.7 are free-choice places. Let T_1, T_2 and T_3 represent the set of output transitions of free-choice places p_5, p_6 and p_7 respectively. Then, $Free(T)$ is given as follows:

$$Free(T) \quad \{T_1, T_2, T_3\}$$
$$\{O(p_5), O(p_6), O(p_7)\}$$
$$\{\{t_6, t_8\}, \{t_5, t_7\}, \{t_9, t_{10}\}\}$$

For each $T_i \in Free(T)$, either all transitions of T_i are enabled simultaneously, or none of them. Firing any one of the transitions in T_i will disable all other transitions in T_i. Hence, in free-choice Petri nets unlike in conflict-free Petri nets there may be several different enable functions for the same marking μ. We define next all such possibilities of transition firings and it is called the set of selection functions.

Definition 2.18

A *selection function* of a marking μ in a free-choice Petri net is a function $e : T \longrightarrow N$ such that

- $\exists$ a sequence of transitions $w = (t_{i_1}, t_{i_2}, \ldots, t_{i_k})$ in which $t_{i_j} \in T(\mu_{j-1})$ for $j = 1, 2, \ldots, k$ and for $\mu_{i_0} = \mu$, where

$$\forall p \in P \quad \mu_{i_j}(p) = \mu_{i_{j-1}}(p) - \#(p, I(t_{i_j}))$$

- $T(\mu_{i_k}) = \emptyset$
- $e(t) \leftarrow \#(t, w) \quad \forall t \in T$

The set of all selection functions of a marking μ is denoted by $Sel(\mu)$.

Definition 2.19

A Petri net **PN** is *singular* iff all selection or enable functions of all reachable markings indicate at most a single firing of a transition. That is,

$$\forall[\mu \in \mathrm{S(PN)}] \quad \forall[e \in Sel(\mu)] \quad \forall[t \in T]$$

$$\mathbf{PN} \quad \text{is} \quad \begin{cases} \text{singular} & \text{if } e(t) \leq 1 \\ \text{nonsingular} & \text{otherwise} \end{cases}$$

The formalism of D-timed Petri nets developed later(subsection 2.4.1) is extended to include non-singular nets whereas D-timed Petri nets presented by Zuberek[ZUB 86a] are valid only for singular nets.

Definition 2.20

A *choice-function* is a function $c : T_i \rightarrow [0, 1]$ s.t. $\sum_{t \in T_i} c(t) = 1$ for each $T_i \in Free(T)$.

The probability associated with a selection function $e \in Sel(\mu)$ is equal to

$$\prod_{T_j \in Free(T)} a(T_j, e) \prod_{t \in T_j} c(t)^{e(t)}$$

where the coefficient $a(T_j, e)$ describes the number of different ways in which choices can be made within T_j. The procedure for determining $a(T_j, e)$ is given by Zuberek[ZUB 85b].

2.4 Timed Petri nets

Petri nets as defined in sections 2.1–2.3 are not complete for the study of performance evaluation since no assumptions are made about the duration of events. There have been several different proposals for extending Petri nets to incorporate time. Ramchandani[RAM 74] introduced timed Petri nets by associating firing times with transitions of Petri nets, Sifakis[SIF 77] instead associated delays with places. Merlin and Farber[MER 76] introduced time Petri nets using two values — Min and Max times, to define a range of delays for each transition. Razouk[RAZ 84] associated firing times as well as enabling times with each transition of a Petri net. Molloy[MOLL 82] introduced stochastic Petri nets in which transition firing times are exponentially distributed random variables, and the corresponding firing rates are associated with transitions of Petri nets. Marsan, Conte and Balbo[MAR 84] further extended stochastic Petri nets to include two types of transitions — timed and immediate. Timed transitions have exponentially distributed firing times, whereas immediate transitions fire in zero time. Zuberek([ZUB 80],[ZUB 85b], [ZUB 86a]) has done a significant amount of work on performance evaluation of computer systems and communication protocols using timed Petri nets based on the extension of the proposal originated by Ramchandani[RAM 74]. Timed Petri nets discussed in this thesis are based on the approach suggested by Zuberek([ZUB 85b],[ZUB 86a]).

The firing times of transitions can be described in several ways. If firing times are non-negative real numbers then the Petri net is called *D-timed*([RAM 74],[ZUB 80],[ZUB 86a],[ZUB 86b]). If firing times are exponentially distributed random variables, and the corresponding firing rates are assigned to transitions then the Petri net is called *M-timed*[ZUB 85b] or *stochastic*[MAR 84].

2.4.1 D-timed Petri nets

Definition 2.21

A *D-timed Petri net*, **DT**, is a tuple, $(\mathbf{PN}, f)$, where,

- **PN** is a Petri net as defined in sections 2.1–2.3
- f is a *firing time function*, $f : T \rightarrow R^+$, where R^+ denotes the set of non-negative real numbers.

Example 2.14

In Fig.2.7, if we let

$$f(t_1) = 1 \quad f(t_2) = 15 \quad f(t_3) = 0 \quad f(t_4) = 0 \quad f(t_5) = 0$$
$$f(t_6) = 0 \quad f(t_7) = 0 \quad f(t_8) = 5 \quad f(t_9) = 2 \quad f(t_{10}) = 0$$

then the Petri net becomes a D-timed Petri net.

Definition 2.22

A *state* s of an D-timed Petri net, **DT**, is a 3-tuple, (μ, ν, γ), where,

- μ is a *marking function*, $\mu : P \rightarrow N$.
- ν is a *firing rank function*, $\nu : T \rightarrow N$. It indicates the number of active firings, i.e., the number of active firings that have been initiated but not yet terminated, for each transition.
- γ is a *remaining-firing-time-function* that assigns the remaining firing time to each independent firing (if any) of a transition. If $\nu(t) = k$ then $\gamma(t)$ is a vector of length k and $\gamma(t)[i]$ contains a non-negative real number representing the remaining firing time of the corresponding active firing i. γ is a partial function and is undefined for those $t \in T : \nu(t) = 0$.

Definition 2.23

An *initial state* s_i of an D-timed Petri net, **DT**, is a 3-tuple (μ_i, ν_i, γ_i) where

$\nu_i \leftarrow e_i \in Sel(\mu_0)$

$\mu_i(p) \leftarrow \mu_0(p) - \sum_{t \in O(p)} \{\#(p, I(t)) * \nu_i(t)\} \quad \forall p \in P$

$\forall t \in T$
  If $\nu_i(t) > 0$
    Then For $l \leftarrow 1, \nu_i(t)$ do
      $\gamma_i(t)[l] \leftarrow f(t)$
    End For
    Else $\gamma_i(t)$ is undefined
  Endif
End $\forall t \in T$

Note: A free-choice D-timed Petri net, DT, may have several different initial states.

Definition 2.24

A state $s_j = (\mu_j, \nu_j, \gamma_j)$ is *directly* e_k*-reachable* from the state $s_i = (\mu_i, \nu_i, \gamma_i)$, $s_j \overset{e_k}{\Leftarrow} s_i$, $\Leftrightarrow$

1. $h_i \leftarrow \min_{t \in T} \min_{1 \le l \le \nu_i(t)} \{\gamma_i(t)[l]\}$
2. $\forall t \in T$
   $a_{ij}(t) \leftarrow 0$
   If $\nu_i(t) > 0$ **Then**
   For $l \leftarrow 1, \nu_i(t)$ **do**
   If $\gamma_i(t)[l] = h_i$ **Then**
   $a_{ij}(t) \leftarrow a_{ij}(t) + 1$
   Endif
   End For
   Endif
   End $\forall t \in T$
3. $\mu_{ij}(p) \leftarrow \mu_i(p) + \sum_{t \in I(p)} \{\#(p, O(t)) * a_{ij}(t)\} \quad \forall p \in P$
4. $e_k \leftarrow Sel(\mu_{ij})$
5. $\mu_j(p) \leftarrow \mu_{ij}(p) \quad \sum_{t \in O(p)} \{\#(p, I(t)) * e_k(t)\} \quad \forall p \in P$
6. $\nu_j(t) \leftarrow \nu_i(t) - a_{ij}(t) + e_k(t) \quad \forall t \in T$
7. $\forall t \in T$
   If $a_{ij}(t) < 0$ **Then**
   For $l \leftarrow 1, \nu_i(t)$ **Do**
   $\gamma_j(t)[l] \leftarrow \gamma_i(t)[l] - h_i$
   End for
   Endif
   If $a_{ij}(t) > 0$ **Then**
   For $l \leftarrow 1, \nu_i(t) - a_{ij}(t)$ **Do**
   $\gamma_j(t)[l] \leftarrow \gamma_i(t)[l + a_{ij}(t)] - h_i$
   End for
   Endif
   If $e_k(t) > 0$ **Then**
   For $l \leftarrow 1, e_k(t)$ **Do**
   $\gamma_j(t)[\nu_i(t) - a_{ij}(t) + l] \leftarrow f(t)$
   End For
   Endif
   End $\forall t \in T$

The state s_j which is directly e_k-reachable from the state s_i is thus obtained by

- terminating those firings for which the remaining firing time is the smallest one; this time is denoted by h_i (steps 1 and 2)
- updating the marking of the net (step 3)
- initiating new firings (if any) which correspond to the selection function e_k from the set $Sel(\mu_{ij})$ (steps 4,5,6 and 7)

Definition 2.25

A state s_j is *reachable* from a state s_i, $s_j \leftarrow s_i$, if

- $\exists$ a sequence of states $\{s_{i_0}, s_{i_1}, \ldots, s_{i_k}\} : s_{i_0} = s_i \wedge s_{i_k} = s_j$
- $s_{i_n} \overset{e_k}{\leftarrow} s_{i_{n-1}} \qquad \forall n = 1, 2, \ldots, k$

Definition 2.26

A set of *reachable states* $\mathbf{S(DT)}$ of an D-timed Petri net $\mathbf{DT}$, is the set of all states of $\mathbf{DT}$ which are reachable from the initial states of $\mathbf{DT}$.

Definition 2.27

A *state graph* $\mathbf{G}$ of an D-timed Petri net $\mathbf{DT}$ is a labeled directed graph $\mathbf{G(DT)} = (V, D, p, h)$, where

- V is a set of vertices equal to $\mathbf{S(DT)}$
- D is a set of directed edges such that $(s_i, s_j) \in D \Leftrightarrow s_j \overset{e_k}{\leftarrow} s_i$
- p is a labeling function that assigns the probability of transition from s_i to s_j to each $(s_i, s_j) \in D$, $p : D \rightarrow [0, 1]$ such that if $s_j \overset{e_k}{\leftarrow} s_i$ then

$$p(s_i, s_j) = \prod_{T_j \in Free(T)} a(T_j, e_k) \prod_{t \in T_j} c(t)^{e_k(t)}$$

  where $c : T \rightarrow [0, 1]$ s.t. $\sum_{t \in O(p)} c(t) = 1$ for every free-choice place p and $c(t) = 1.0$ for all other transitions

- h is a sojourn time function that assigns the time spent in the state s_i (μ_i, ν_i, γ_i) to each state in the set V. That is, $h : V \rightarrow R^+$ such that

$$h(s_i) = h(s_i) = \min_{t \in T} \min_{1 \le l \le \nu_i(t)} \{\gamma_i(t)[l]\}$$

Definition 2.28

A *transition graph* $G_t = (V, D, p)$ is a directed graph where

- V is a set of vertices
- D is a set of directed edges; each directed edge is a 3-tuple (v_i, v_j, t_{ij}) where v_i is the source vertex, v_j is the destination vertex and t_{ij} is the transition time
- p is a set of probabilities associated with each directed edge in D

Definition 2.29

A transition graph $G_a = (V_a, D_a, p_a)$ is *adjoint* to a state graph $G = (V, D, h, p)$ $\Leftrightarrow$

$$V_a = V$$
$$D_a = \{(v_i, v_j, h(v_i)) : (v_i, v_j) \in D\}$$
$$p_a(v_i, v_j, t_{ij}) = p(v_i, v_j) \quad \forall (v_i, v_j, t_{ij}) \in D_a$$

Definition 2.29 can be used to obtain transition graph from state graph.

2.4.2 M-timed Petri nets

Definition 2.30

An *M-timed* Petri net **MT** is a tuple $(\mathbf{PN}, \gamma)$ where

- **PN** is a Petri net as defined in sections 2.1–2.3
- γ is a *firing rate function* $\gamma : T \rightarrow R^+$

Definition 2.31

A *state* s of an M-timed Petri net **MT** is a tuple (μ, ν) where

- μ is a *marking function* $\mu : P \rightarrow N$
- ν is a *firing rank function* $\nu : T \rightarrow N$ as defined before.

Definition 2.32

An *initial state* s_i of an M-timed Petri net **MT** is a tuple (μ_i, ν_i) where

- $\nu_i = f_i \in Sel(\mu_0)$
- $\mu_i(p) = \mu_0(p) - \sum_{t \in O(p)} \{\#(p, I(t)) * \nu_i(t)\} \quad \forall p \in P$

Note: A free-choice M-timed Petri net **MT** may have several different initial states.

Definition 2.33

A state $s_j = (\mu_j, \nu_j)$ is *directly* (t_k, e_l)*-reachable* from the state $s_i = (\mu_i, \nu_i) \Longleftrightarrow$

1. $\nu_i(t_k) > 0$
2. $\forall p \in P \quad \mu_{ij}(p) \leftarrow \mu_i(p) + \#(p, O(t_k))$
3. $e_l \in Sel(\mu_{ij})$
4. $\mu_j(p) \leftarrow \mu_{ij}(p) - \sum_{t \in O(p)} \{\#(p, I(t)) * e_l(t)\} \quad \forall p \in P$
5. $\forall t \in T \quad \nu_j(t) \leftarrow \nu_i(t) + e_l(t) - \begin{cases} 1 & \text{if } t = t_k \\ 0 & \text{otherwise} \end{cases}$

The state s_j which is directly (t_k, e_l) reachable from the state s_i is thus obtained by

- terminating the t_k firing (step 1)
- updating the marking of a net (step 2)
- initiating new firings (if any) which are determined by a selection function e_l from the set $Sel(\mu_{ij})$ (steps 3,4 and 5)

Definition 2.34

A *state graph* G of an M-timed Petri net **MT** is a labeled directed graph $\mathrm{G}(\mathbf{MT}) = (V, D, p, h)$ where

- V is a set of vertices equal to $\mathbf{S(MT)}$, the set reachable states of a net **MT**
- D is a set of directed edges such that $(s_i, s_j) \in D \Longleftrightarrow s_j \overset{(t_k, e_l)}{\longleftarrow} s_i$
- p is a labeling function which assigns the probability of transition from s_i to s_j to each edge $(s_i, s_j) \in D, p : D \rightarrow [0, 1]$ such that if $s_j \overset{(t_k, e_l)}{\longleftarrow} s_i$ then

$$p(s_i, s_j) = \gamma(t_k) * \nu_i(t_k) \prod_{T_j \in Free(T)} a(T_j, e_l) \prod_{t \in T_j} c(t)^{e_l(t)} / \sum_{t \in T} \gamma(t) * \nu_i(t)$$

- h is a sojourn time function that assigns the average time spent in the state $s_i = (\mu_i, \nu_i)$ to each state in the set V. That is, $h : V \rightarrow R^+$ such that

$$h(v_i) = h(s_i) = 1 / \sum_{t \in T} \gamma_i(t) * \nu_i(t)$$

2.5 Summary

We illustrated the basic concepts of Petri nets with the help of examples. We also introduced a new definition — *guarded due to inheritance place* — which assists in classifying Petri nets as conflict-free nets.

We extended the formalism of D-timed Petri nets to include non-singular nets. We finally established the framework suitable for performance evaluation — that is, the definition of state graph. Once the state graph or the transition graph is obtained from a given timed Petri net model, then we can derive performance measures such as delay or throughput. This is illustrated with the help of two examples in Chapter 4. The procedures for obtaining the state graphs of D-timed and M-timed Petri nets are automated. State graphs shown in Tables 4.2, 4.3 and 4.4 (Chapter 4) are generated by our computer programs.

The transition graphs obtained from the analysis of timed Petri net models are known to be large. In order to speed-up the performance evaluation, we introduce in the next chapter some general reduction rules to reduce the size of the transition graphs.

Chapter 3

Graph Reduction Rules

Probabilistic state or transition graphs obtained from timed Petri net analysis of real-life protocols are known to be large. To facilitate and speed-up the performance evaluation process, we present reduction rules to reduce the size of probabilistic transition graphs. These rules can also be used for calculating the running time of computer programs.

In the past ([BEI 70] , [GRA 73] , [ZUB 86a]), attempts have been made to derive general reduction rules. Beizer[BEI 70] and Graham[GRA 73] employed reduction rules to calculate the running time of computer programs. Zuberek[ZUB 86a] employed similar reduction rules for protocol performance evaluation.

In section 3.1, we present various reduction rules, namely, vertex reduction, junction removal, decision removal, multi-in multi-out vertex removal, merging parallel self-loops, merging parallel edges, and self-loop removal. In section 3.2, we introduce a more general vertex folding rule than those presently existing. For example, Zuberek's[ZUB 86a] definition of vertex folding is valid only for equal transition times. The vertex folding rule presented in section 3.2, on the other hand, holds both for equal and unequal transition times. The derivation of the vertex folding rule is based on the law of conservation of transition time. We present the complete algorithm for vertex folding. To the best of our knowledge, this is the first general vertex folding algorithm. In section 3.3, we summarize the results of this chapter. All these rules are incorporated into a graph reduction software package which can be used in automated performance prediction. Graph reduction software tools such as one developed in this chapter are important and form an integral part of automated protocol performance prediction tools[RUD 83].

To facilitate the presentation, we list below the notations used in this chapter.

TABLE OF NOTATIONS:

$G(G', G''; G_m)$	(reduced;modified) probabilistic transition graph
$V(V', V''; V_m)$	The set of vertices in $G(G', G''; G_m)$
$D(D', D''; D_m)$	The set of directed edges in $G(G', G''; G_m)$
$p(p', p''; p_m)$	The set of probabilites associated with each element of $D(D', D''; D_m)$.
$I(X)(O(X))$	The set of input(output) vertices of vertex X
$\#I(X)$	The number of elements in $I(X)$
$v_i, v_j, v_k, v_l, v_m, v_+$	The elements of V or V' or V'' or V_m
$v_k, v_{k+1}, \ldots, v_{k+n}$	The elements of V or V' or V'' or V_m
$E(v_i, v_j)$	The set of directed edges from v_i to v_j
$L(v_j)$	The set of self-loops of v_j
t_{ij}^k	The transition time associated with the kth directed edge from v_i to v_j
p_{ij}^k	The probability associated with the kth directed edge from v_i to v_j
$p(e), p'(e), p''(e), p_m(e)$	probabilities associated with the directed edge e
LCTT	Law of Conservation of Transition Time
T1,T2,T3,T3-A,T3-B	Transformations
$\emptyset$	empty set
$\wedge(\vee)$	AND(OR)
$f(t)$	false(true)
$\sum$	summation
$\in$	belongs to
$\exists$	there exists
$\forall$	for all
$\notin$	does not belong to
$\nexists$	there does not exist
$<\ >$	if and only if
$\ >$	follows from earlier step or steps
s.t.	such that

3.1 Reduction Rules

In this section, we present seven reduction rules: vertex reduction, junction removal, decision removal, multi-in multi-out vertex removal, merging parallel self-loops, merging parallel edges, and self-loop removal. These rules are also given in [BEI 70], [GRA 73], and [ZUB 86a]. However, we present them in a slightly different manner.

In Chapter 2, we defined probabilistic state graphs, probabilistic transition graphs and showed how to obtain the latter from the former. Since this chapter exclusively deals with the derivation of reduction rules to reduce the size of the probabilistic transition graph, to make it self-contained we define again here the probabilistic transition graph.

Definition 3.1
A *probabilistic transition graph*, $G = (V, D, p)$, is a directed graph where

- V is a set of vertices,
- D is a set of directed edges; each directed edge is a 3-tuple (v_i, v_j, t_{ij}) where v_i is the source vertex, v_j is the destination vertex and t_{ij} is the transition time, and
- p is a set of probabilities associated with each directed edge in D.

Interpretation of t_{ij}^k and p_{ij}^k
If there exists more than one directed edge from v_i to v_j then we denote these directed edges by (v_i, v_j, t_{ij}^k) where k=1,2,...,number of directed edges from v_i to v_j. When there is only one directed edge from v_i to v_j, we omit k and denote this directed edge by (v_i, v_j, t_{ij}). A similar interpretation holds for p_{ij}.

Definition 3.2
A *reduced probabilistic transition graph* $G' = (V', D', p')$ or $G'' = (V'', D'', p'')$ is a probabilistic transition graph obtained by applying any one of the reduction rules presented in this chapter.

Definition 3.3
A *modified probabilistic transition graph* $G_m = (V_m, D_m, p_m)$ is a probabilistic transition graph obtained by either adding a vertex or by relocating the incoming directed edges of a vertex to some other vertex.

Definition 3.4
Vertex v_j is a *junction vertex* if $\#I(v_j) > 1 \wedge \#O(v_j) = 1$

Definition 3.5
Vertex v_j is a *decision vertex* if $\#I(v_j) = 1 \wedge \#O(v_j) > 1$

Definition 3.6

Vertex v_j is a *multi-in multi-out vertex* if $\#I(v_j) > 1 \wedge \#O(v_j) > 1$

Definition 3.7

Let v_i and v_j be two vertices. Then, we denote the set of all directed edges from v_i to v_j by $E(v_i, v_j)$. That is,

$$E(v_i, v_j) = \{(v_l, v_m, t_{lm}^k) \in D : l = i \wedge m = j\}$$

If $\#E(v_i, v_j) > 1$ then we say that v_i has *parallel edges* to v_j.

