1

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

copie de votre référence

Our file - Notre référence

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.

Canada

Localization and Diagnosis of Structural Problems in Petri Net Models

by

Zhengping You

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of
Master of Science

**Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario**

June 30, 1993

**© copyright
1993, Zhengping You**

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-89878-X

Canada

Name ZHENG PING YOU

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

Systems Science

SUBJECT TERM

0790

SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523

Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language 0679
General 0289
Ancient 0290
Linguistics 0291
Modern
Literature 0401
General 0294
Classical 0295
Comparative 0297
Medieval 0298
Modern 0316
African 0591
American 0305
Asian 0352
Canadian (English) 0355
Canadian (French) 0593
English 0311
Germanic 0312
Latin American 0315
Middle Eastern 0313
Romance 0314
Slavic and East European

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion
General 0318
Biblical Studies 0321
Clergy 0319
History of 0320
Philosophy of 0322
Theology 0469

SOCIAL SCIENCES

American Studies 0323
Anthropology
Archaeology 0324
Cultural 0326
Physical 0327
Business Administration
General 0310
Accounting 0272
Banking 0770
Management 0454
Marketing 0338
Canadian Studies 0385
Economics
General 0501
Agricultural 0503
Commerce-Business 0505
Finance 0508
History 0509
Labor 0510
Theory 0511
Folklore 0358
Geography 0366
Gerontology 0351
History
General 0578

Ancient 0579
Medieval 0581
Modern 0582
Black 0328
African 0331
Asia, Australia and Oceania 0332
Canadian 0334
European 0335
Latin American 0336
Middle Eastern 0333
United States 0337
History of Science 0585
Law 0398
Political Science
General 0615
International Law and Relations 0616
Public Administration 0617
Recreation 0814
Social Work 0452
Sociology
General 0626
Criminology and Penology 0627
Demography 0938
Ethnic and Racial Studies 0631
Individual and Family Studies 0628
Industrial and Labor Relations 0629
Public and Social Welfare 0630
Social Structure and Development 0700
Theory and Methods 0344
Transportation 0709
Urban and Regional Planning 0999
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture
General 0473
Agronomy 0285
Animal Culture and Nutrition 0475
Animal Pathology 0476
Food Science and Technology 0359
Forestry and Wildlife 0478
Plant Culture 0479
Plant Pathology 0480
Plant Physiology 0817
Range Management 0777
Wood Technology 0746
Biology
General 0306
Anatomy 0287
Biostatistics 0308
Botany 0309
Cell 0379
Ecology 0329
Entomology 0353
Genetics 0369
Limnology 0793
Microbiology 0410
Molecular 0307
Neuroscience 0317
Oceanography 0416
Physiology 0433
Radiation 0821
Veterinary Science 0778
Zoology 0472
Biophysics
General 0786
Medical 0760

EARTH SCIENCES

Biogeochemistry 0425
Geochemistry 0996

Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleocology 0426
Paleontology 0418
Paleozoology 0985
Polynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0768
Health Sciences
General 0566
Audiology 0300
Chemotherapy 0992
Dentistry 0567
Education 0350
Hospital Management 0769
Human Development 0758
Immunology 0982
Medicine and Surgery 0564
Mental Health 0347
Nursing 0569
Nutrition 0570
Obstetrics and Gynecology 0380
Occupational Health and Therapy 0354
Ophthalmology 0381
Pathology 0571
Pharmacology 0419
Pharmacy 0572
Physical Therapy 0382
Public Health 0573
Radiology 0574
Recreation 0575