Definition 3.8

Vertex v_j has a *self-loop* if $v_j \in I(v_j)$. We denote the set of all self-loops of v_j by $L(v_j)$. That is,

$$L(v_j) = \{(v_l, v_m, t_{lm}^k) \in D : l = j \wedge m = j\}$$

In this chapter, we occasionally refer to probabilistic transition graphs as simply graphs. Also, if a set is singleton, we omit braces.

Rule 1: Vertex reduction

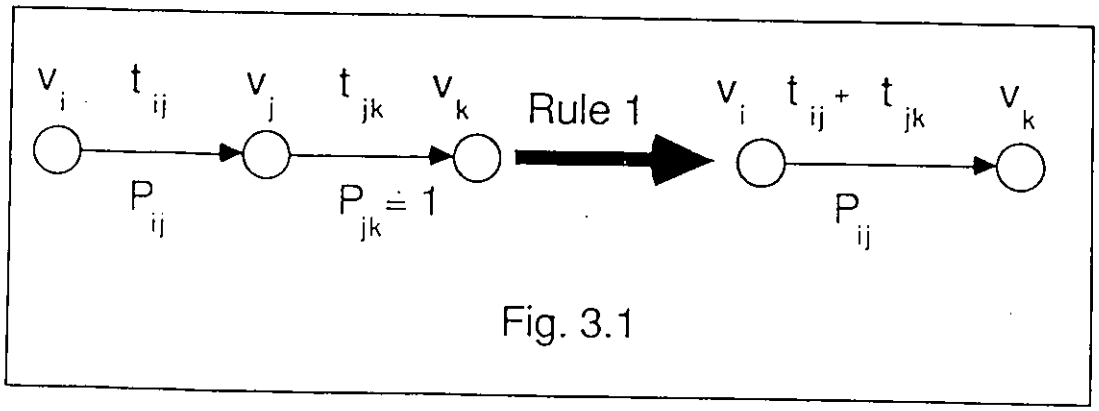
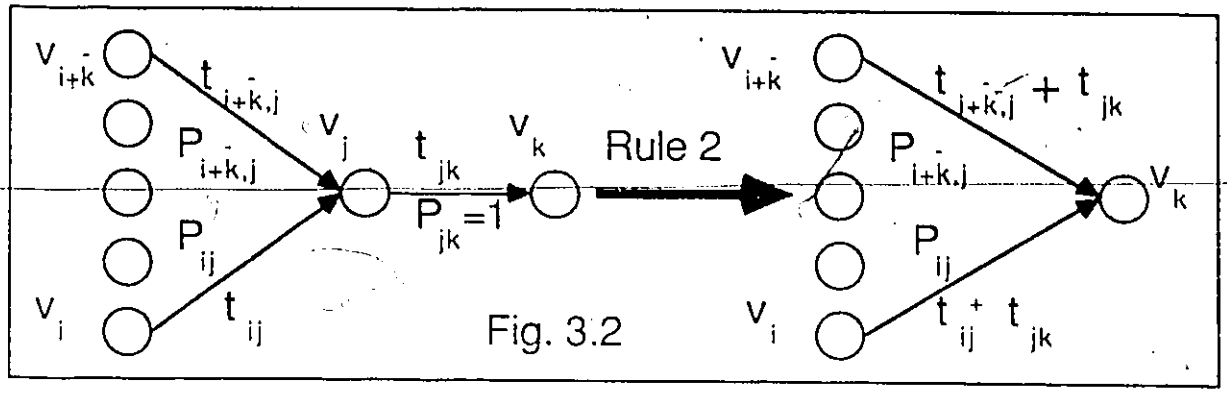


Fig. 3.1

If $I(v_j) = v_i \wedge O(v_j) = v_k$
Then $V' \leftarrow V - v_j$
$D' \leftarrow D - (v_i, v_j, t_{ij}) - (v_j, v_k, t_{jk}) + (v_i, v_k, t_{ij} + t_{jk})$

$$p'(v_l, v_m, t_{lm}) = \begin{cases} p(v_i, v_j, t_{ij}) & \text{if } l = i \wedge m = k \\ p(v_l, v_m, t_{lm}) & \text{otherwise} \end{cases}$$
$$\forall (v_l, v_m, t_{lm}) \in D'$$

Rule 2: Junction removal

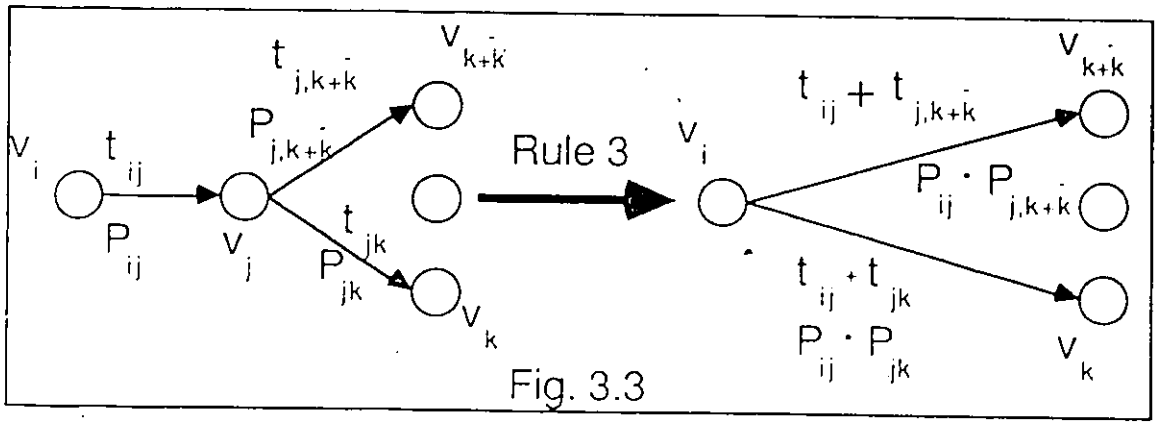


If $\#I(v_j) = n \wedge O(v_j) = v_k \wedge n > 1$
Then $V' \leftarrow V - v_j$
$D' \leftarrow D - (v_j, v_k, t_{jk}) - (v_l, v_j, v_{lj}) + (v_l, v_k, t_{lj} + t_{jk}) \qquad \forall v_l \in I(v_j)$

$$p'(v_l, v_m, t_{lm}) \leftarrow \begin{cases} p(v_l, v_j, t_{lj}) & \text{if } v_l \in I(v_j) \wedge m = k \\ p(v_l, v_m, t_{lm}) & \text{otherwise} \end{cases}$$
$$\forall (v_l, v_m, t_{lm}) \in D'$$

Note: When $n = 1$, this rule becomes Rule 1 : Vertex reduction.

Rule 3: Decision removal



If $I(v_j) = v_i \wedge \#O(v_j) = n \wedge n > 1$
Then $V' = V - v_j$
$D' = D - (v_i, v_j, t_{ij}) - (v_j, v_m, t_{jm}) + (v_i, v_m, t_{ij} + t_{jm}) \qquad \forall v_m \in O(v_j)$

$$p'(v_l, v_m, t_{lm}) = \begin{cases} p(v_i, v_j, t_{ij}) * p(v_j, v_m, t_{jm}) & \text{if } l = i \wedge v_m \in O(v_j) \\ p(v_l, v_m, t_{lm}) & \text{otherwise} \end{cases}$$
$$\forall (v_l, v_m, t_{lm}) \in D'$$

Note: When $n = 1$, this rule becomes Rule 1 : Vertex reduction.

Rule 4: Multi-in Multi-out Vertex removal

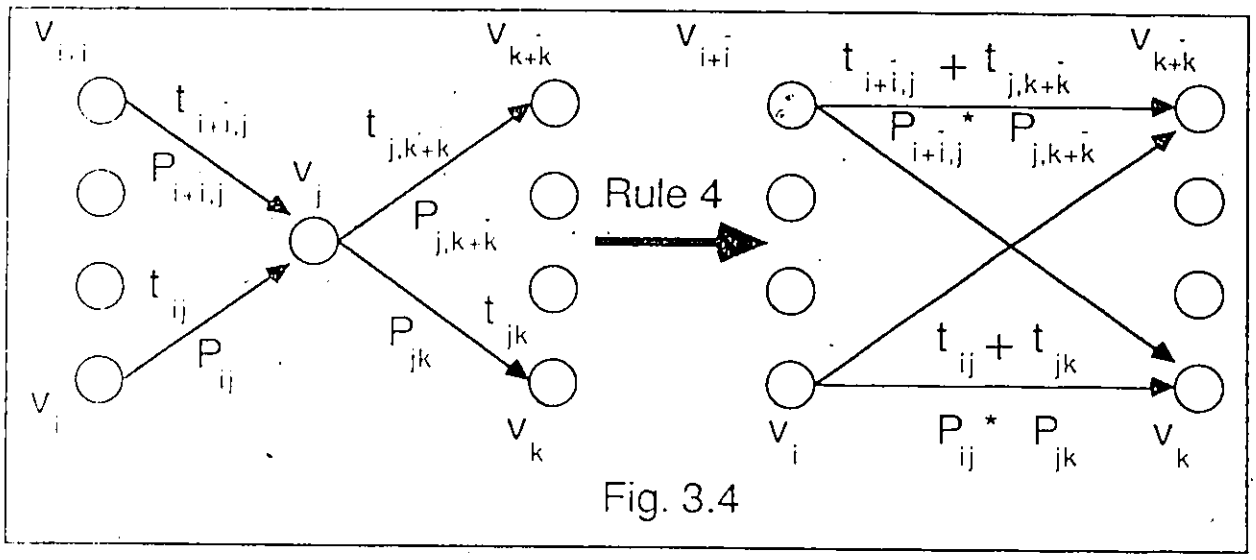


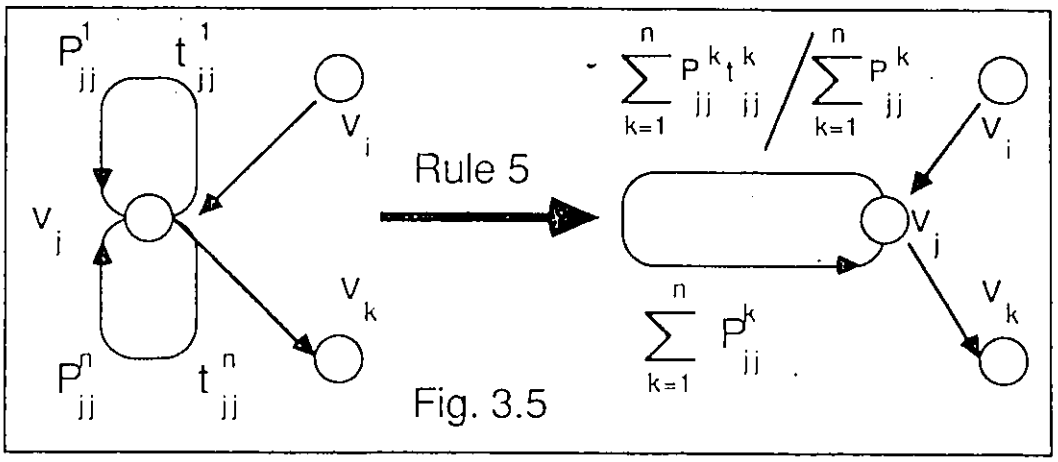
Fig. 3.4

If $\#I(v_j) = m \wedge \#O(v_j) = n \wedge m > 1 \wedge n > 1$
Then $V' = V - v_j$
$D' = D - (v_l, v_j, t_{lj}) - (v_j, v_m, t_{jm}) + (v_l, v_m, t_{lj} + t_{jm})$
$\forall v_l = I(v_j) \wedge \forall v_m \in O(v_j) \wedge \forall v_l \in I(v_j)[\forall v_m \in O(v_j)]$

$$p'(v_l, v_m, t_{lm}) = \begin{cases} p(v_l, v_j, t_{lj}) * p(v_j, v_m, t_{jm}) & \text{if } v_l \in I(v_j) \wedge v_m \in O(v_j) \\ p(v_l, v_m, t_{lm}) & \text{otherwise} \end{cases}$$
$$\forall (v_l, v_m, t_{lm}) \in D'$$

Note: When $m = 1$, $n = 1$ Rule 4 becomes Rule 1.
When $m > 1$, $n = 1$ Rule 4 becomes Rule 2.
When $m = 1$, $n > 1$ Rule 4 becomes Rule 3.

Rule 5: Merging parallel self-loops

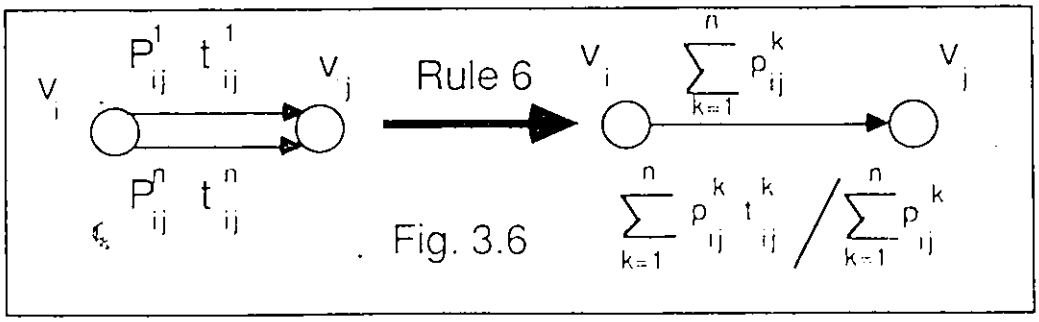


If $\#L(v_j) = n \wedge n \geq 2$
Then $V' \leftarrow V$

$$D' \leftarrow D - L(v_j) + (v_j, v_j, \frac{\sum_{k=1}^{n} p_{jj}^k t_{jj}^k}{\sum_{k=1}^{n} p_{jj}^k}).$$

$$p'(v_l, v_m, t_{lm}) \leftarrow \begin{cases} \sum_{k=1}^{n} p_{jj}^k & \text{if } l = m = j \\ p(v_l, v_m, t_{lm}) & \text{otherwise} \end{cases}$$
$$\forall (v_l, v_m, t_{lm}) \in D'$$

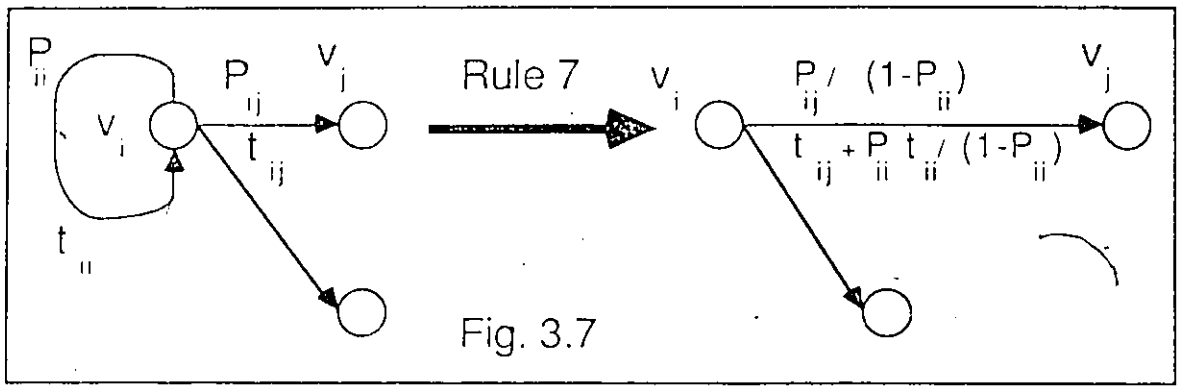
Rule 6: Merging parallel edges



If $\#E(v_i, v_j) = n \wedge n \geq 2$
Then $V' \leftarrow V$
$D' \leftarrow D - E(v_i, v_j) + (v_i, v_j, \frac{\sum_{k=1}^{n} p_{ij}^k t_{ij}^k}{\sum_{k=1}^{n} p_{ij}^k})$

$$p'(v_l, v_m, t_{lm}) \leftarrow \begin{cases} \sum_{k=1}^{n} p_{ij}^k & \text{if } l = i \wedge m = j \\ p(v_l, v_m, t_{lm}) & \text{otherwise} \end{cases}$$
$$\forall (v_l, v_m, t_{lm}) \in D'$$

Rule 7: Self-loop removal



If $\#L(v_i) = 1 \wedge \nexists v_j : v_j \neq v_i \wedge v_j \in O(v_i)$
Then $V' \leftarrow V$
$D' \leftarrow D - L(v_i) - (v_i, v_j, t_{ij}) + (v_i, v_j, t_{ij} + p_{ii} * t_{ii}/(1 - p_{ii}))$

$$p'(v_l, v_m, t_{lm}) \leftarrow \begin{cases} p_{ij}/(1 - p_{ii}) & \text{if } l = i \wedge m = j \\ p(v_l, v_m, t_{lm}) & \text{otherwise} \end{cases}$$
$$\forall (v_l, v_m, t_{lm}) \in D'$$

Note: Modified probabilities and transition times must be calculated for all other remaining edges with v_i as a source vertex.

3.2 Derivation of Vertex folding rule

In this section, we present the derivation of vertex folding rule. It is based on the law of conservation of transition time(LCTT).

Law of Conservation of Transition Time(LCTT)

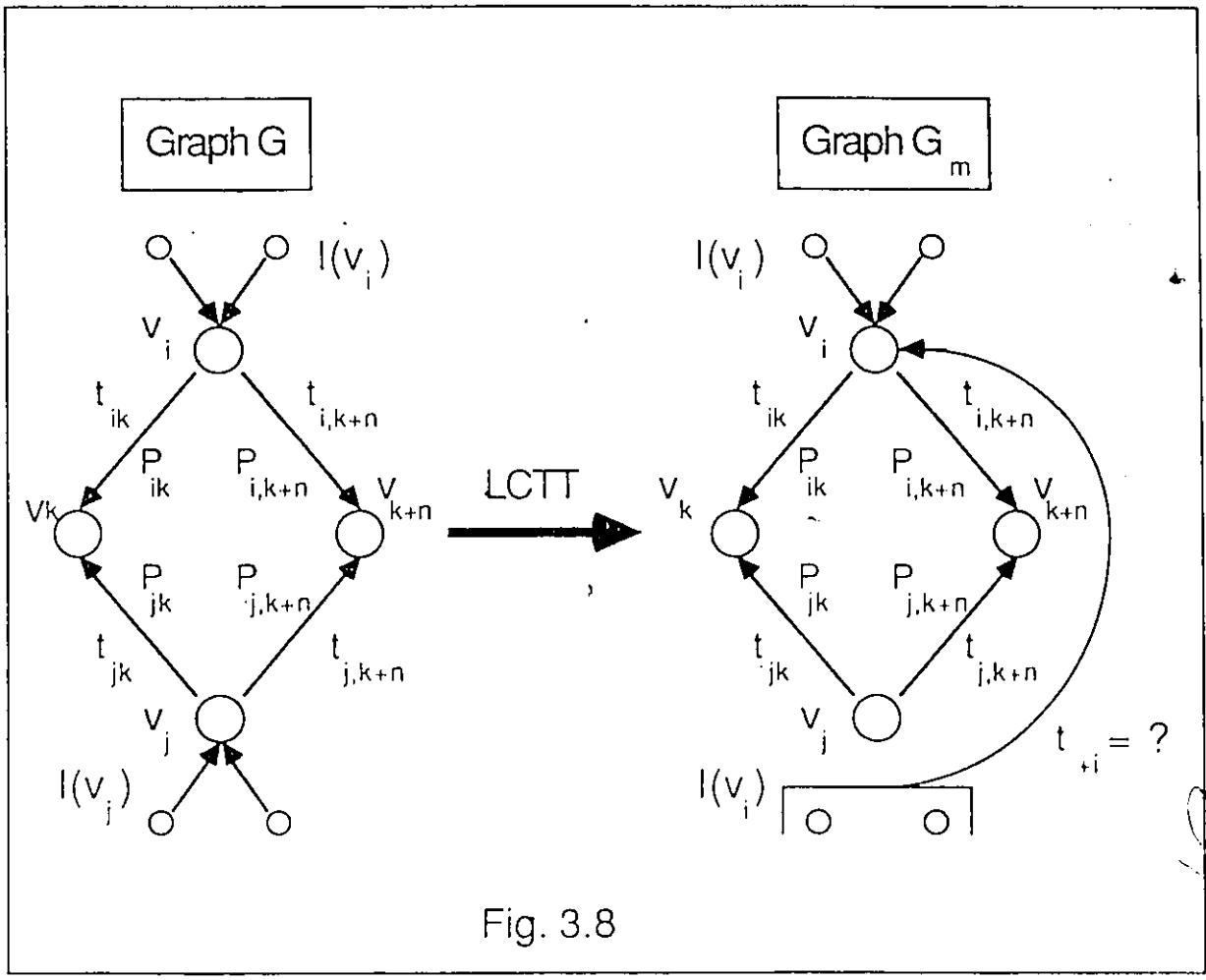


Fig. 3.8

1.	$v_i \neq v_j$
2.	$\#O(v_i) > 1 \wedge \#O(v_j) > 1$
3.	$O(v_i) = O(v_j)$
4.	$p_{i,k+\bar{k}} = p_{j,k+\bar{k}} \qquad \forall \bar{k} = 0, 1, \ldots, \#O(v_i) - 1$
5.	$t_{i-} \quad \forall v_- \in O(v_i)$ are equal
6.	$t_{j-} \quad \forall v_- \in O(v_j)$ are equal
7.	$v_i \notin O(v_i) \wedge v_j \notin O(v_j)$

Table 0.1: Conditions of LCTT

Let $v_i, v_j \in V$ (Fig.3.8) and satisfy the conditions in Table 3.1. Condition 7 will be relaxed later(Transformation T3). Let $t_{i-} \neq t_{j-}$. The case, $t_{i-} = t_{j-}$ is trivial and is treated in Procedure EQUAL(section 3.2). It is interesting to note that Zuberek's[ZUB 86a] definition of vertex folding is valid only for the case, $t_{i-} = t_{j-}$. The analysis presented here includes the case, $t_{i-} \neq t_{j-}$. We present the complete algorithm for vertex folding in the latter part of this section(Rule 8 : Vertex folding). To the best of our knowledge, this is the first general vertex folding algorithm.