Speech Pathology 0460
Toxicology 0383
Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences
Chemistry
General 0485
Agricultural 0749
Analytical 0486
Biochemistry 0487
Inorganic 0488
Nuclear 0738
Organic 0490
Pharmaceutical 0491
Physical 0494
Polymer 0495
Radiation 0754
Mathematics 0405
Physics
General 0605
Acoustics 0986
Astronomy and Astrophysics 0606
Atmospheric Science 0608
Atomic 0748
Electronics and Electricity 0607
Elementary Particles and High Energy 0798
Fluid and Plasma 0759
Molecular 0609
Nuclear 0610
Optics 0752
Radiation 0756
Solid State 0611
Statistics 0463
Applied Sciences
Applied Mechanics 0346
Computer Science 0984

Engineering
General 0537
Aerospace 0538
Agricultural 0539
Automotive 0540
Biomedical 0541
Chemical 0542
Civil 0543
Electronics and Electrical 0544
Heat and Thermodynamics 0348
Hydraulic 0545
Industrial 0546
Marine 0547
Materials Science 0794
Mechanical 0548
Metallurgy 0743
Mining 0551
Nuclear 0552
Packaging 0549
Petroleum 0765
Sanitary and Municipal 0554
System Science 0790
Geotechnology 0428
Operations Research 0796
Plastics Technology 0795
Textile Technology 0994

PSYCHOLOGY

General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

The undersigned recommend to the Faculty of Graduate Studies
and Research acceptance of the thesis

“Localization and Diagnosis of Structural Problems in Petri Net Models”

submitted by Zhengping You, M.S.
in partial fulfillment of the requirements for
the degree of Master of Science

Thesis Supervisors

Chair, Department of Systems and Computer Engineering

Carleton University

Abstract

With the development of computer hardware and software technology, it is not difficult to specify the structural properties of large Petri net models. The real bottleneck in structural analysis of Petri net models is met when the models have structural problems such as structural nonliveness and structural unboundedness. It is difficult to localize the sources of problems and diagnose them in order to fix the models. The thesis defines the linear programming concepts of nonviability and infeasibility for Petri net models and relates them to well-known Petri net structural properties such as structural unboundedness, inconsistency, nonconservativeness, and nonrepetitiveness. It then applies viability and infeasibility analysis techniques, originally developed for processing network model debugging, to the localization of structural problems in Petri net models. To assist the diagnosis of the problems, we also classify some bad structures and relate them to subclasses of siphons and traps. Several examples are given to show how to diagnose the problems using the structural information given by the localization algorithms.

To aid compositional structural analysis of large Petri net models, the thesis discusses several special communication media which preserve good structural properties from the subsystems to the global system. It also presents the Petri box viability and infeasibility analysis algorithm to localize the sources of the problems to a small subsystem for easy diagnosis.

Finally, a Petri net model of a distributed simulation system is taken as a case study to show the effectiveness of our method in debugging large Petri net models.

Acknowledgment

Thanks are due to my supervisors Dr. John W. Chinneck and Dr. Murray Woodside for their supervision, technical discussions at various stages in the development of this thesis. This thesis would not be completed without their ideas, patience and financial support. Thanks are also due to Philippe Schnoebel who sent me test cases.

This thesis is dedicated to my grandmother and parents.

Table of Contents

List of Figures.....	viii
List of Symbols.....	xi
Index of Terms	xii
Chapter 1 Introduction	1
1.1 Problem Statement.....	1
1.2 Outline of the Thesis.....	2
Chapter 2 Structural Analysis of Petri Nets.....	5
2.1 P/T Nets and Their Properties.....	5
2.1.1 Fundamental Concepts of P/T Nets	5
2.1.2 Behavior Properties.....	8
2.1.3 Structural Properties of Petri Nets	9
2.2 Current Techniques to Detect Structural Problems in Petri Nets.....	12
2.2.1 Invariant Analysis.....	12
2.2.2 Siphons and Traps.....	14
Chapter 3 Debugging Processing Networks by Viability and Infeasibility Analysis	15
3.1 Processing Network Concepts	15
3.2 Processing Network Representation of LPs.....	18
3.3 Localization of Infeasibility by IIS Technique.....	20

3.4	Localization of Nonviability	22
Chapter 4	Processing Network Models of Petri Nets	25
Chapter 5	Analyzing Petri Net Structure by Viability and Infeasibility Analysis	28
5.1	Nonviability and Infeasibility Interpretation of Structural Problems in Petri Nets.....	28
5.2	Structural Debugging of Petri Net Models	30
5.2.1	Algorithms	31
5.2.2	Advantages of the Algorithms	32
5.2.3	Implementation	33
5.3	Nonviability, Infeasibility and Siphons and Traps.....	34
5.4	Experiments	40
5.4.1	Check Nonrepetitiveness	40
5.4.2	Check Inconsistency	41
5.4.3	Check Structural Unboundedness.....	41
5.4.4	Check Nonconservativeness	42
5.4.5	A Localization Example	43
5.5	Diagnosis of Nonviability	44
5.5.1	A Case of a Reader/Writer System	44
5.5.2	A Case of a Deadlock in a User Server Protocol.....	46

Chapter 6	Compositional Viability and Infeasibility Analysis of Large Petri Nets....	49
6.1	Viability and Infeasibility Analysis of Petri Nets Composed via Shared Places	49
6.2	Viability and Infeasibility Analysis of Petri Nets Composed via Shared Transitions.....	53
6.3	Viability and Infeasibility Analysis of Petri Nets Composed via Shared Places and Transitions.....	57
6.4	Petri Box Analysis Approach	60
6.4.1	Decomposition	60
6.4.2	Algorithm.....	64
6.4.3	Localizing a smaller <i>INS</i> or <i>IIS</i> in a Large Petri net without Modules	66
7.0	A Case Study	69
7.1	A Distributed Simulation System	69
7.2	Petri Net Model of the System.....	70
7.3	Model Debugging Experiments	73
8.0	Conclusions.....	77
8.1	Contributions	77
8.2	Future Research	78
Appendix A.....		79
Appendix B.....		82
References.....		84

List of Figures

FIGURE 2.1	Subclasses of Petri Nets.....	6
FIGURE 2.2	A P/T Net.....	7
FIGURE 3.1	A Processing Node	16
FIGURE 3.2	A Processing Network	17
FIGURE 3.3	A Nonconserving Processing Node	18
FIGURE 3.4	A processing network representation of an LP.....	20
FIGURE 4. 1	Conflict.....	25
FIGURE 4. 2	Transformation between a Petri net and a Processing Network	27
FIGURE 5.1	A nonviable Petri net	34
FIGURE 5.2	<i>INS</i> generated by nonviability.....	34
FIGURE 5.3	An <i>INS</i> of Figure 5.1.....	36
FIGURE 5.4	A nonviable P/T net due to a generating cycle	37
FIGURE 5.5	A nonviable P/T net due to a pure siphon.....	38
FIGURE 5.6	An <i>INS</i> of Figure 5.5.....	39
FIGURE 5.7	A viable P/T net with traps and a siphon	39
FIGURE 5.8	A Petri net with structural problems	40
FIGURE 5.9	A processing network model (5-5) of the P/T net in Figure 5.8.....	41
FIGURE 5.10	An <i>INS</i> of the P/T net in Figure 5.8	42
FIGURE 5.11	A Petri net model	42