Objective: To delete vertex v_j (that is, to fold v_j onto v_i)

Case i. Let $I(v_j) \neq \emptyset$

The steps needed to be taken before deleting vertex v_j are:

1. Delete directed edges $(v_+, v_j, t_{+j}) \qquad \forall v_+ \in I(v_j)$
2. For every deleted directed edge in step 1, add a new directed edge (v_+, v_i, t_{+i})
3. Calculate p_{+i} and t_{+i} for every new directed edge as described below

Hence, the modified probabilistic transition graph G_m(Fig.3.8) can be obtained from graph G(Fig.3.8) if

$$V_m \leftarrow V$$

$$D_m \leftarrow D - (v_+, v_j, t_{+j}) + (v_+, v_i, t_{+i}) \qquad \forall v_+ \in I(v_j)$$

$$p_m(v_l, v_m, t_{lm}) \leftarrow \begin{cases} p(v_+, v_j, t_{+j}) & \text{if } v_l = v_+ \in I(v_j) \wedge m = i \\ p(v_l, v_m, t_{lm}) & \text{otherwise} \end{cases}$$

$$\forall (v_l, v_m, t_{lm}) \in D_m$$

and t_{+i} is given by the following equation:

$$\boxed{t_{+i} = t_{+j} + t_{j-} - t_{i-}} \qquad \forall v_+ \in I(v_j) \tag{3.1}$$

Case ii. Let $I(v_j) = \emptyset$

Let $I(v_i) \neq \emptyset$. The case, $I(v_i) = \emptyset$ is trivial and is considered in the later part of this section(Algorithm for vertex folding).

The only step needed to be taken before deleting vertex v_j is to adjust t_{+i} and t_{i-} $\forall v_+ \in I(v_i), v_- \in O(v_i)$ as described below.

Hence, the modified probabilistic transition graph G_m can be obtained from graph G if

$$V_m \leftarrow V$$

$$D_m \leftarrow D(\text{modified } t_{+i}, t_{i-} \forall v_+ \in I(v_i), v_- \in O(v_i) \text{ as given below})$$

$$p_m(v_l, v_m, t_{lm}) \leftarrow p(v_l, v_m, t_{lm}) \qquad \forall (v_l, v_m, t_{lm}) \in D_m$$

$$\boxed{t_{i-_{new}} = t_{j-}} \qquad \forall v_- \in O(v_i) \tag{3.2}$$

$$\boxed{t_{+i_{new}} = t_{+i_{old}} + t_{i-} - t_{j-}} \qquad \forall v_+ \in I(v_i) \tag{3.3}$$

Equations (3.1) and (3.2)-(3.3) are called the *Laws of Conservation of Transition Time*(LCTT).

We have now established the basis for deleting vertex v_j from G, a process called the vertex folding. The only restriction is that t_{+i} and $t_{+i_{new}}$, calculated using equations (3.1) and (3.3) respectively must be non-negative. Hence, the transformations for deleting vertex v_j(that is, folding v_j onto v_i) for cases(i) and (ii) respectively are as follows:

Transformation T1:

$$V' \leftarrow V - v_j$$

$$D' \leftarrow D - (v_j, v_-, t_{j-}) - (v_+, v_j, t_{+j}) + (v_+, v_i, t_{+i}) \qquad \forall v_- \in O(v_j), v_+ \in I(v_j)$$

and p', t_{+i} are same as p_m and t_{+i} respectively given in case(i).

Transformation T2:

$$V' \leftarrow V - v_j$$

$$D' \leftarrow D - (v_j, v_-, t_{j-}) \qquad \forall v_- \in O(v_j)$$

and $p', t_{+i_{new}}, t_{i-_{new}}$ are same as p_m, $t_{+i_{new}}$ and $t_{i-_{new}}$ respectively given in case(ii).

Example 3.1

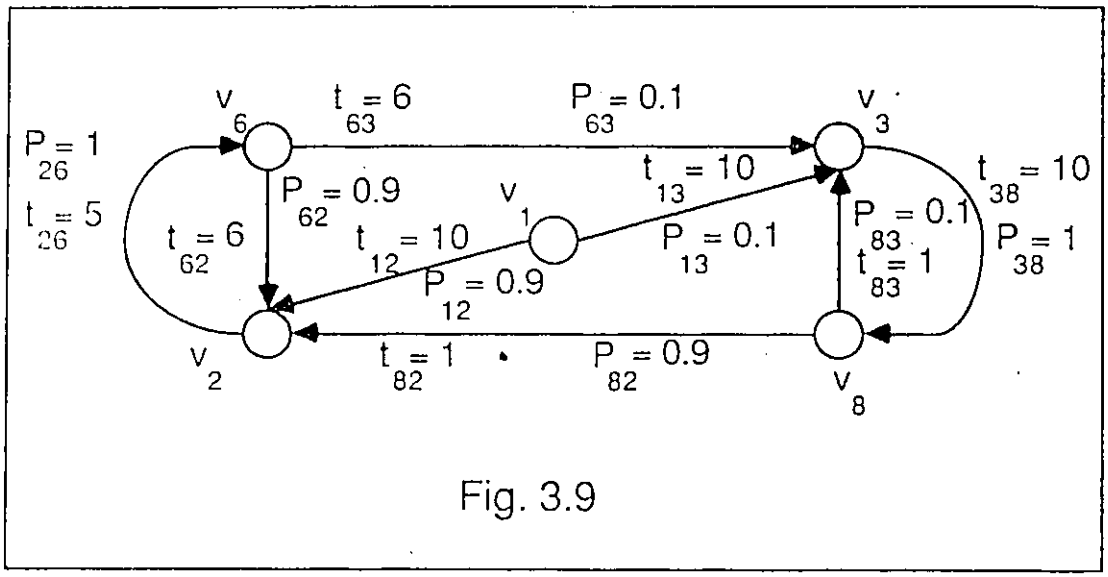


Fig. 3.9

$$V = \{v_1, v_2, v_3, v_6, v_8\}$$

$$D = \left\{ \begin{array}{l} (v_1, v_2, 10), (v_1, v_3, 10), (v_2, v_6, 5), (v_3, v_8, 10) \\ (v_6, v_2, 6), (v_6, v_3, 6), (v_8, v_2, 1), (v_8, v_3, 1) \end{array} \right\}$$

$$p = \{ 0.9, 0.1, 1.0, 1.0, 0.9, 0.1, 0.9, 0.1 \}$$

The probabilistic transition graph shown in Fig.3.9 is taken from [ZUB 86b](after applying Rule 1 : Vertex reduction) to demonstrate LCTT, Transformation T1 and Transformation T2.

Case-i. Let $v_i = v_6, v_j = v_8, v_k = v_2, v_{k+1} = v_3$

Referring to Fig.3.9, we obtain:

$$O(v_6) = O(v_8) = \{v_2, v_3\}$$

$$p_{6,2} = p_{8,2} = 0.9$$

$$p_{6,3} = p_{8,3} = 0.1$$

$$t_{6,2} = t_{6,3} = 6 = t_{i-}$$

$$t_{8,2} = t_{8,3} = 1 = t_{j-}$$

$$v_6 \notin O(v_6), v_8 \notin O(v_8)$$

$$I(v_8) = \{v_3\} \neq \emptyset$$

Hence, all the conditions in Table 3.1 are satisfied. Now, we can obtain the modified probabilistic transition graph as shown in Fig.3.10 using LCTT. That is,

$$V_m = V = \{v_1, v_2, v_3, v_6, v_8\}$$

$$D_m = D - (v_3, v_8, 10) + (v_3, v_6, t_{3,6}) = \left\{ \begin{array}{l} (v_1, v_2, 10), (v_1, v_3, 10), (v_2, v_6, 5), (v_3, v_6, t_{3,6}) \\ (v_6, v_2, 6), (v_6, v_3, 6), (v_8, v_2, 1), (v_8, v_3, 1) \end{array} \right\}$$

$$p_m = \{0.9, 0.1, 1.0, 1.0, 0.9, 0.1, 0.9, 0.1\}$$

and $t_{3,6}$ is calculated as follows:

$$t_{3,6} = t_{3,8} + t_{j-} - t_{i-} = 10 + 1 - 6 = 5$$

Now applying Transformation T1, we obtain the reduced probabilistic transition graph as shown in Fig.3.11.

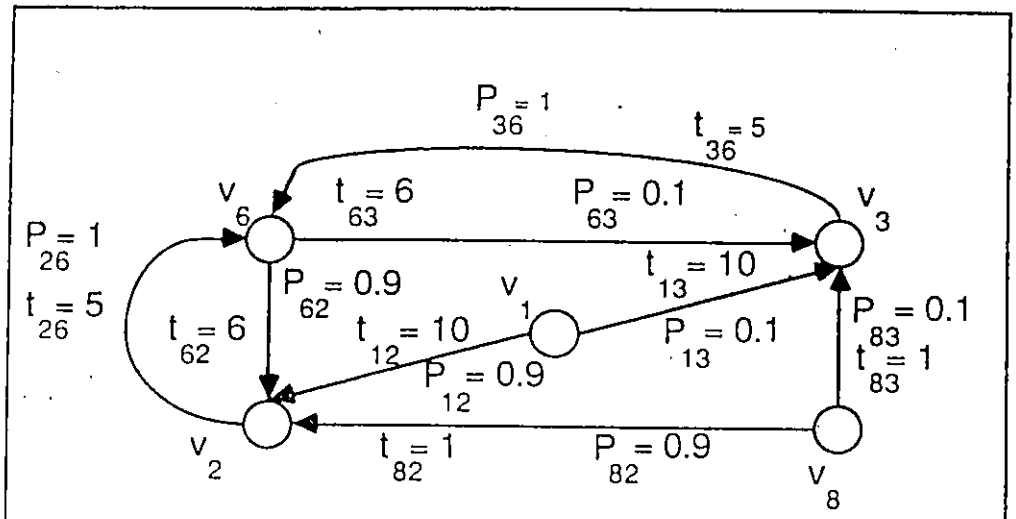


Fig. 3.10

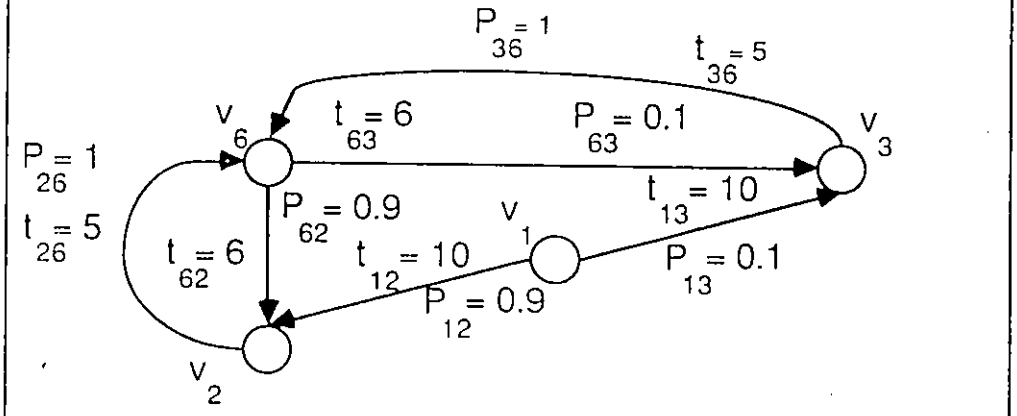


Fig. 3.11

Case ii. Let $v_i = v_6, v_j = v_1, v_k = v_2, v_{k+1} = v_3$

Referring to Fig.3.9, we obtain:

$$O(v_6) = O(v_1) = \{v_2, v_3\}$$

$$p_{6,2} = p_{1,2} = 0.9$$

$$p_{6,3} = p_{1,3} = 0.1$$

$$t_{6,2} = t_{6,3} = 6 = t_i$$

$$t_{1,2} = t_{1,3} = 10 = t_{j-}$$

$$v_6 \notin O(v_6), v_1 \notin O(v_1)$$

$$I(v_1) = 0, I(v_6) = \{v_2\}$$

Hence, all the conditions in Table 3.1 are satisfied. Now, we can obtain the modified probabilistic transition graph as shown in Fig.3.12 using LCTT. That is,

$$V_m = V = \{v_1, v_2, v_3, v_6, v_8\}$$

$$D_m = \left\{ \begin{array}{l} (v_1, v_2, 10), (v_1, v_3, 10), (v_2, v_6, t_{2,6_{new}}), (v_3, v_8, 10) \\ (v_6, v_2, t_{6,2_{new}}), (v_6, v_3, t_{6,3_{new}}), (v_8, v_2, 1), (v_8, v_3, 1) \end{array} \right\}$$

$$p_m = \left\{ 0.9, 0.1, 1.0, 1.0, 0.9, 0.1, 0.9, 0.1 \right\}$$

and $t_{2,6_{new}}, t_{6,2_{new}}, t_{6,3_{new}}$ are calculated as follows:

$$t_{6,2_{new}} = t_{j-} = 10$$

$$t_{6,3_{new}} = t_{j-} = 10$$

$$t_{2,6_{new}} = t_{2,6_{old}} + t_i - t_j = 5 + 6 - 10 = 1$$

Now applying Transformation T2, we obtain the reduced probabilistic transition graph as shown in Fig.3.13.

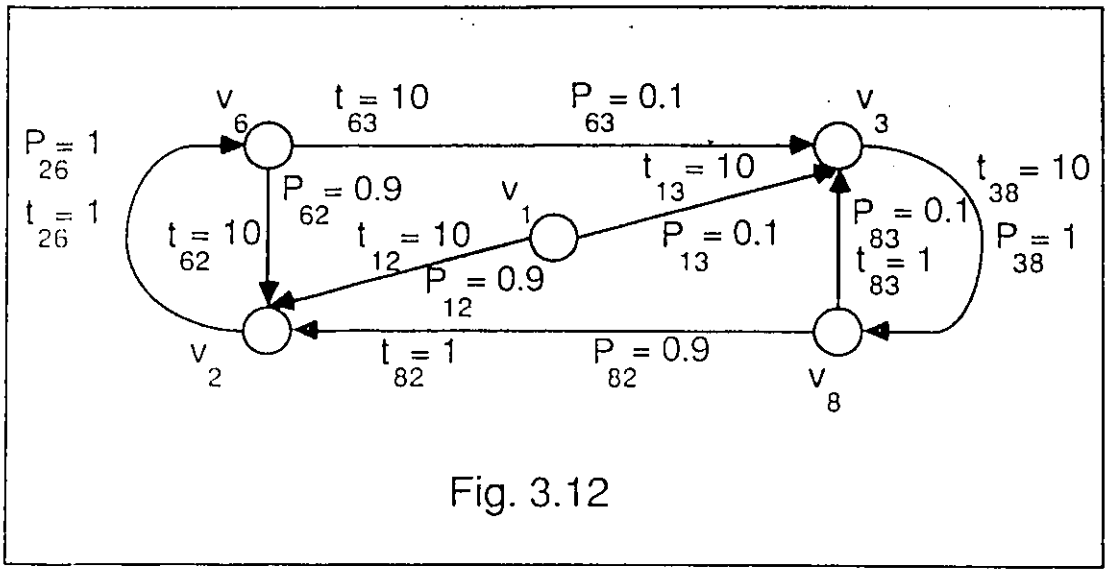


Fig. 3.12

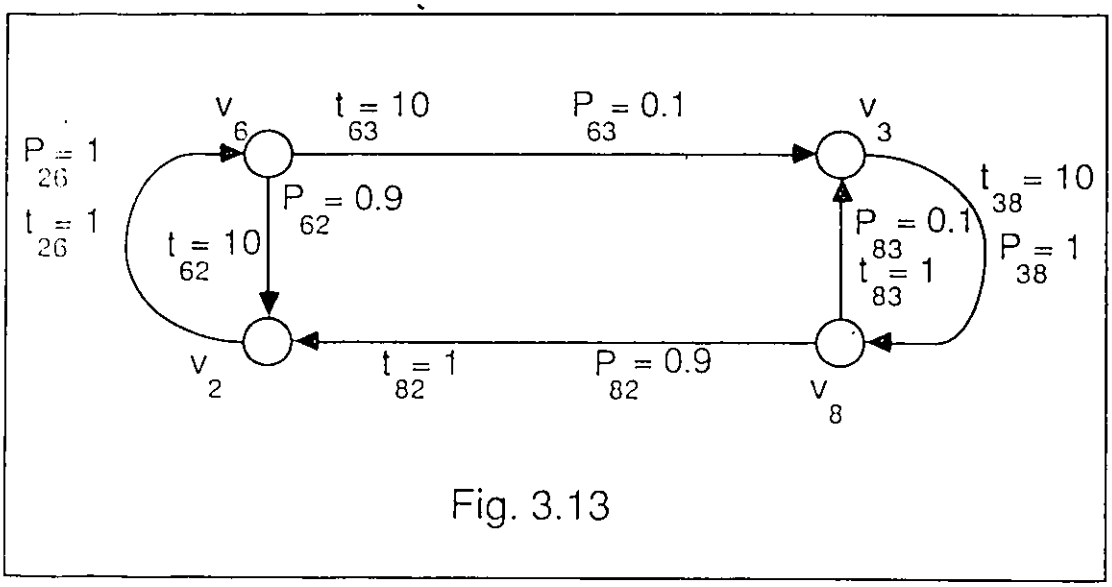


Fig. 3.13

Definition 3.9
Let CVF $= \{v_1, v_2, \ldots, v_n : v_i \in V \wedge n > 1\}$. Then, CVF is said to be a *candidate set for vertex folding* if

1. $\#O(v_i) > 1 \qquad \forall v_i \in \text{CVF}$
2. $O(v_i)$ must be identical $\forall v_i \in \text{CVF}$
3. $\forall k = 0, 1, \ldots, \#O(v_i) - 1 \quad [\forall v_i \in \text{CVF} \quad p_{i,k+k}$ must be equal]

The set of all CVF in a probabilistic transition graph is said to be a *Vertex folding set* and is denoted by VFS. That is,

$\text{VFS} = \{\text{CVF}_i : \text{CVF}_i \text{ is a CVF}\}$

Example 3.2

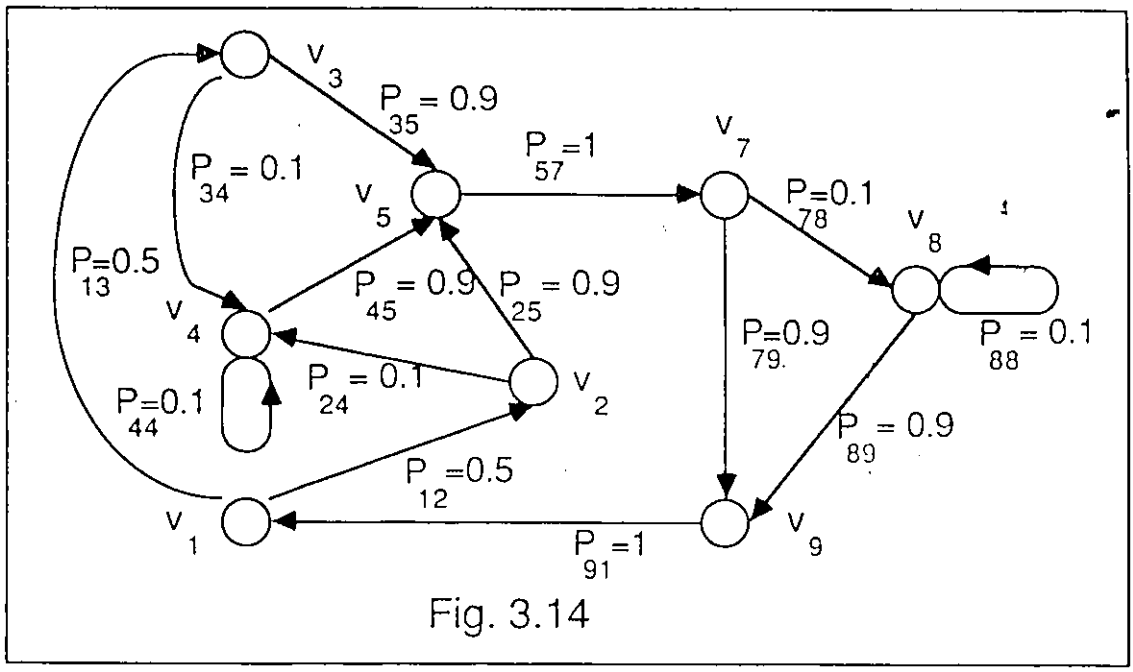


Fig. 3.14

Referring to Fig.3.14, we obtain:

$O(v_2) = O(v_3) = O(v_4) = \{v_4, v_5\}$

$p_{2,4} = p_{3,4} = p_{4,4} = 0.1 \quad / \quad p_{2,5} = p_{3,5} = p_{4,5} \quad 0.9$

Therefore, $\text{CVF}_1 \quad \{v_2, v_3, v_4\}$

$$O(v_7) = O(v_8) = \{v_8, v_9\}$$

$$p_{7,8} = p_{8,8} = 0.1 \quad \wedge \quad p_{7,9} = p_{8,9} = 0.9$$

Therefore, $\mathrm{CVF}_2 = \{v_7, v_8\}$ and $\mathrm{VFS} = \{\mathrm{CVF}_1, \mathrm{CVF}_2\} = \{\{v_2, v_3, v_4\}, \{v_7, v_8\}\}$.

Note: We did not include transition times in Fig.3.14 since they are not needed for computing CVF.

Definition 3.10

Vertex folding rule is said to be *enabled* if $\mathrm{VFS} \neq \emptyset$.

Definition 3.11

Let $v_i, v_j \in \mathrm{CVF}$. Then, we say that edges (v_i, v_-, t_{i-}) and (v_j, v_-, t_{j-}) for every $v_- \in O(v_i)$ are said to have *equal transition time* if

$$t_{i-} \quad \forall v_- \in O(v_i) \quad \text{are equal} \quad \wedge \quad t_{j-} \quad \forall v_- \in O(v_j) \quad \text{are equal} \quad \wedge \; . \; t_{i-} = t_{j-}$$

and *unequal transition time* if

$$\exists v_- \in O(v_i) : t_{i-} \neq t_{j-}$$

Example 3.3 [Equal transition time]

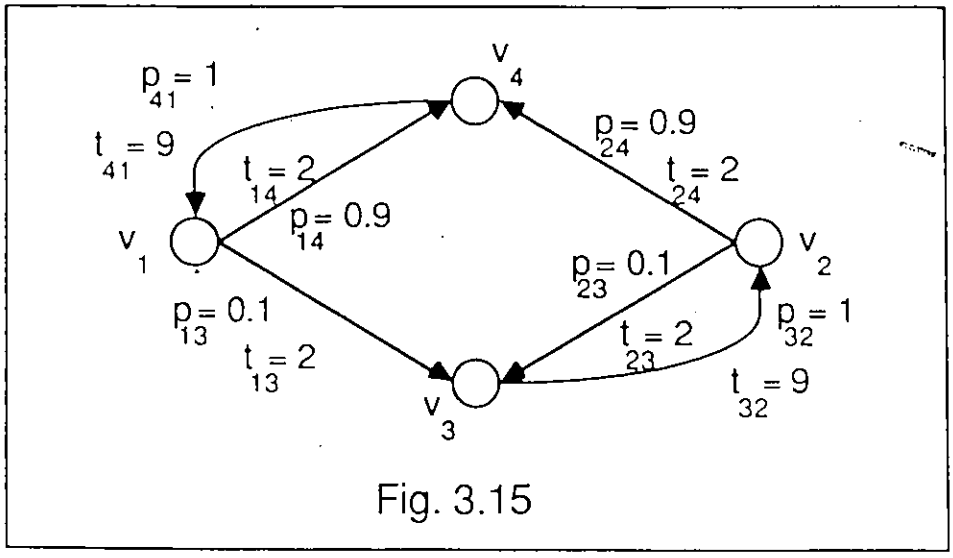


Fig. 3.15

Referring to Fig.3.15, we obtain:

$$O(v_1) = O(v_2) = \{v_3, v_4\}$$

$p_{1,3} = p_{2,3} = 0.1$

$p_{1,4} = p_{2,4} = 0.9$

Therefore, CVF $= \{v_1, v_2\}$

$t_{1,3} = t_{2,3} = 2$

$t_{1,4} = t_{2,4} = 2$

Example 3.4 [Unequal transition time]

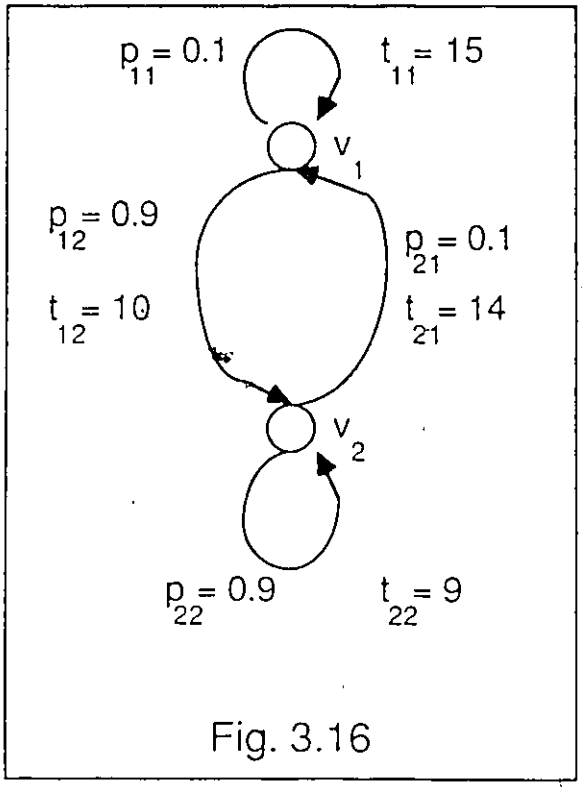


Fig. 3.16

Referring to Fig.3.16, we obtain:

$O(v_1) = O(v_2) = \{v_1, v_2\}$

$p_{1,1} = p_{2,1} = 0.1$

$p_{1,2} \cdot p_{2,2} = 0.9$

Therefore, CVF $= \{v_1, v_2\}$

$t_{1,1} = 15 \neq t_{2,1} = 14 \quad \wedge \quad t_{1,2} = 10 \neq t_{2,2} = 9$

Definition 3.12

Vertex folding rule is said to be *firable* if it is enabled.

The algorithm which specifies the firing rules for vertex folding is described next.

Rule 8 : Vertex folding

Let vertices v_i and v_j in a graph G be members of the same CVF $\in$ VFS. Then, G' as a result of vertex folding (v_j onto v_i) can be obtained as follows:

If $(v_i, v_{+i}, t_{+i}) \wedge (v_j, v_{+j}, t_{+j}) \quad \forall v_{+} \in O(v_i)$ have equal transition time
Then Procedure EQUAL
Else Procedure UNEQUAL
Endif

Procedure EQUAL
$V' = V - v_j$
Case $(I(v_j) = \emptyset)$ **Of**
t: $D' = D - (v_j, v_{+j}, t_{+j}) \quad \forall v_{+} \in O(v_j)$
f: **Case** $(v_j \notin I(v_j))$ **Of**
t: $D' = D - (v_j, v_{+j}, t_{+j}) + (v_{+}, v_j, t_{+j}) + (v_{+}, v_i, t_{+j})$
$\forall v_{+} \in O(v_j) \wedge \forall v_{+} \in I(v_j)$
f: $D' = D - (v_j, v_{+j}, t_{+j}) - (v_{+}, v_j, t_{+j}) + (v_l, v_i, t_{+j})$
$\forall v_{+} \in O(v_j) \wedge \forall v_{+} \in I(v_j) \wedge \forall v_l \in I(v_j) \wedge v_l \neq v_j$
End {inner Case}
End {outer Case}
End Procedure EQUAL

Note: In procedure EQUAL, we did not mention about p' since we assume that whenever a directed edge is deleted so also is its associated probability and also whenever a directed edge is relocated so also is its associated probability.

```
Procedure UNEQUAL
        Case (I(v_j) = ∅, I(v_i) = ∅) Of
          (t,t):  G' ← G
          (t,f):  Case (v_i ∉ I(v_i)) Of
                       t: T2
                       f: T3,T2
                  End {Case (t,f)}
          (f,t),(f,f):
                  Case (v_j ∉ I(v_j), v_i ∉ I(v_i)) Of
                       (t,t): T1
                       (t,f),(f,t),(f,f): T3,T1
                  End {Case (f,t),(f,f)}
        End{main Case}
End Procedure UNEQUAL


Transformation T3
        If ∃v_{k+k} ∈ V : v_{k+k} ≠ v_i ∧ v_{k+k} ≠ v_j ∧ v_{k+k} ∈ O(v_i)
           Then T3-A
           Else T3-B,T3-A
        Endif
End Transformation T3
```

Before we describe transformations T3-A and T3-B, we state and prove a theorem on which T3-A and T3-B are based.

Theorem 3.1 (From Self-loop to Junction)

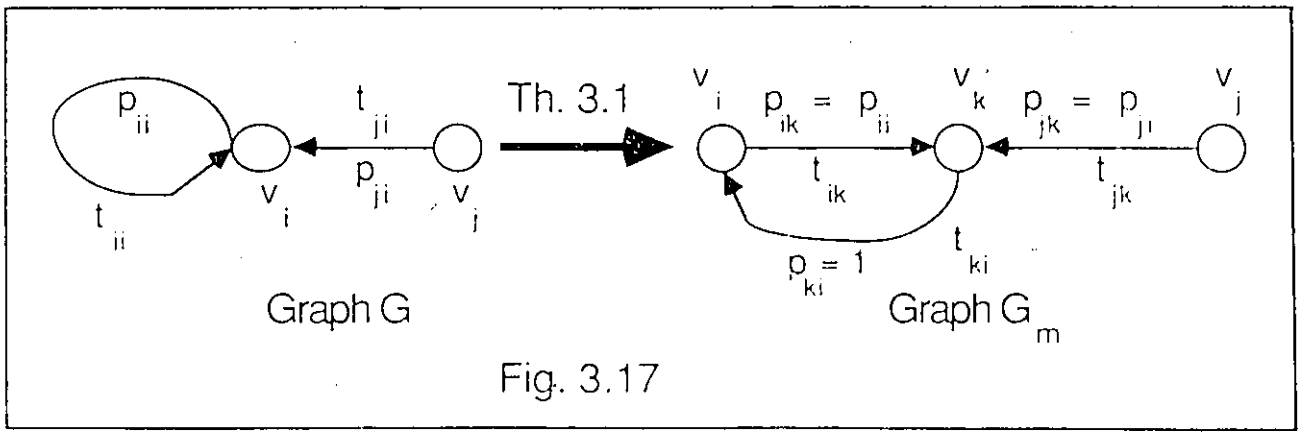


Fig. 3.17

Let v_i and v_j be two vertices in G (Fig.3.17). Then, the graph G_m is constructed by the following set of transformations:

$$V_m \leftarrow V + v_k$$

$$D_m \leftarrow D - (v_i, v_j, t_{ij}) - (v_j, v_i, t_{ji}) + (v_i, v_k, t_{ik}) + (v_k, v_i, t_{ki}) + (v_j, v_k, t_{jk})$$

$$p_m(v_l, v_m, t_{lm}) \leftarrow \begin{cases} p(v_i, v_j, t_{ij}) & \text{if } l = i \wedge m = k \\ p(v_j, v_i, t_{ji}) & \text{if } l = j \wedge m = k \\ 1 & \text{if } l = k \wedge m = i \\ p(v_l, v_m, t_{lm}) & \text{otherwise} \end{cases}$$

$$\forall (v_l, v_m, t_{lm}) \in D_m$$

and, t_{ik}, t_{jk} and t_{ki} are given as follows:

case(i) t_{ik} and t_{jk} are known

$$t_{ki} \leftarrow (t_{ij} - t_{ik}) \text{ or } (t_{ji} - t_{jk})$$

case(ii) t_{ik} and t_{jk} are not known

If $t_{ji} > t_{ij}$ then choose t_{ik} from $[0, t_{ij}]$
$t_{jk} \leftarrow t_{ik} + t_{ji} - t_{ij}$
else choose t_{jk} from $[0, t_{ji}]$
$t_{ik} \leftarrow t_{jk} + t_{ij} - t_{ji}$

Endif

$$t_{ki} \leftarrow (t_{ij} - t_{ik}) \text{ or } (t_{ji} - t_{jk})$$

Proof: (By deconstruction)

We will show that the reduced graph(let it be G″) obtained by removing the junction at v_k in G_m (hence we call it deconstruction) is in fact graph G. Let the graph G″ be characterized by (V'', D'', p'').

Applying Rule 2 : Junction removal to G_m, we obtain,

V'' $V_m - v_k = V + v_k - v_k = V$

D'' $D_m - (v_i, v_k, t_{ik}) - (v_k, v_i, t_{ki}) - (v_j, v_k, t_{jk}) + (v_i, v_i, t_{ik} + t_{ki})$
$+ (v_j, v_i, t_{jk} + t_{ki})$

$D - (v_i, v_i, t_{ij}) - (v_j, v_i, t_{ji}) + (v_i, v_k, t_{ik}) + (v_k, v_i, t_{ki})$
$+ (v_j, v_k, t_{jk}) - (v_i, v_k, t_{ik}) - (v_k, v_i, t_{ki}) - (v_j, v_k, t_{jk})$
$+ (v_i, v_i, t_{ik} + t_{ki}) + (v_j, v_i, t_{jk} + t_{ki})$

$D - (v_i, v_i, t_{ij}) - (v_j, v_i, t_{ji}) + (v_i, v_i, t_{ik} + t_{ki}) + (v_j, v_i, t_{jk} + t_{ki})$

D since $t_{ki} = t_{ij} - t_{ik}$
and $t_{ki} = t_{ji} - t_{jk}$

$$
\begin{aligned}
& p''(v_l, v_m, t_{lm}) \\
&= \begin{cases} p_m(v_i, v_k, t_{ik}) * p_m(v_k, v_i, t_{ki}) & \text{if } l = i \wedge m = i \\ p_m(v_j, v_k, t_{jk}) * p_m(v_k, v_i, t_{ki}) & \text{if } l = j \wedge m = i \\ p_m(v_l, v_m, t_{lm}) & \text{otherwise} \end{cases} \\
& \quad \forall (v_l, v_m, t_{lm}) \in D'' \\
&= \begin{cases} p_m(v_i, v_i, t_{ii}) & \text{if } l = i \wedge m = i \\ p_m(v_j, v_i, t_{ji}) & \text{if } l = j \wedge m = i \\ p_m(v_l, v_m, t_{lm}) & \text{otherwise} \end{cases} \\
& \quad \forall (v_l, v_m, t_{lm}) \in D \\
&= p_m(v_l, v_m, t_{lm}) \quad \forall (v_l, v_m, t_{lm}) \in D
\end{aligned}
$$

$$
\begin{aligned}
&\text{Since} \quad t_{ki} = t_{ii} - t_{ik} = t_{ji} - t_{jk} \\
&\Longrightarrow \quad t_{ik} = t_{jk} + t_{ii} - t_{ji} \\
&\text{Also} \quad t_{jk} = t_{ik} + t_{ji} - t_{ii}
\end{aligned}
$$

Now, we will prove the results stated in Case(ii).

Case(a): Choose t_{ik} from $[0, t_{ii}]$

Suppose, we choose $t_{ik} : t_{ik} \notin [0, t_{ii}]$. There are two options:

(L) $t_{ik} < 0$

(R) $t_{ik} > t_{ii}$

Option(L) is clearly not possible since t_{ik} must be non-negative.

Option(R): Let $t_{ik} = t_{ii} + \Delta t$ and $\Delta t > 0$ be arbitrary.

$$
\begin{aligned}
\text{Since, } t_{ki} &= t_{ii} - t_{ik} \\
\Longrightarrow t_{ki} &= t_{ii} - t_{ii} - \Delta t \\
&= -\Delta t \\
&< 0
\end{aligned}
$$

This is contradiction.

Case(b): Choose t_{jk} from $[0, t_{ji}]$

Proof is similar to case(a).
Suppose, we choose $t_{jk} : t_{jk} \notin [0, t_{ji}]$. There are two options:

(L) $t_{jk} < 0$

(R) $t_{jk} > t_{ji}$

Option(L) is clearly not possible since t_{jk} must be non-negative.

Option(R): Let $t_jk = t_{ji} + \Delta t$ and $\Delta t > 0$ be arbitrary.

$$\begin{aligned} \text{Since, } t_{ki} &= t_{ji} - t_{jk} \\ \Rightarrow t_{ki} &= t_{ji} - t_{ji} - \Delta t \\ &= -\Delta t \\ &< 0 \end{aligned}$$

This is contradiction.

Q.E.D.

Next, we describe transformations T3-A and T3-B which utilize the results of Theorem 3.1. Specifically, T3-A utilizes the results stated in Case(i) of Theorem 3.1 whereas T3-B utilizes the results stated in Case(ii) of Theorem 3.1. Also, observe that the results in Theorem 3.1 are valid for both cases: $p_{ii} = p_{ji}$ and $p_{ii} \neq p_{ji}$ since p_{ii} and p_{ji} are arbitrary and no assumptions are made about their equality or otherwise. However, in transformations T3-A and T3-B, when there is more than one self-loop at v_i and/or v_j, we always select such a pair of edges for which $p_{ii} = p_{ji}$ and/or $p_{ij} = p_{jj}$.

Transformation T3-A

Objective: Removal of all self-loops at v_i and v_j, and constucting junctions instead.

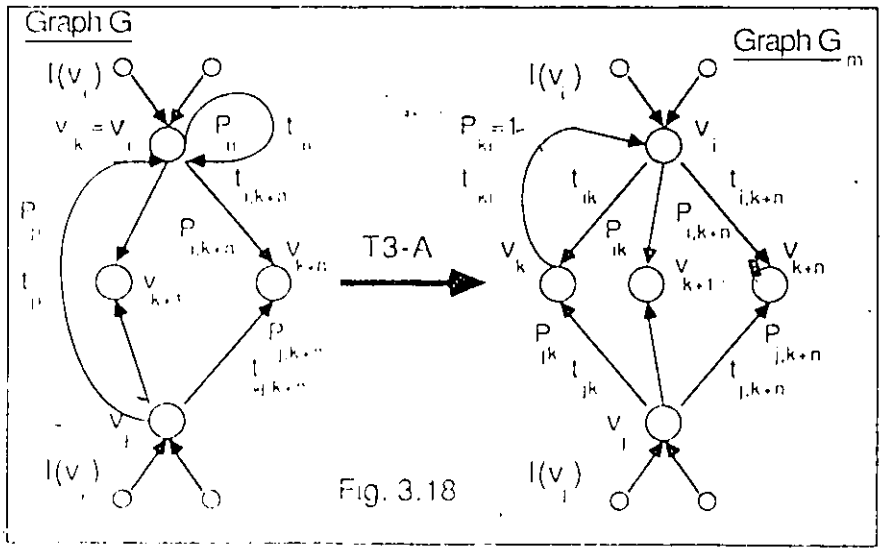


Fig. 3.18

If $\quad \forall v_{k+k} : v_{k+k} \neq v_i \wedge v_{k+k} \neq v_j \wedge v_{k+k} \in O(v_i)$
$[t_{i,k+k}$ are equal $\wedge\ t_{j,k+k}$ are equal$]$

Then $\quad V_m \leftarrow V + v_k$
$D_m \leftarrow D - (v_i, v_i, t_{ii}) - (v_j, v_i, t_{ji}) + (v_i, v_k, t_{ik})$
$+(v_k, v_i, t_{ki}) + (v_j, v_k, t_{jk})$

$$p_m(v_l, v_m, t_{lm}) \leftarrow \begin{cases} p(v_j, v_i, t_{ji}) & \text{if } l = j \wedge m = k \\ p(v_i, v_i, t_{ii}) & \text{if } l = i \wedge m = k \\ 1 & \text{if } l = k \wedge m = i \\ p(v_l, v_m, t_{lm}) & \text{otherwise} \end{cases}$$

$$\forall (v_l, v_m, t_{lm}) \in D_m$$

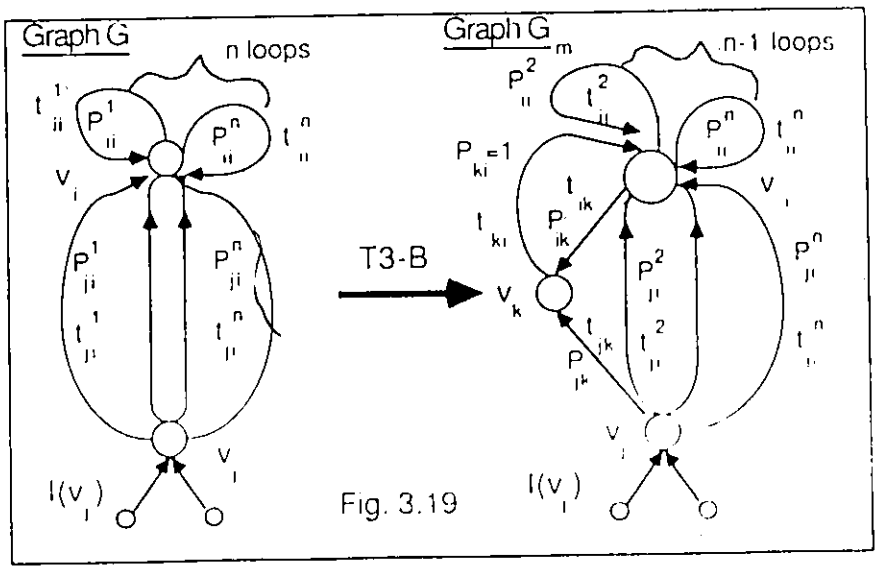
and t_{ik}, t_{jk} and t_{ki} are given as follows:

$t_{ik} \leftarrow t_{i-}$ for any $v_- \neq v_i \wedge v_- \neq v_j \wedge v_- \in O(v_i)$
$t_{jk} \leftarrow t_{j-}$ for any $v_- \neq v_i \wedge v_- \neq v_j \wedge v_- \in O(v_j)$
$t_{ki} \leftarrow (t_{ii} - t_{ik}) \vee (t_{ji} - t_{jk})$

Apply T3-A until all self-loops at v_i and v_j are removed since one application of T3-A removes just one self-loop at v_i or v_j. t_{ki} must be non-negative.

Transformation T3-B:

Objective: Select a self-loop either at v_i or v_j and construct a junction at $v_k \neq v_i \wedge v_k \neq v_j$ by removing the selected self-loop.



The graph G_m from G can be obtained as follows:

$$V_m \leftarrow V + v_k$$

$$D_m \leftarrow D - (v_i, v_i, t_{ii}^1) - (v_j, v_i, t_{ji}^1) + (v_i, v_k, t_{ik}) + (v_k, v_i, t_{ki}) + (v_j, v_k, t_{jk})$$

$$p_m(v_l, v_m, t_{lm}) \leftarrow \begin{cases} p(v_j, v_i, t_{ji}^1) & \text{if } l = j \wedge m = k \\ p(v_i, v_i, t_{ii}^1) & \text{if } l = i \wedge m = k \\ 1 & \text{if } l = k \wedge m = i \\ p(v_l, v_m, t_{lm}) & \text{otherwise} \end{cases}$$

$$\forall (v_l, v_m, t_{lm}) \in D_m$$

and t_{ik}, t_{jk} and t_{ki} are given as follows:

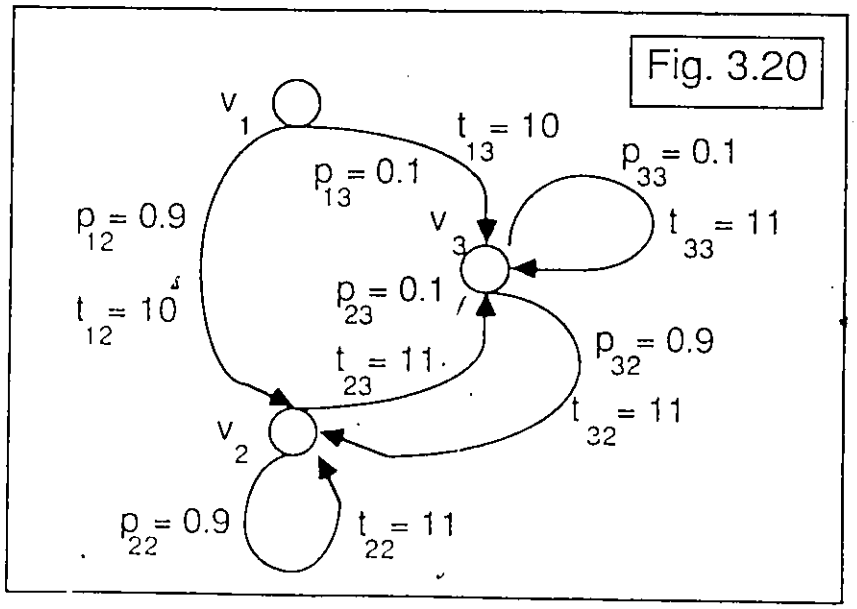
If $t_{ji}^1 > t_{ii}^1$ then choose t_{ik} from $[0, t_{ii}^1]$
$t_{jk} \leftarrow t_{ik} + t_{ji}^1 - t_{ii}^1$
else choose t_{jk} from $[0, t_{ji}^1]$
$t_{ik} \leftarrow t_{jk} + t_{ii}^1 - t_{ji}^1$

Endif

$$t_{ki} \leftarrow (t_{ii}^1 - t_{ik}) \vee (t_{ji}^1 - t_{jk})$$

We illustrate transformations T3-A and T3-B with the help of the following example.

Example 3.5



Referring to Fig.3.20, we obtain:

$O(v_1) = O(v_2) = O(v_3) = \{v_2, v_3\}$

$p_{1,2} = p_{2,2} = p_{3,2} = 0.9$

$p_{1,3} = p_{2,3} = p_{3,3} = 0.1$

Therefore, CVF $= \{v_1, v_2, v_3\}$

Therefore, VFS $= \{\{v_1, v_2, v_3\}\}$

Case i. Let $v_i = v_1 \wedge v_j = v_2$(Fig.3.20)

Applying Transformation T3-A (i.e., remove self-loop at v_2), we obtain:

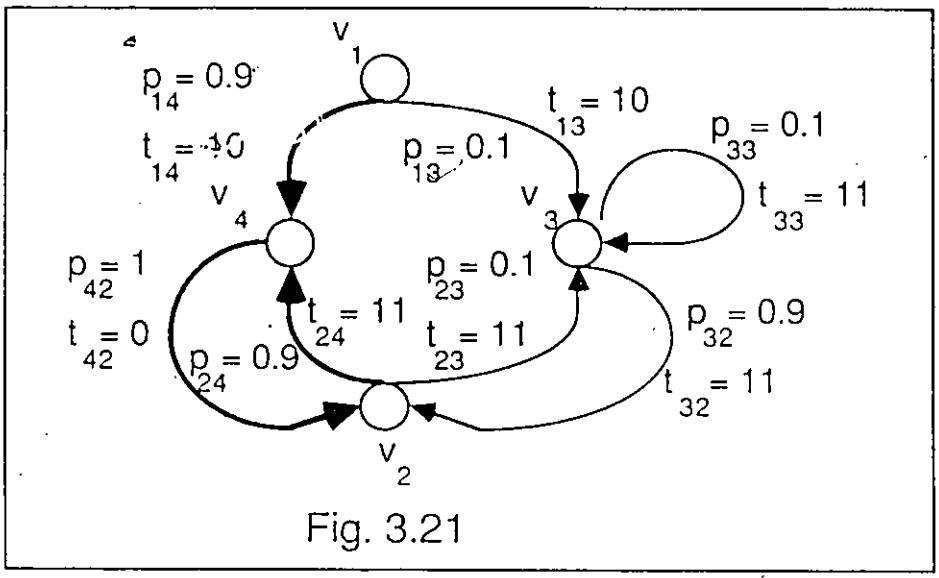


Fig. 3.21

$t_{1,4} \leftarrow t_{1,3} = 10$

$t_{2,4} \leftarrow t_{2,3} = 11$

$t_{4,2} \leftarrow t_{2,2} + t_{2,4} = 11 + 11 = 0$

Note: Directed edges $(v_1, v_4, t_{1,4})$,$(v_4, v_2, t_{4,2})$ and $(v_2, v_4, t_{2,4})$ are shown by bold arcs to demonstrate the construction of junction from self-loop.

Case ii. Let $v_i = v_3 \wedge v_j = v_2$(Fig.3.20)

(a) Select self-loop at v_3

Now applying transformation T3-B(i.e., remove self-loop at v_3), we obtain :

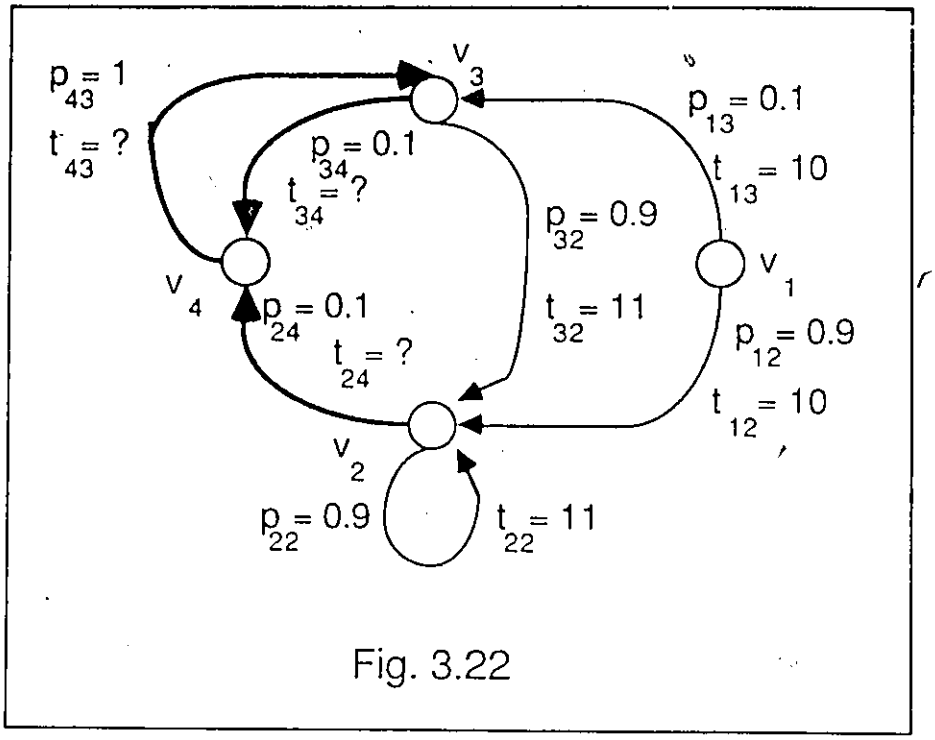


Fig. 3.22

Calculation of $t_{3,4}$, $t_{2,4}$ and $t_{4,3}$ is as follows:

$t_{2,3} = 11$, $t_{3,3} = 11$

Choose $t_{2,4}$ from $[0, t_{2,3}] = [0, 11]$

Let $t_{2,4} = 5$

$\Rightarrow t_{3,4} = t_{2,4} + t_{3,3} - t_{2,3} = 5 + 11 - 11 = 5$

Hence, $t_{4,3} = t_{3,3} - t_{3,4} = 11 - 5 = 6$

Also, check that $t_{4,3} = t_{2,3} - t_{2,4} = 11 - 5 = 6$

(b) Select self-loop at v_2

Now applying transformation T3-B(i.e., remove self-loop at v_2), we obtain :

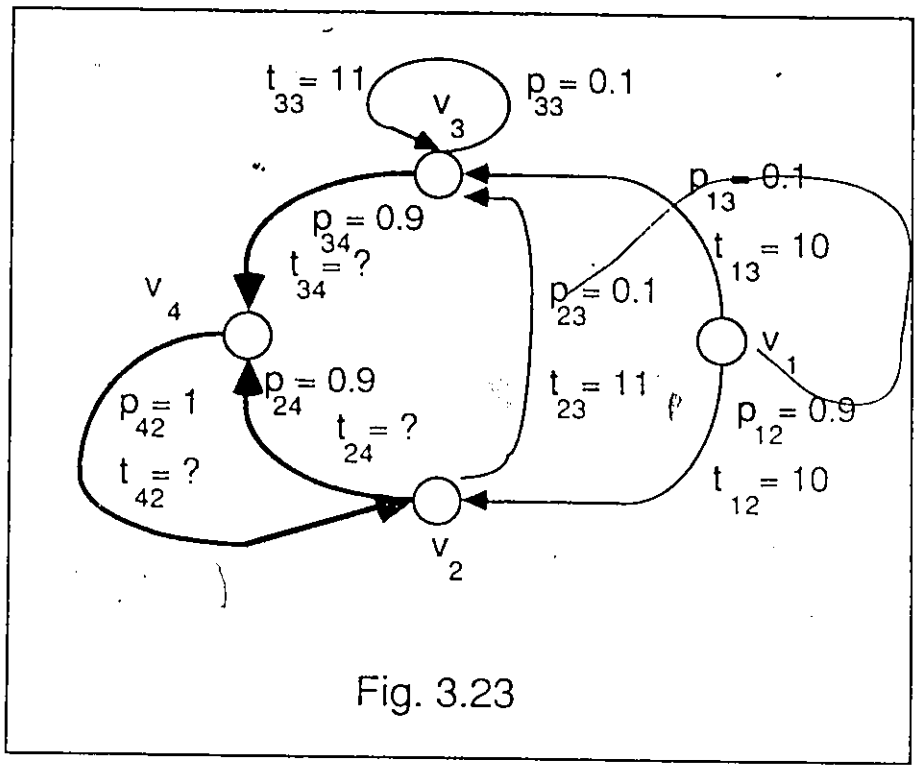


Fig. 3.23

Calculation of $t_{3,4}$, $t_{2,4}$ and $t_{4,2}$ is as follows:

$t_{3,2} = 11$, $t_{2,2} = 11$

Choose $t_{3,4}$ from $[0, t_{3,2}] = [0, 11]$

Let $t_{3,4} = 2$

$\Longrightarrow t_{2,4} = t_{3,4} + t_{2,2} - t_{3,2} = 2 + 11 - 11 = 2$

Hence, $t_{4,2} = t_{2,2} - t_{2,4} = 11 - 2 = 9$

Also, check that $t_{4,2} = t_{3,2} - t_{3,4} = 11 - 2 = 9$

3.3 Summary

We presented a set of reduction rules, namely, vertex reduction, junction removal, decision removal, multi-in multi-out vertex removal, merging parallel self-loops, merging parallel edges, self-loop removal, and vertex folding. These rules help reduce the size of probabilistic transition graphs obtained from the analysis of timed Petri net models.

We developed a more general algorithm for vertex folding than those presently existing. It is based on the law of conservation of transition time. To the best of our knowledge, this is the first general vertex folding algorithm.

All these reduction rules are incorporated into a graph reduction software package which can be used in automated performance prediction. Graph reduction software tools such as these are important and form an integral part of automated protocol performance prediction[RUD 83].

These reduction rules may be applied in a variety of orders as required to eliminate certain transitions or vertices and preserve those deemed to be important. We demonstrate the application of these rules to a version of stop-and-wait protocol in the next chapter.

Chapter 4

Application of timed Petri nets to performance analysis

Analyses of timed Petri nets were presented in Chapter 2. Graph reduction rules to reduce the size of the state transition graphs were introduced in Chapter 3. Based on the formalisms developed in Chapters 2 and 3, we illustrate in this chapter the application of timed Petri nets to performance analysis.

For the purpose of illustration, we consider two examples. The first example deals with the analysis of a stop-and-wait protocol, modeled as a D-timed Petri net. The second example deals with the analysis of an office copying system, modeled as an M-timed Petri net.

Example 1 : Stop-and-wait protocol

Protocols in which the sender sends one frame and then waits for an acknowledgement before proceeding are called *stop-and-wait* protocols[TAN 81]. The communication channel in this example is assumed to be noisy — frames may be either damaged or lost completely. Data traffic is simplex — going only from the sender to the receiver; frames however travel in both directions, reverse traffic consisting of acknowledgement frames. Hence, the communication channel is assumed to be capable of bidirectional information transfer. A half-duplex physical channel would suffice for this example since we have assumed strict alteration of flow - first the sender sends a (DATA) frame, the receiver responds with a dummy (ACKnowledgement) frame; the sender then sends another frame, the receiver responds again and so on.

Due to noise in communication channel, if DATA or ACK is lost or damaged, the sender will timeout and send the frame again. The timeout interval, $t_0 > t_d + t_p + t_a$, where,

- t_d stands for time required for DATA to reach the receiver

- t_p stands for processing time required for receiver to process the DATA
- t_a stands for time required for ACK to reach the sender

The D-timed Petri net shown in Fig.4.1 is a model of the above described stop-and-wait protocol.

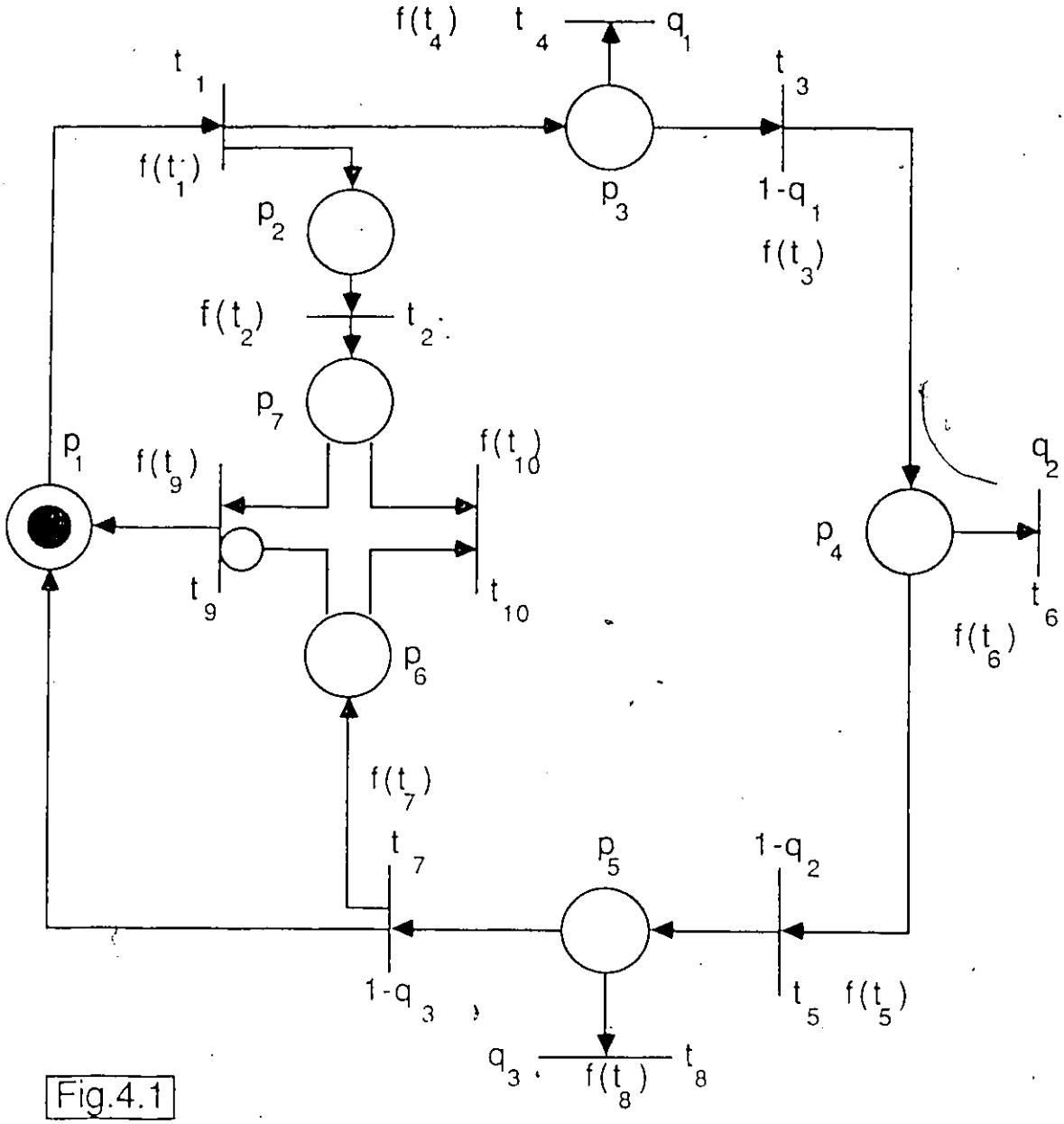


Fig.4.1

The interpretation of Petri net graph(Fig. 4.1) is as follows:

p_1	:	Sender	t_1	:	Send MSG
p_2	:	Timer	t_2	:	Start timer
p_3	:	Message(MSG) ready	t_3	:	Transmit(XMIT) MSG
p_4	:	Receiver	t_4	:	MSG lost
p_5	:	ACK ready	t_5	:	MSG received and Send ACK
p_6	:	ACK arrived	t_6	:	MSG error
p_7	:	End-of-timeout	t_7	:	XMIT ACK
q_1	:	Probability of losing MSG	t_8	:	ACK lost
q_2	:	Probability of MSG being damaged	t_9	:	Retransmit MSG
			t_{10}	:	Kill timeout token
q_3	:	Probability of losing ACK	$f(t)$	:	Firing time of transition t

Sender (place p_1) sends messages to Receiver (place p_4), and for each received message, p_4 confirms by sending an acknowledgement to p_1. Hence, the communication between p_1 and p_4 takes place through the loop $(p_1, t_1, p_3, t_3, p_4, t_5, p_5, t_7, p_1)$. Communication channel being noisy, there are non-zero probabilities — q_1,q_2 and q_3 respectively of — losing messages, messages being damaged and losing acknowledgements respectively.

Free-choice place p_3 and transition t_4 model the lost messages. The probability that the transition t_4 would be selected for firing is q_1 and $(1 - q_1)$ is the probability of selecting the transition t_3 for firing.

Free-choice place p_4 and transition t_6 model the damaged messages. The probability that the transition t_6 would be selected for firing is q_2 and $(1 - q_2)$ is the probability of selecting the transition t_5 for firing.

Free-choice place p_5 and transition t_8 model the lost acknowledgements. The probability that the transition t_8 would be selected for firing is q_3 and $(1 - q_3)$ is the probability of selecting the transition t_7 for firing.

A *timeout* mechanism is used by the sender to recover either lost messages/ acknowledgements or damaged messages. It is designed the following way.

When firing of transition t_1(Send MSG) is completed, a token(MSG) is deposited in place p_3 and another token(timeout) is also deposited in place p_2 at the same time. A token in p_2 immediately starts firing transition t_2(Start timer). The firing time of t_2 is large enough to permit the transfer of message and an acknowledgement. If neither message is lost nor damaged, and acknowledgement is not lost, i.e., transitions t_3,t_5 and t_7 are selected for firing, then a token in place p_1 and also another token in place p_6 are deposited at the same time prior to depositing a token in place p_7.

Token in place p_6 disables transition t_9 since (p_6, t_9) is an inhibitor arc. Hence, the message will not be retransmitted after successful transfer. It is characterized

by the cycle -- $(p_1, t_1, p_3, t_3, p_4, t_5, p_5, t_7, p_1)$. Now that there is a token in place p_1, the exchange of new message between p_1 and p_4 could be started; token in place p_6 will however continue to stay put till the end of timeout period. After the end of timeout period, that is, after the completion of firing t_2, a token will be deposited in place p_7 -- thus starting the firing of transition t_{10} and removing the timeout token from the system.

If either message is lost or damaged, or acknowledgement is lost, then the token will never be deposited in place p_6. After the end of timeout period, a token will be deposited in place p_7 -- thus starting the firing of transition t_9. After the completion of firing transition t_9, a token will be deposited in place p_1 -- indicating the message must be retransmitted.

There is also a non-zero probability that an acknowledgement might arrive at the sender's end -- but damaged. This is not taken into account in this example but can be easily modeled with an additional free-choice place.

In real-life situations this possibility might have to be considered. We have ignored it here only to keep the illustration of performance analysis as brief and simple as possible.

If the message is damaged then the receiver does not respond with ACK frame, but instead, discards the message(token). This is modeled by transition t_6. In case the message is not damaged, the receiver responds with ACK frame even if the message is duplicate. This is modeled by transition t_5. It is assumed that the receiver's IMP(Inter Message Processor) does not deliver to its host both the duplicate and damaged messages. In this example, we are concerned whether the receiver responds with ACK frame or not.

The dynamic behaviour of this model(Fig. 4.1) is analysed using the formalism described in Chapter 2. Two cases -- $(f(t_2), f(t_3), q_1, q_2, q_3) = (20, 10, 0.1, 0.05, 0.1)$ and $(30, 20, 0.2, 0.1, 0.1)$, corresponding to transfer of messages of two different lengths are considered. The transition graphs characterizing the dynamic behaviour of the model for two cases are shown in Figs. 4.2 and 4.3. The set of reachable states is given in Tables 4.2 and 4.3. Table 4.1 gives the firing times and free-choice probabilities and initial marking which are common to both cases.

In Figs. 4.2 and 4.3, each directed edge is labeled with a tuple (p, h_i) where p is the probability and h_i is the transition time as given in Tables 4.2 and 4.3. The path representing error-free or successful transfer of messages is shown by bold directed edges.

In Chapter 3, we presented the reduction rules to reduce the transition graphs. We now apply the reduction rules to both Figs. 4.2 and 4.3. Complete reduction process is shown next.

Firing times:

$$f(t_1) = 1 \quad f(t_4) = 0 \quad f(t_5) = 2 \quad f(t_6) = 0$$
$$f(t_7) = 5 \quad f(t_8) = 0 \quad f(t_9) = 0 \quad f(t_{10}) = 0$$

Free-choice probabilities:

$$c(t_1) = 1.0 \quad c(t_2) = 1.0 \quad c(t_7) = 0.9$$
$$c(t_8) = 0.1 \quad c(t_9) = 1.0 \quad c(t_{10}) = 1.0$$

Initial marking:

$$\mu_0 = (1, 0, 0, 0, 0, 0, 0)$$

Table 4.1: Firing times,free-choice probabilities and initial marking which are common to both cases -- 1 and 2

s_i	μ_i	v_i	h_i	μ_{ij}	e_k	s_j	p
1	0000000	10000000000	1	0110000	01010000000	2	0.10
					01100000000	3	0.90
2	0000000	01010000000	0	0000000	00000000000	4	1.00
3	0000000	01100000000	10	0001000	00000100000	5	0.05
					00001000000	6	0.95
4	0000000	01000000000	20	0000001	00000000010	7	1.00
5	0000000	01000010000	0	0000000	00000000000	8	1.00
6	0000000	01001000000	2	0000100	00000001000	9	0.10
					00000010000	10	0.90
7	0000000	00000000010	0	1000000	10000000000	1	1.00
8	0000000	01000000000	10	0000001	00000000010	7	1.00
9	0000000	01000001000	0	0000000	00000000000	11	1.00
10	0000000	01000010000	5	1000010	10000000000	12	1.00
11	0000000	01000000000	8	0000001	00000000010	7	1.00
12	0000010	11000000000	1	0110010	01010000000	13	0.10
					01100000000	14	0.90
13	0000010	02010000000	0	0000010	00000000000	15	1.00
14	0000010	02100000000	2	0000011	00000000001	16	1.00
15	0000010	02000000000	2	0000011	00000000001	17	1.00
16	0000000	01100000001	0	0000000	00000000000	18	1.00
17	0000000	01000000001	0	0000000	00000000000	19	1.00
18	0000000	01100000000	8	0001000	00000100000	5	0.05
					00001000000	6	0.95
19	0000000	01000000000	18	0000001	00000000010	7	1.00

Table 4.2 State transitions for $f(t_2) = 20$, $f(t_3) = 10$

$q_1 = c(t_4) = 0.1$, $q_2 = c(t_6) = 0.05$, $q_3 = c(t_8) = 0.1$

s_i	μ_i	v_i	h_i	μ_{ij}	e_k	s_j	p
1	0 0 0 0 0 0 0	1 0 0 0 0 0 0 0 0 0 0	1	0 1 1 0 0 0 0	0 1 0 1 0 0 0 0 0 0	2	0.20
					0 1 1 0 0 0 0 0 0 0	3	0.80
2	0 0 0 0 0 0 0	0 1 0 1 0 0 0 0 0 0	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	4	1.00
3	0 0 0 0 0 0 0	0 1 1 0 0 0 0 0 0 0	20	0 0 0 1 0 0 0	0 0 0 0 0 1 0 0 0 0	5	0.10
					0 0 0 0 1 0 0 0 0 0	6	0.90
4	0 0 0 0 0 0 0	0 1 0 0 0 0 0 0 0 0	30	0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 1 0	7	1.00
5	0 0 0 0 0 0 0	0 1 0 0 0 1 0 0 0 0	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	8	1.00
6	0 0 0 0 0 0 0	0 1 0 0 1 0 0 0 0 0	2	0 0 0 0 1 0 0	0 0 0 0 0 0 0 1 0 0	9	0.10
					0 0 0 0 0 0 1 0 0 0	10	0.90
7	0 0 0 0 0 0 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	1.00
8	0 0 0 0 0 0 0	0 1 0 0 0 0 0 0 0 0	10	0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 1 0	7	1.00
9	0 0 0 0 0 0 0	0 1 0 0 0 0 0 1 0 0	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	11	1.00
10	0 0 0 0 0 0 0	0 1 0 0 0 0 1 0 0 0	5	1 0 0 0 0 1 0	1 0 0 0 0 0 0 0 0 0	12	1.00
11	0 0 0 0 0 0 0	0 1 0 0 0 0 0 0 0 0	8	0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 1 0	7	1.00
12	0 0 0 0 0 1 0	1 1 0 0 0 0 0 0 0 0	1	0 1 1 0 0 1 0	0 1 0 1 0 0 0 0 0 0	13	0.20
					0 1 1 0 0 0 0 0 0 0	14	0.80
13	0 0 0 0 0 1 0	0 2 0 1 0 0 0 0 0 0	0	0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 0	15	1.00
14	0 0 0 0 0 1 0	0 2 1 0 0 0 0 0 0 0	2	0 0 0 0 0 1 1	0 0 0 0 0 0 0 0 0 1	16	1.00
15	0 0 0 0 0 1 0	0 2 0 0 0 0 0 0 0 0	2	0 0 0 0 0 1 1	0 0 0 0 0 0 0 0 0 1	17	1.00
16	0 0 0 0 0 0 0	0 1 1 0 0 0 0 0 0 1	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	18	1.00
17	0 0 0 0 0 0 0	0 1 0 0 0 0 0 0 0 1	0	0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	19	1.00
18	0 0 0 0 0 0 0	0 1 1 0 0 0 0 0 0 0	18	0 0 0 1 0 0 0	0 0 0 0 0 1 0 0 0 0	5	0.10
					0 0 0 0 1 0 0 0 0 0	6	0.90
19	0 0 0 0 0 0 0	0 1 0 0 0 0 0 0 0 0	28	0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 1 0	7	1.00

Table 4.3 State transitions for $f(t_2) = 30$, $f(t_3) = 20$

$q_1 = c(t_4) = 0.2$, $q_2 = c(t_6) = 0.1$, $q_3 = c(t_8) = 0.1$

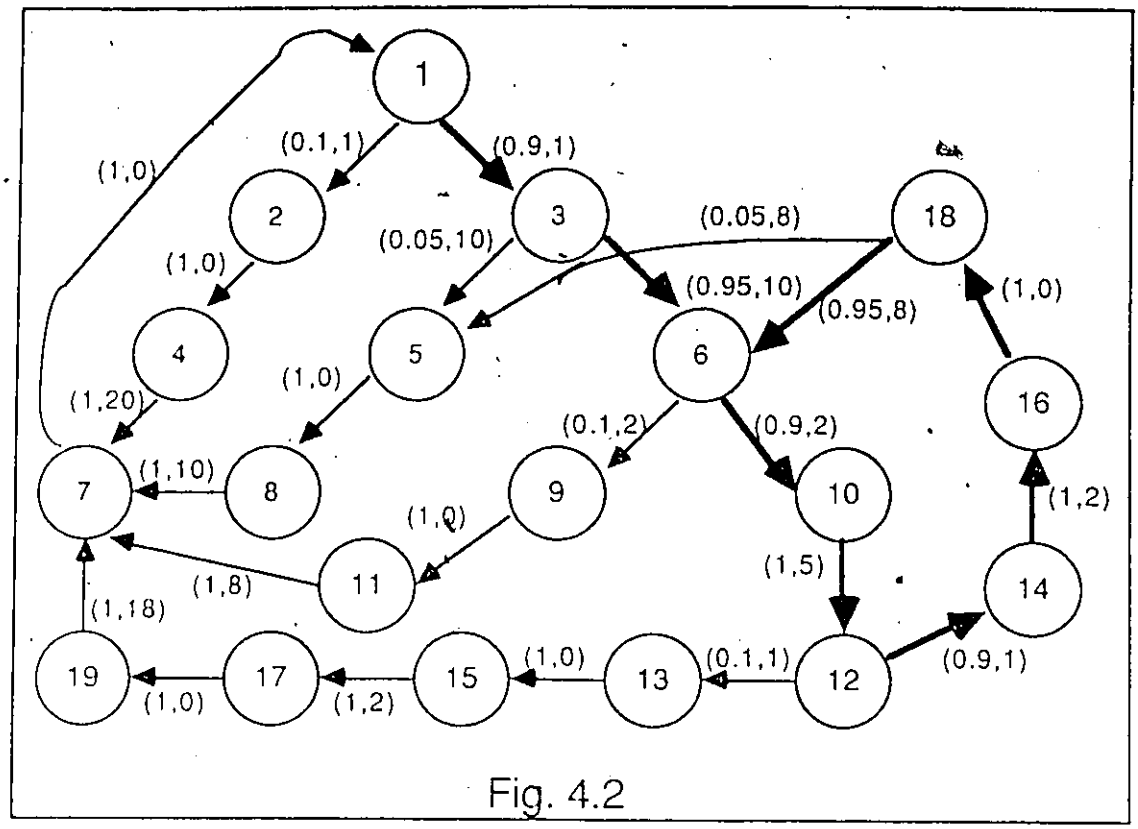

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
(1,0)
(0.1,1)
(0.9,1)
(1,0)
(0.05,10)
(0.05,8)
(0.95,10)
(0.95,8)
(1,0)
(1,20)
(1,0)
(0.1,2)
(0.9,2)
(1,10)
(1,Q)
(1,2)
(1,8)
(1,5)
(1,18)
(1,0)
(1,2)
(1,0)
(0.1,1)
(0.9,1)

Fig. 4.2

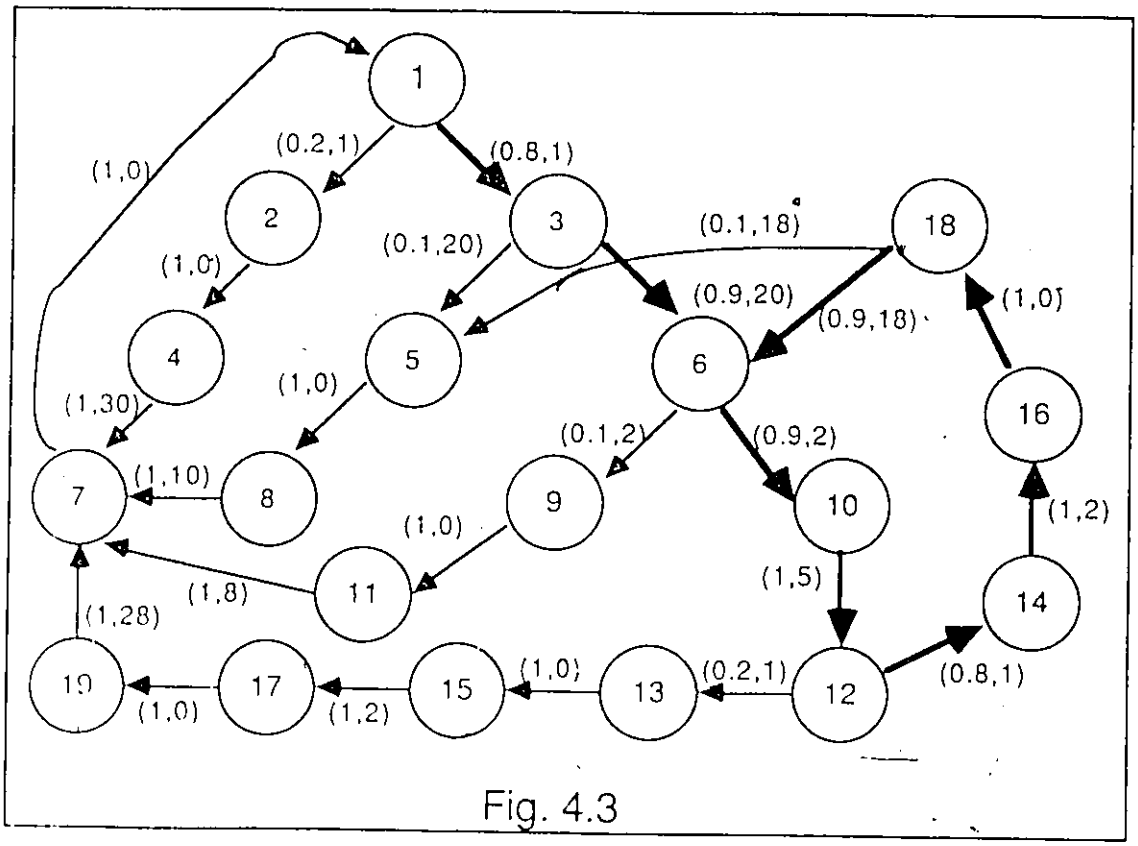

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
(1,0)
(0.2,1)
(0.8,1)
(1,0)
(0.1,20)
(0.1,18)
(0.9,20)
(0.9,18)
(1,0)
(1,30)
(1,0)
(0.1,2)
(0.9,2)
(1,10)
(1,0)
(1,2)
(1,8)
(1,5)
(1,28)
(1,0)
(1,2)
(1,0)
(0.2,1)
(0.8,1)

Fig. 4.3

Case 1. Reduction of Fig. 4.2

Apply vertex reduction rule to vertices — 2 , 4 , 8 , 9 , 10 , 11 , 13 , 14 , 15 , 16 , 17 and 19 in Fig. 4.2.

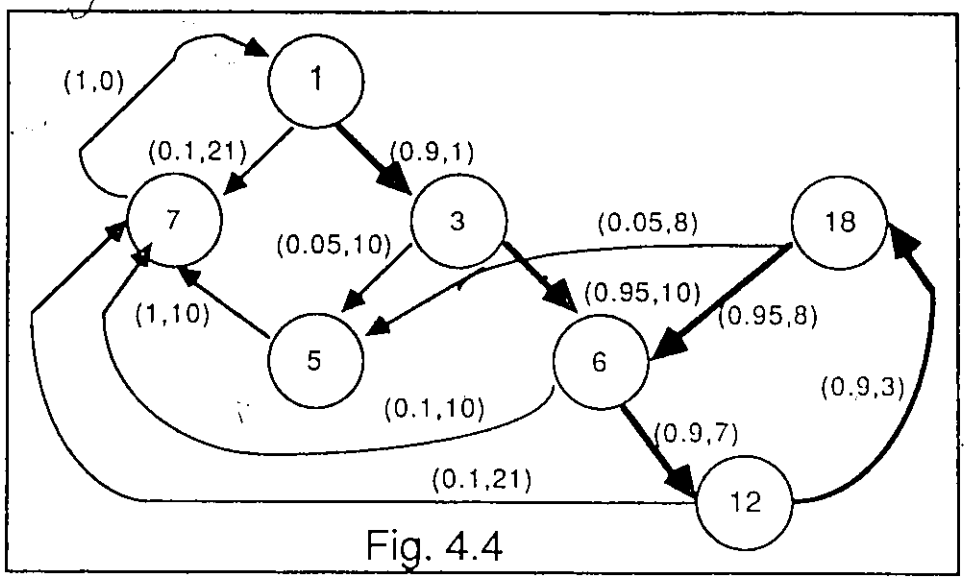


Fig. 4.4

Referring to Fig. 4.4, we find that (v_i means vertex i)

$$O(v_{18}) = O(v_3) = \{v_5, v_6\}$$
$$p_{3,5} = p_{18,5} = 0.05 \quad \wedge \quad p_{3,6} = p_{18,6} = 0.95$$

The vertex folding rule therefore is enabled and we apply it to vertices - - 3 and 18. Let $v_i = v_3 \wedge v_j = v_{18}$. Then, $t_{i,5} = 10 \neq t_{j,5} = 8$, $I(v_j) = \{v_{12}\} \neq \emptyset$, $I(v_i) = \{v_1\} \neq \emptyset$, $v_j \notin I(v_j)$ and $v_i \notin I(v_i)$. From Procedure UNEQUAL in Chapter 3 we find that, Transformation T1 can be applied. That is, delete vertex 18 and directed edges $(v_{18}, v_5, 8)$, $(v_{18}, v_6, 8)$ and $(v_{12}, v_{18}, 3)$. Add a new directed edge $(v_{12}, v_3, t_{12,3})$ with $t_{12,3}$ calculated using equation (3.1). That is,

$$\begin{aligned} t_{12,3} &= t_{12,18} + t_{18,5} - t_{3,5} \\ &= 3 + 8 - 10 = 1 \end{aligned}$$

Hence, the result is Fig. 4.5.

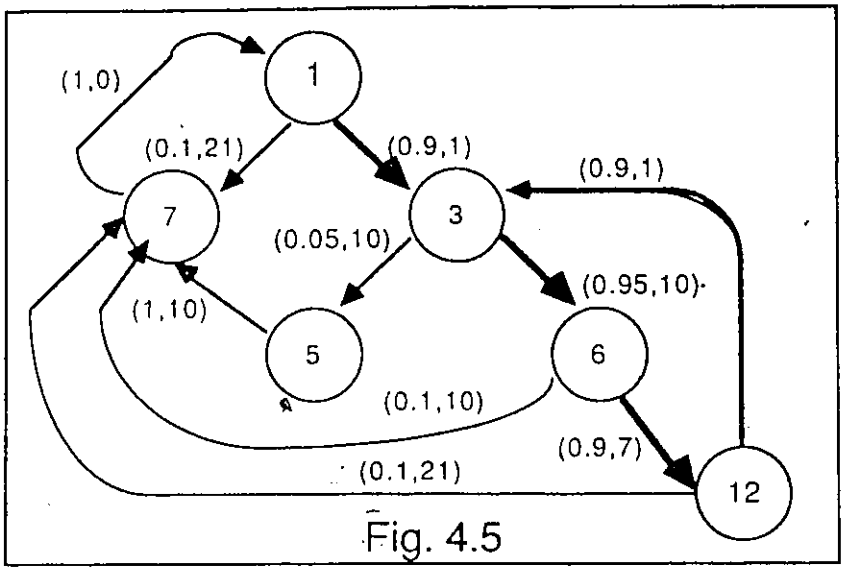


Fig. 4.5

Apply vertex reduction rule to vertex 5 in Fig. 4.5.

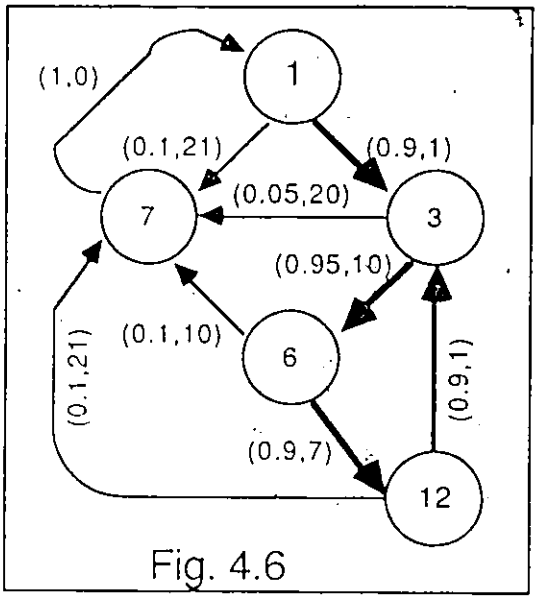


Fig. 4.6

Apply vertex folding rule to vertices — 1 and 12 in Fig. 4.6.

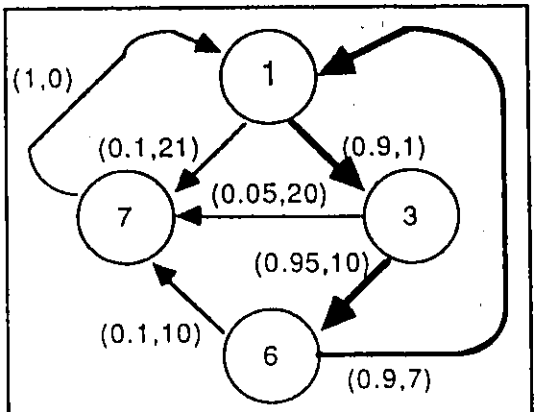


Fig. 4.7

Apply junction removal rule to vertex 7 in Fig. 4.7.

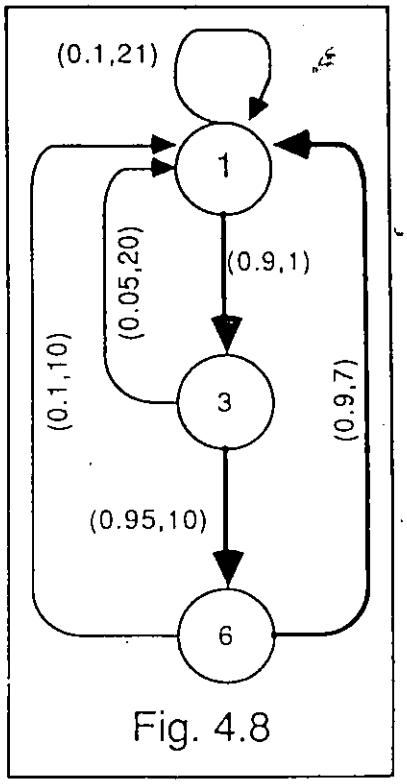


Fig. 4.8

Apply decision removal rule to vertex 6 in Fig. 4.8.

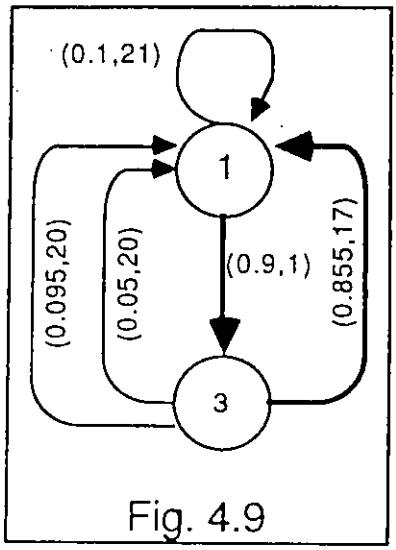


Fig. 4.9

Apply decision removal rule to vertex 3 in Fig. 4.9.

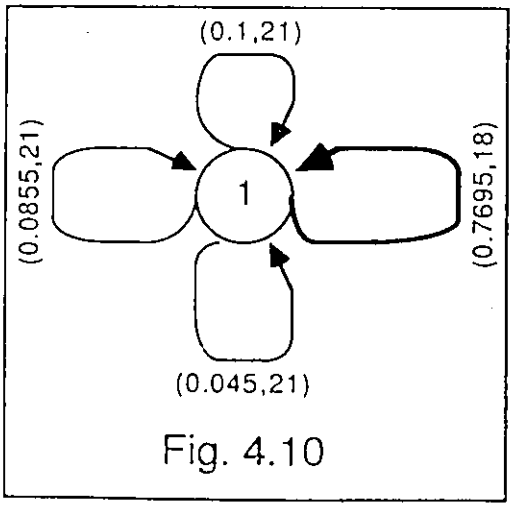


Fig. 4.10

Merge all unsuccessful parallel self-loops in Fig. 4.10.

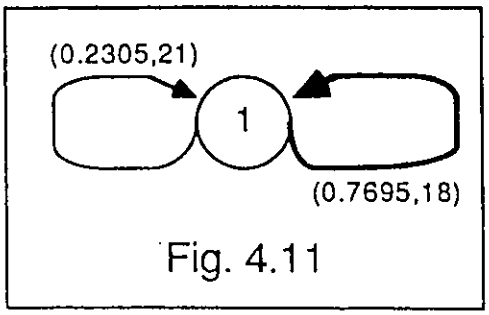


Fig. 4.11

Case 2. Reduction of Fig. 4.3

In a similar manner as in Case 1, applying reduction rules to Fig. 4.3, we obtain (intermediate steps are not shown here) :

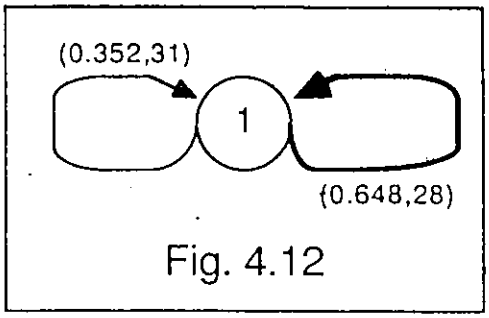


Fig. 4.12

Let h_1 denote the time required to traverse the unsuccessful path and h_2 denote the time required to traverse the error-free path. We find from Fig.4.11, $h_1 = 21$ and $h_2 = 18$ for Case 1. For Case 2, from Fig.4.12 we find, $h_1 = 31$ and $h_2 = 28$.

Definition 4.1

Efficiency of the protocol , ε, is equal to the time spent on correct transfers to the total time elapsed.

For stop-and-wait protocol(Fig.4.1), ε is given as follows:

$$\varepsilon = \frac{(1-q_1)(1-q_2)(1-q_3)h_2}{[q_1 + (1-q_1)q_2 + (1-q_1)(1-q_2)q_3]h_1 + [(1-q_1)(1-q_2)(1-q_3)]h_2}$$

For Case 1, $\varepsilon = \frac{0.7695 \times 18}{0.2035 \times 21 + 0.7695 \times 18} = \frac{13.851}{18.6915} = 0.741032$

For Case 2, $\varepsilon = \frac{0.648 \times 28}{0.352 \times 31 + 0.648 \times 28} = \frac{18.144}{29.056} = 0.6244$

Calculation of optimal message length

Now, we attempt to calculate the optimal message length for which the *performance* or *throughput* is maximum.

The message is composed of — data block and service block. Data block is the actual message that the sender would like to send it to the receiver. Service block is the overhead involved in making the complete communicating system operational. Service block consists of various control bits such as — synchronization, identification etc., and is of fixed length. It seems — to obtain the better *throughput*, data block (hence message) must be long. Longer messages, however, are more prone to errors.

To simplify the analysis, we have assumed as in [ZUB 85a], q_1 and q_2 — probabilities of — losing message and message being damaged respectively, to be a linear functions of message length z. It is also assumed that the message transfer time is a linear function of z. Hence,

$$\begin{aligned} q_1 &= q_{11} + q_{01}z \\ q_2 &= q_{21} + q_{02}z \\ h_1 &= a_1 + a_0 z \\ h_2 &= a_2 + a_0 z \end{aligned}$$

The *performance* of the protocol , η , is defined as the ratio of time spent on correct data transfers to the total time elapsed.

For stop-and-wait protocol(Fig.4.1), η is given as follows:

$$\eta = \frac{\epsilon a_0 z}{h_2}$$

$$\therefore \frac{(1-q_1)(1-q_2)(1-q_3)h_2 a_0 z}{\{[q_1 + (1-q_1)q_2 + (1-q_1)(1-q_2)q_3]h_1 + [(1-q_1)(1-q_2)(1-q_3)]h_2\}h_2}$$

$$\frac{(1-q_1)(1-q_2)(1-q_3)a_0 z}{[q_1 + (1-q_1)q_2 + (1-q_1)(1-q_2)q_3]h_1 + [(1-q_1)(1-q_2)(1-q_3)]h_2}$$

Simplify the denominator:

$$
\begin{aligned}
& [q_1 + (1-q_1)q_2 + (1-q_1)(1-q_2)q_3]h_1 + (1-q_1)(1-q_2)(1-q_3)h_2 \\
= \; & [q_1 + (1-q_1)q_2]h_1 + (1-q_1)(1-q_2)[q_3 h_1 + (1-q_3)h_2] \\
= \; & [q_1 + (1-q_1)q_2]h_1 + (1-q_1)(1-q_2)[q_3 h_1 - q_3 h_2 + h_2] \\
= \; & [q_1 + (1-q_1)q_2]h_1 + (1-q_1)(1-q_2)[q_3(a_1 - a_2) + h_2] \\
& \text{since } h_1 - h_2 = a_1 - a_2 \\
= \; & [q_1 + (1-q_1)q_2]h_1 + (1-q_1)(1-q_2)h_2 + (1-q_1)(1-q_2)q_3(a_1 - a_2) \\
= \; & q_1 h_1 + (1-q_1)[q_2 h_1 + (1-q_2)h_2] + (1-q_1)(1-q_2)q_3(a_1 - a_2) \\
= \; & q_1 h_1 + (1-q_1)[q_2(h_1 - h_2) + h_2] + (1-q_1)(1-q_2)q_3(a_1 - a_2) \\
= \; & q_1 h_1 + (1-q_1)h_2 + (1-q_1)q_2(a_1 - a_2) + (1-q_1)(1-q_2)q_3(a_1 - a_2) \\
= \; & q_1(h_1 - h_2) + h_2 + (1-q_1)q_2(a_1 - a_2) + (1-q_1)(1-q_2)q_3(a_1 - a_2) \\
= \; & q_1(a_1 - a_2) + h_2 + (a_1 - a_2)(1-q_1)[q_2 + (1-q_2)q_3] \\
= \; & a_1 q_1 - a_2 q_1 + h_2 + (a_1 - a_2)(1-q_1)[q_2 + (1-q_2)q_3] \\
= \; & a_1(q_{11} + q_{01}z) - a_2(q_{11} + q_{01}z) + a_2 + a_0 z + (a_1 - a_2)(1-q_1)[q_2 + (1-q_2)q_3] \\
= \; & a_1 q_{11} + (1-q_{11})a_2 + [a_0 + (a_1 - a_2)q_{01}]z + (a_1 - a_2)(1-q_1)[q_2 + (1-q_2)q_3]
\end{aligned}
$$

Rewriting η, we obtain:

$$\eta = \frac{a_0 z(1-q_1)(1-q_2)(1-q_3)}{a_1 q_{11} + (1-q_{11})a_2 + [a_0 + (a_1 - a_2)q_{01}]z + \mathrm{RF}_\eta} \tag{4.1}$$

where, $\mathrm{RF}_\eta = (a_1 - a_2)(1-q_1)[q_2 + (1-q_2)q_3]$.

Now suppose, $q_2 = q_3 = 0$ then we have,

$$\eta = \frac{a_0 z(1-q_1)}{a_1 q_{11} + (1-q_{11})a_2 + [a_0 + (a_1 - a_2)q_{01}]z}$$

This is the expression for performance of simple protocol obtained by Zuberek [ZUB 85a].

Assume $a_0 = 1$. Since, acknowledgements in Example 1 contain no data blocks only service blocks, the acknowledgement transfer time, that is, firing time of transition t_7, is constant. Assume that the error checking and sending acknowledgement, that is, firing time of transition t_5 takes constant time. Hence, we have,

$$f(t_7) = 5 \wedge f(t_5) = 2$$

Assuming the length of the service block in any message is equal to the length of the service block in an acknowledgement, we obtain the message transfer time, that is, firing time of transition t_3 to be

$$f(t_3) = f(t_7) + z = 5 + z$$

In Example 1(Fig. 4.1) the time required for traversing the successful transfer cycle — $(p_1, t_1, p_3, t_3, p_4, t_5, p_5, t_7, p_1)$ is given by

$$\begin{aligned} h_2 &= f(t_1) + f(t_3) + f(t_5) + f(t_7) \\ &= 1 + 5 + z + 2 + 5 \\ &= 13 + z \end{aligned}$$

We have assumed, $h_2 = a_2 + a_0 z$. Therefore,

$$a_2 + a_0 z = 13 + z \Longrightarrow a_2 = 13$$

For error-recovery mechanism to work properly, we must have (Fig. 4.1),

$$\begin{aligned} f(t_2) &> f(t_3) + f(t_5) + f(t_7) \\ &= 5 + z + 2 + 5 \\ &= 12 + z \\ \Rightarrow \quad f(t_2) &= 12 + z + \Delta \qquad \text{where } \Delta > 0 \end{aligned}$$

In Example 1, the amount of time the sender waits before sending a duplicate message is given by

$$\begin{aligned} h_1 &= f(t_1) + f(t_2) \\ &= 1 + 12 + \Delta + z \\ &= 13 + \Delta + z \end{aligned}$$

We have assumed, $h_1 = a_1 + a_0 z$. Therefore,

$$a_1 + a_0 z = 13 + \Delta + z \Longrightarrow a_1 = 13 + \Delta$$

Now for Case 1, we have $f(t_2) = 20 \wedge z = 5$. Therefore,

$$12 + \Delta + z = 12 + \Delta + 5 = 20 \Rightarrow \Delta = 3$$

Therefore, $a_1 = 13 + \Delta = 13 + 3 = 16$.
Assume $q_{01} = 0.01$ and $q_{02} = 0.005$. From Case 1, we have $z = 5$, $q_1 = 0.1$, $q_2 = 0.05$ and $q_3 = 0.1$.

$$\begin{aligned} \text{Since, } q_1 &= q_{11} + q_{01}z \\ \Longrightarrow \quad 0.1 &= q_{11} + 0.05 \\ \Longrightarrow \quad q_{11} &= 0.05 \end{aligned}$$

$$\begin{aligned} \text{Also, } q_2 &= q_{21} + q_{02}z \\ \Longrightarrow \quad 0.05 &= q_{21} + 0.025 \\ \Longrightarrow \quad q_{21} &= 0.025 \end{aligned}$$

Now substituting $a_0 = 1$, $a_1 = 16$, $a_2 = 13$, $q_{01} = 0.01$, $q_{02} = 0.005$, $q_{11} = 0.05$, $q_{21} = 0.025$, $q_1 = 0.05 + 0.01z$, $q_2 = 0.025 + 0.005z$ and $q_3 = 0.1$ in the expression for η i.e., equation (4.1), we get

$$\eta = \frac{0.9z(0.95 - 0.01z)(0.975 - 0.005z)}{D_\eta} \tag{4.2}$$

where,

$$D_\eta = 13.15 + 1.03z + 3(0.95 - 0.01z)[0.025 + 0.005z + 0.1(0.975 - 0.005z)]$$

Fig. 4.13 shows η as a function of z. To maximize η, find $z \geq 0 : d\eta/dz = 0$. That is, find the positive root of the following equation.

$$b_0z^4 + b_1z^3 + b_2z^2 + b_3z + b_4 = 0 \tag{4.3}$$

where

$$\begin{aligned} b_0 &= 1.215E - 8 \\ b_1 &= 9.352E - 5 \\ b_2 &= -1.163E - 2 \\ b_3 &= -3.523E - 1 \\ b_4 &= 1.125E + 1 \end{aligned}$$

Solving equation (4.3), we obtain $z = 20.43$. The maximum value of performance is 0.345 for $z = 20.43$.

Comparing the results (Fig. 4.13) obtained here to those of Zuberek[ZUB 85a], we find quite naturally the performance of the stop-and-wait protocol is deteriorated. Zuberek[ZUB 85a] considers the Petri net model of stop-and-wait protocol in which the effect of losing messages and acknowledgements are lumped together. Losing messages, messages being damaged and losing acknowledgements are modeled explicitly in the present analysis.

We observe that the optimum value of performance is fallen from 0.486[ZUB 85a] to 0.345. The effect on optimum message-length, however, is not substantial.

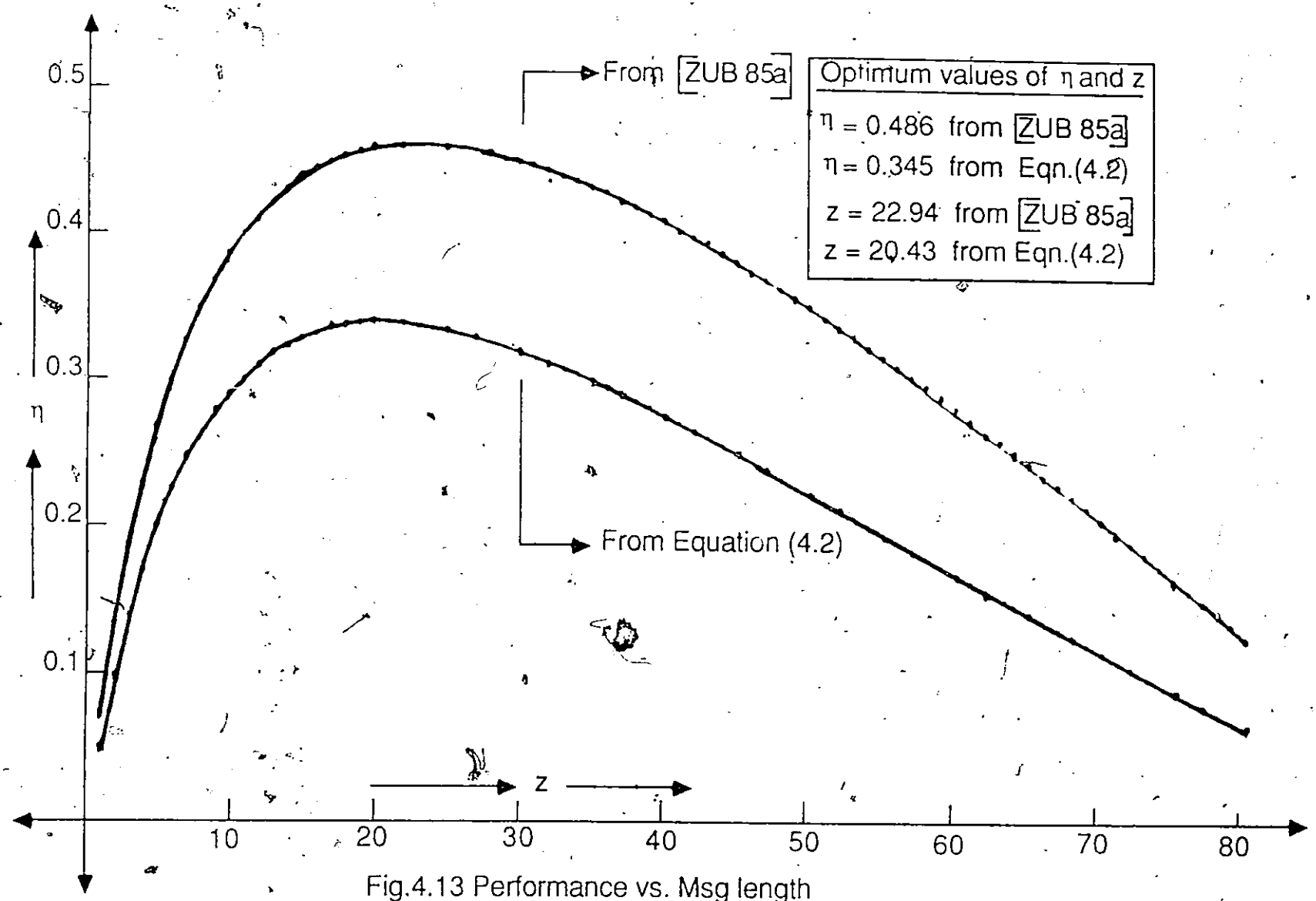


Fig.4.13 Performance vs. Msg length

Example 2 : Office copying system

A simple office copying system is analysed in this example. It is assumed that the office copying system consists of one or more copiers and staplers. It is also assumed that there are two classes of jobs. Class-1 jobs due to managers whereas class-2 jobs due to secretaries.

Secretaries spend a certain amount of time preparing a paper for reproduction (job preparation step). They then go to the copying machine — make copies, collate and staple them, and leave. This process repeats.

Similarly, the managers spend a certain amount of time preparing the documents. They then go to the copying machine — make copies and leave. This process also repeats.

A closed-network model of the office copying system described above with two classes of users and non-preemptive priority scheduling is shown in Fig. 4.14.

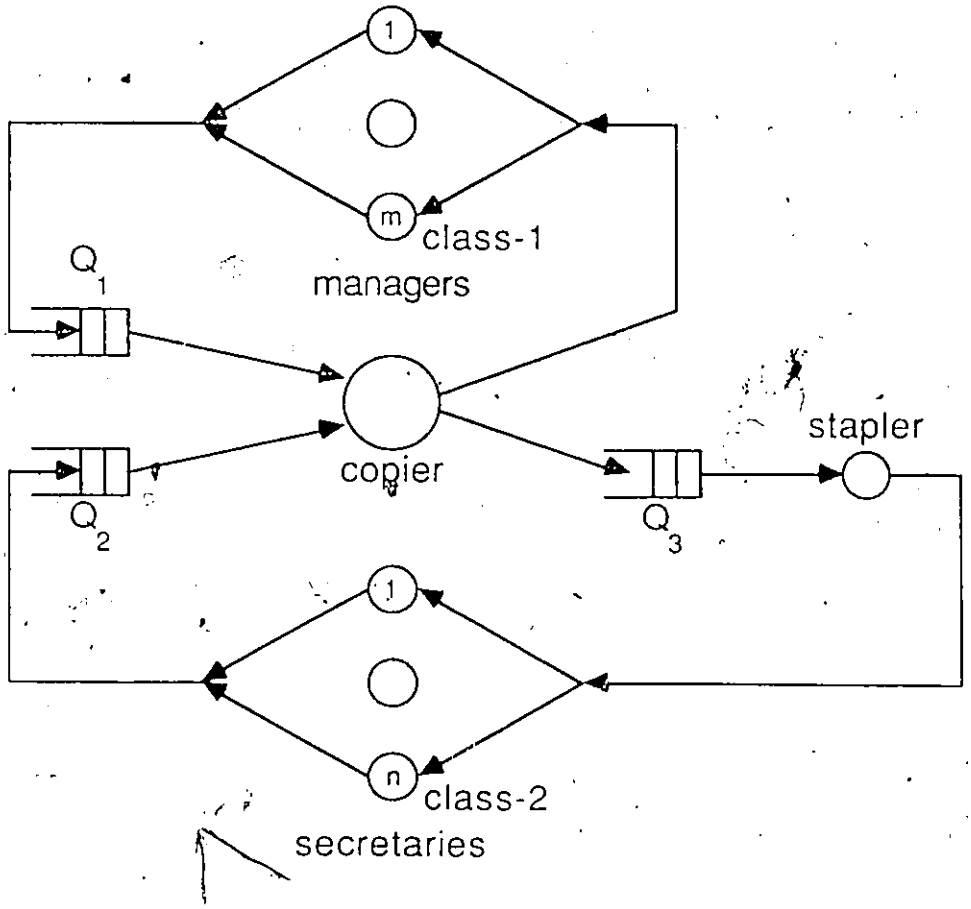
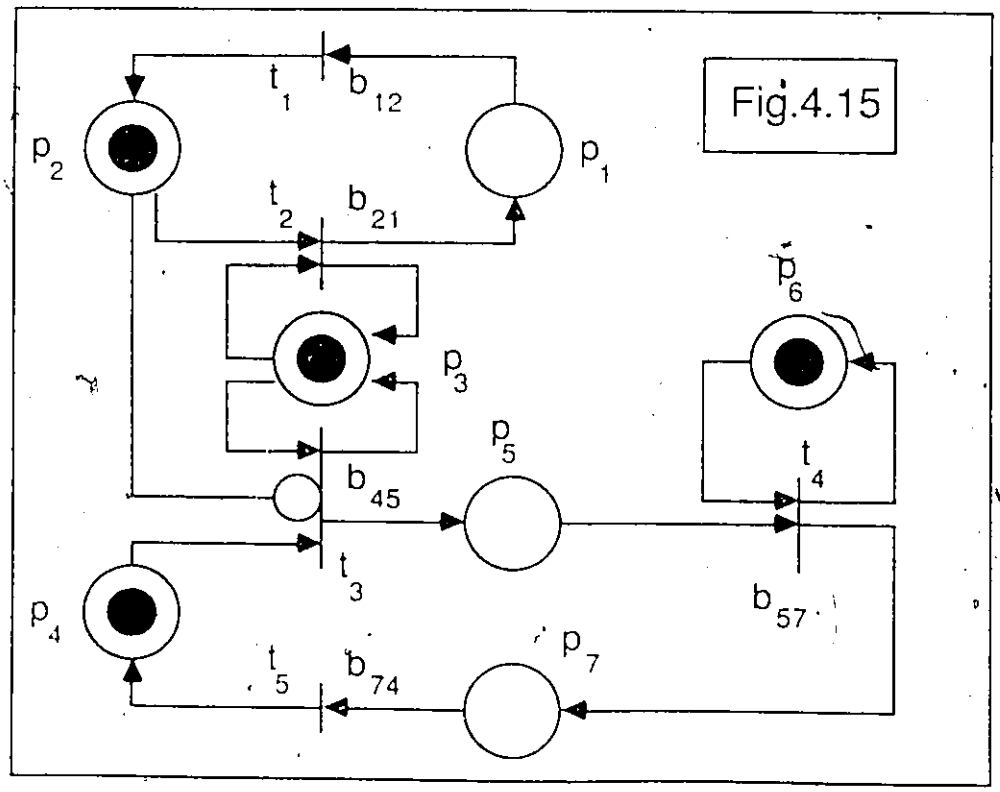


Fig.4.14 Closed-network model of office copying system

We assume that each manager generates exactly one class-1 job at a time and each secretary generates exactly one class-2 jobs at a time. Hence, the number of class-1 and class-2 jobs in the model is equal to the number of managers and secretaries respectively in the office system. From Fig. 4.14, we observe that there are m class-1 jobs and n class-2 jobs. Q_1 and Q_2 are waiting queues of class-1 and class-2 jobs respectively for the copier. Q_3 is the waiting queue of class-2 jobs for the stapler. To simplify the solution, the following assumptions are made:

1. All jobs within the same class are statistically identical.
2. The jobs of the same class are served by FCFS (First-Come-First-Served) discipline.
3. Class-1 jobs have higher priority than class-2 jobs.
4. Service times of copier and stapler, and job preparation times are exponentially distributed.

The M-timed Petri net(Fig. 4.15) models the office copying system described above. The interpretation of Petri net graph is on the next page.



p_1	:	ready for class-1 job preparation	t_1	:	prepare class-1 job
p_2	:	waiting queue Q_1	t_2	:	process class-1 job
p_3	:	copier	t_3	:	process class-2 job
p_4	:	waiting queue Q_2	t_4	:	post process (staple) class-2 job
p_5	:	waiting queue Q_3	t_5	:	prepare class-2 job
p_6	:	stapler			
p_7	:	ready for class-2 job preparation			

b_{12} : firing rate of t_1 or class-1 job preparation rate
b_{21} : firing rate of t_2 or service rate of class-1 jobs
b_{45} : firing rate of t_3 or service rate of class-2 jobs
b_{57} : firing rate of t_4 or post processing service rate of class-2 jobs
b_{74} : firing rate of t_5 or class-2 job preparation rate

The initial number of tokens in places p_1 and p_2 represents the number of class-1 jobs or the number of managers. The initial number of tokens in places p_4, p_5 and p_7 represents the number of class-2 jobs or the number of secretaries. The initial number of tokens in place p_3 and p_6 represents the number of copiers and staplers respectively.

Using the formalism described in Chapter 2, for initial marking $\mu_0 = (0, 1, 1, 1, 0, 1, 0)$ and firing rate $\gamma = (b_{12}, b_{21}, b_{45}, b_{57}, b_{74}) = (0.025, 0.2, 0.1, 0.5, 0.05)$, the set of reachable states, $\mathbf{S(MT)}$ for M-timed Petri net (Fig. 4.15) are given in Table 4.4.

Performance measures such as average throughput rate, average turnaround time and average waiting time in queue can be derived from stationary probabilities of the Petri net states. As in [ZUB 85b], the stationary or equilibrium probabilities $y(s)$ of the states $s \in \mathbf{S(MT)}$ are obtained from the state-transition probabilities $p(s_i, s_j)$ and the average sojourn times $h(s)$ by solving a system of simultaneous linear equations.

$$\sum_{(s_j, s_i) \in D} p(s_j, s_i) y(s_j)/h(s_j) = y(s_i)/h(s_i) \qquad i = 1, 2, \ldots, k-1$$

$$\sum_{1 \le i \le k} y(s_i) = 1$$

where $h(s)$ is the average time spent in the state $s = (\mu, \nu)$ and is given by

$$h(s) = 1/\sum_{t \in T} \gamma(t) * \nu(t)$$

and k is the number of states in the set $\mathbf{S(MT)}$.

The stationary probabilities $y(s)$ and the average sojourn times $h(s)$ for M-timed Petri net (Fig. 4.15) are given in Table 4.5.

s_i	μ_i	v_i	t_k	μ_{ij}	e_l	s_j	p
1	0 0 0 1 0 1 0	0 1 0 0 0	2	1 0 1 1 0 1 0	1 0 1 0 0	2	1.000
2	0 0 0 0 0 1 0	1 0 1 0 0	1	0 1 0 0 0 1 0	0 0 0 0 0	3	0.200
			3	0 0 1 0 1 1 0	0 0 0 1 0	4	0.800
3	0 1 0 0 0 1 0	0 0 1 0 0	3	0 1 1 0 1 1 0	0 1 0 1 0	5	1.000
4	0 0 1 0 0 0 0	1 0 0 1 0	1	0 1 1 0 0 0 0	0 1 0 0 0	5	0.048
			4	0 0 1 0 0 1 1	0 0 0 0 1	6	0.952
5	0 0 0 0 0 0 0	0 1 0 1 0	2	1 0 1 0 0 0 0	1 0 0 0 0	4	0.286
			4	0 0 0 0 0 1 1	0 0 0 0 1	7	0.714
6	0 0 1 0 0 1 0	1 0 0 0 1	1	0 1 1 0 0 1 0	0 1 0 0 0	7	0.333
			5	0 0 1 1 0 1 0	0 0 1 0 0	2	0.667
7	0 0 0 0 0 1 0	0 1 0 0 1	2	1 0 1 0 0 1 0	1 0 0 0 0	6	0.800
			5	0 0 0 1 0 1 0	0 0 0 0 0	1	0.200

Table 4.4 State transitions

s	h(s)	y(s)
1	5.000	0.019
2	8.000	0.245
3	10.000	0.061
4	1.905	0.051
5	1.429	0.011
6	13.333	0.538
7	4.000	0.075

Table 4.5 Stationary probabilities

Performance measures

The copier is idle if $\mu(p_3) = 1$. Examining Table 4.4, we find that in states s_4 and s_6, $\mu(p_3) = 1$. The equilibrium probability that the copier is idle is equal to $y(s_4) + y(s_6)$. From Table 4.5, we find that $y(s_4) + y(s_6) = 0.051 + 0.538 = 0.589$. Hence, the utilization of the copier is equal to $1 - (y(s_4) + y(s_6)) = 1 - 0.589 = 0.411$.

Observe also that the copier is not idle when either $f(t_2) > 0$ or $f(t_3) > 0$ i.e., when processing either class-1 job or class-2 job. Examining Table 4.4, we find that in states s_1, s_5 and s_7, $f(t_2) > 0$. Hence, the utilization of the copier by class-1 jobs (managers) is equal to $y(s_1) + y(s_5) + y(s_7) = 0.019 + 0.011 + 0.075 = 0.105$. Similarly, in states s_2 and s_3, $f(t_3) > 0$. Hence, the utilization of the copier by class-2 jobs (secretaries) is equal to $y(s_2) + y(s_3) = 0.245 + 0.061 = 0.306$. The total utilization of the copier is equal to the sum of the utilization of the copier by class-1 jobs and class-2 jobs, i.e., equal to 0.105+0.306=0.411 as obtained above.

Next, we calculate the average turnaround time and the average waiting time in queue for both classes of jobs.

Class-1 jobs:

average service time of copier $= 1/0.2 = 5$ min

average job preparation time $= 1/0.025 = 40$ min

average throughput rate

$$= \frac{\text{utilization of copier by class-1 jobs}}{\text{average service time of copier}}$$

$$= \frac{0.105}{5} = 0.021 \text{ jobs per min}$$

average turnaround time $= 1/0.021 = 47.62$ min

average waiting time

= average turnaround time − average job preparation time

− average service time

$= 47.62 - 40 - 5 = 2.62$ min

Class-2 jobs:

average service time of copier $= 1/0.1 = 10$ min

average service time of stapler $= 1/0.5 = 2$ min

average job preparation time $= 1/0.05 = 20$ min

average throughput rate

$$= \frac{\text{utilization of copier by class-2 jobs}}{\text{average service time of copier}}$$

$$= \frac{0.306}{10} = 0.0306 \text{ jobs per min}$$

average turnaround time $= 1/0.0306 = 32.68$ min

average waiting time

= average turnaround time − average job preparation time

− average service time of copier − average service time of stapler

$= 32.68 - 20 - 10 - 2 = 0.68$ min

Finally, the office copying system (copier and stapler) is idle if $\mu(p_3) = 1$ and $\mu(p_6) = 1$. Examining Table 4.4, we find that in state s_6, $\mu(p_3) = 1$ and $\mu(p_6) = 1$. The equilibrium probability that the office copying system is idle is equal to $y(s_6) = 0.538$. Hence, the utilization of the office copying system is equal to $1 - y(s_6) = 1 - 0.538 = 0.462$.

We observe from above computations that the utilization of the copier is only 41.1 per cent. This suggests that the copier is under utilized. We could permit managers and secretaries from the neighbouring department to use the copier in order to increase its utilization; however this will increase the average waiting time for both the classes of jobs. Hence, there is a trade off between the average waiting time and the utilization of the copier. These computations help making the decision of installing the optimum number of copiers in the office.

Chapter 5

Summary

This thesis deals with system performance analysis using timed Petri nets. Two classes of timed Petri nets — D-timed and M-timed Petri nets are examined. The following is a brief summary of the thesis.

- We extended the formalism of D-timed Petri nets to include non-singular nets. We also introduced a new definition — *guarded due to inheritance place* — to assist in classifying Petri nets as conflict free nets.

- Analyses of D-timed and M-timed Petri nets are automated. The state graphs of D-timed and M-timed petri nets shown in Tables 4.2, 4.3 and 4.4 (Chapter 4) are generated by our computer programs.

- State graphs(transition graphs) obtained from the analysis of timed Petri net models are known to be large. Large graphs however are difficult to analyse and difficult to understand. The more states present in the graph being analysed, the more expensive computationally it is to calculate the performance measures such as turnaround time or throughput of the modeled system. In this thesis we present a set of reduction rules, namely, vertex reduction, junction removal, decision removal, multi-in multi-out vertex removal, merging parallel self-loops, merging parallel edges, and self-loop removal. These reduction rules permit systematic reduction of states in a graph to any size desired. From these reduced graphs numerical values of performance parameters such as throughput and delay can be easily computed.

- We also developed a more general algorithm for vertex folding than those presently existing. It is based on the law of conservation of transition time. To the best of our knowledge, this is the first general vertex folding algorithm.

- We demonstrated the application of these reduction rules to a version of stop-and-wait protocol.

- All these rules are incorporated into a graph reduction software package which can be used in automated performance prediction. Graph reduction software tools such as these are important and form an integral part of automated protocol performance prediction tools[RUD 83].

- We finally illustrated the application of timed Petri nets to performance prediction with the help of two examples.

  1. The first example deals with the analysis of a stop-and-wait protocol. It is modeled as a D-timed Petri net. The model of stop-and-wait protocol developed in this thesis is more general than one employed in an earlier study[ZUB 85a]. Losing messages, messages being damaged and losing acknowledgements are modeled explicitly. We calculate the optimum message length and performance of the stop-and-wait protocol.

  2. The second example deals with the analysis of an office copying system. It is modeled as an M-timed Petri net. We calculate various performance measures such as average turnaround time and average waiting time in queue.

Three computer programs dealing with D-timed Petri net analysis, M-timed Petri net analysis and graph reduction rules are written in PASCAL programming language. All these programs are presently available on IBM-PC AT; if needed these programs could be easily ported to other machines.

The data structure used in all three computer programs is a linked list. Hence, the size of the timed Petri net and also the size of the transition graph that these computer programs can handle is constrained only by Turbo Pascal compiler but not by any data structure used inside these computer programs.

The computer programs dealing with D-timed Petri net analysis, M-timed Petri net analysis and graph reduction rules comprise respectively of 2100, 1900 and 1450 lines of PASCAL code.

The computer programs dealing with D-timed and M-timed Petri net analyses generate the following results:

1. Classification of net
   The computer programs examine the timed Petri net and determine whether it is conflict free or not. These programs output all those places which are shared , free-choice and guarded.

2. Construction of the state graph
   The computer programs construct the state graph based on the timed Petri net formalism developed in Chapter 2 and output the state graph in the form

of a table consisting of the present state, the next state, the transition time, the probability and the enable functions.

3. Computation of stationary probabilities
   The computer programs solve the system of simultaneous linear equations (page 78) to obtain the stationary probabilites of the states in the state graph.

The computer program dealing with the graph reduction rules has two phases: enabling phase and firing phase.

In the enabling phase, the computer program examines the given graph and displays all the graph reduction rules which are enabled. Then it waits for the user to choose any one of the enabled rules for firing.

In the firing phase, the computer program fires the user selected rule and makes the appropriate changes to the original graph as required by the rule.

Hence, the computer program is in a loop — (enabling phase, firing phase, enabling phase) until no more graph reduction rules are enabled or the user stops by keying in the character 's' which stands for stopping the computation.

With a friendly user interface, these programs would be useful in predicting the performance measures such as turnaround time or throughput of computer systems, communication protocols and office information systems.

We recommend two possible areas of further study.

1. Development of the formalism for timed Petri nets with general distribution firing times

2. To reflect the power of our tool, we suggest the following two practical problems be attempted

   (a) Performance analysis of IBM token ring network

   (b) Bus contention problem in Ethernet

Bibliography

[AGE 79] Agerwala, T. "Putting Petri nets to work", IEEE Computer, December 1979, pp.85-94.

[BEI 70] Beizer, B. "Analytical techniques for the statistical evaluation of program running time", Fall Joint Computer Conference, 1970, pp. 519-524.

[GRA 73] Graham, R.H. "Performance prediction", in: Advance Course in Software Engineering, Lecture notes in Economics and Mathematical Systems (Springer-verlag, Berlin, 1973), vol. 81, pp.395-463.

[LEV 87] Leveson, N.G. and Stolzy, J.L. "Safety analysis using Petri nets", IEEE Trans. on Software Engineering, Vol. SE-13, No. 3, 1987, pp.386-397.

[MAR 84] Marsan, M.A., Conte, G., and Balbo, G. "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems", ACM Trans. on Computer systems, Vol. 2, No.2, 1984, pp.93-122.

[MER 76] Merlin, P.M. and Farber, D.J. "Recoverability of communication protocols — Implications of a theoretical study", IEEE Tran. on Communication, Vol. COM-24, No. 9, 1976, pp. 1036-1043.

[MOLL 82] Molloy, M.K. "Performance analysis using stochastic Petri nets", IEEE Trans. on Computers, Vol. C-31, No. 9, 1982, pp.913-917.

[PET 77] Peterson, J.L. "Petri nets", ACM Computing surveys, Vol. 9, No. 3, 1977, pp.223-252.

[PET 81] Peterson, J.L. *Petri net Theory and Modeling of systems*, Englewood Cliffs, NJ: Prentice-Hall, 1981.

[RAM 74] Ramchandani, C. "Analysis of asychronous concurrent systems by timed Petri nets", Project MAC Report NO. MAC-TR-120, MIT, Feb. 1974.

[RAZ 84] Razouk, R.R. "The derivation of performance expressions for communication protocols from timed Petri nets", Computer Communication Review, Vol. 14, No. 2, 1984, pp.210-217.

[REI 82] Reiser, M. "Performance evaluation of data communication systems", Proc. IEEE, Vol. 70, 1982, pp.171-196.

[RUD 83] Rudin, H. "From formal protocol specification towards automated performance prediction", Protocol Specification, Testing and Validation, III, 1983, pp.257-269.

[RUD 84] Rudin, H. "Improved algorithm for estimating protocol performance", Protocol Specification, Testing and Verfication, IV, 1984, pp.515-525.

[SIF 77] Sifakis, J. "Petri nets for performance evluation", in: Beilner, H. and Gelenbe, E. (eds.), Measuring, Modeling and Evaluating Computer Systems, 1977, pp.75-93.

[TAN 81] Tanenbaum, A.S. *Computer Networks*, Englewood Cliffs, NJ: Prentice-Hall, 1981.

[ZUB 80] Zuberek, W.M. "Timed Petri nets and preliminary performance evaluation", in: Proceedings. IEEE 7th Annual Symp. Computer Architecture, 1980, pp.88-96.

[ZUB 85a] Zuberek, W.M. "Extended D-timed Petri nets, timeouts, and analysis of communication protocols", in: Proceedings. ACM Annual conference, Denver,Co. 1985, pp.10-15.

[ZUB 85b] Zuberek, W.M. "Augmented M-timed Petri nets, modeling and performance evluation of computer systems", Trans. Soc. Computer Simulation, Vol. 2, No. 1, 1985, pp.135-153.

[ZUB 86a] Zuberek, W.M. "Inhibitor D-Timed Petri nets and performance analysis of communication protocols", INFOR, vol.24, No.3, 1986, pp.231-249.

[ZUB 86b] Zuberek, W.M. "Modified D-timed Petri nets, timeouts, and modeling of communication protocols", in: Proceedings. 6th Int. Conf. on Distributed Computing Systems, 1986, pp.452-457.