FIGURE 5.12	The processing network model (5-6) of Figure 5.11	43
FIGURE 5.13	The Dual <i>INS</i> of the P/T net in Figure 5.11	43
FIGURE 5.14	A Petri net model of a reader/writer system	45
FIGURE 5.15	An <i>INS</i> of the reader/writer model.....	46
FIGURE 5.16	The Petri net model of a user server protocol.....	47
FIGURE 5.17	An <i>INS</i> of the protocol model	47
FIGURE 5.18	Another <i>INS</i> of the protocol model.....	48
FIGURE 6.1	Composition of nets by shared places.....	50
FIGURE 6.2	A nonviable net composed by shared transitions.....	55
FIGURE 6.3	Shared by Transitions	56
FIGURE 6.4	Composition of nets via shared transitions and places	58
FIGURE 6.5	An <i>INS</i> of Figure 6.4(c)	58
FIGURE 6.6	Composition via a well formed medium.....	59
FIGURE 6.7	A Petri Box	61
FIGURE 6.8	Composition of Petri Boxes.....	61
FIGURE 6.9	Decomposition of Petri Box Model.....	63
FIGURE 6.10	A Block and Link Structure Petri Net.....	63
FIGURE 6.11	Localizing an <i>INS</i> or <i>IIS</i> to a smaller subnet	64
FIGURE 6.12	Petri Box 1	58
FIGURE 6.13	Petri Box 2	68
FIGURE 6.14	An <i>INS</i> of Box 2.....	68

FIGURE 7.1	An example of distributed simulation.....	70
FIGURE 7.2	A Petri box of receiver module	71
FIGURE 7.3	A Petri box of simulation module	72
FIGURE 7.4	A Petri box of transmitter module	73
FIGURE 7.5	A <i>INS</i> of the model	74
FIGURE 7.6	An <i>INS</i> of the model	75
FIGURE 7.7	A <i>INS</i> of example 3.....	76
FIGURE B.1	Receiver and transmitter code	82
FIGURE B.2	Simulator code	83

List of Symbols

- A** Incidence matrix of a Petri net;
- F** A set of edges of a Petri net;
- M** Marking of a Petri net;
- N** A Petri net $N=(P,T,F,W,M)$;
- P** A set of Places of a Petri net;
- PN** A Petri net structure $PN=(P,T,F,W)$;
- PV** A set of processing nodes of a processing network;
- RV** A set of regular nodes of a processing network;
- T** A set of Transitions of a Petri net;
- W** A vector of weights of edges in a Petri net.

Index of Terms

A

absorbing cycle 36

B

bipartite processing network 18

boundedness 8

C

composition 50, 53, 56, 57, 59

composition by shared places 50

conservativeness 10

consistency 11

constraint nonviability 22

D

deadlock 8, 46

deletion filtering 21

dual nonviable 29

E

elastic filtering 21

equation rank nonviability 22

F

firing 7

free choice net 6

G

generating cycle 36

I

inconsistency 41

infeasibility 65
invariants 12
irreducible nonviable system 23
irreducibly inconsistent system 20

L

large Petri net 64,66

liveness 8

M

marking 7

MINOS(IIS) 33

N

nonconservativeness 42

nonrepetitiveness 40

nonviability 28, 44, 65

nonviable 22, 28

nonviable transition 29

P

P/T nets 5

Petri box 60, 61, 62, 65, 66, 71, 72, 73

Petri net 5

Petri net structure 5

processing networks 16

processing node 16

PROFLOW 33

p-semiflow 12, 26

pure processing network 17

pure siphon 37

pure trap 35

R

reachability 8

regular nodes 16

repetitiveness 10

reverse and dual Nets 11

S

sensitivity filtering 21

siphon 14

s-semiflow 12

state machine 6

static structural properties 11

st-component 14

structural Boundedness 10

structural Infeasibility 29

structural Liveness 9

structural Unboundedness 41

structurally dual infeasible 29

support 13

T

trap 14

t-semiflow 26

V

viability 62

W

well-formed medium 59

Chapter 1 Introduction

Petri nets provide powerful models with an intuitively appealing visualization power for concurrent systems. Successful application areas include modeling and analysis of distributed systems, concurrent and parallel programs and specification and verification of communication protocols. Petri net models provide a formal framework wherein the two basic problems of analysis and synthesis can be investigated.

Analysis techniques examine whether a net model satisfies a certain set of properties of good behavior such as liveness and boundedness. There are three major techniques for answering these questions: reachability, reduction and structural analysis. For systems with a finite state space (or bounded systems), reachability analysis can answer all of the analysis questions. However, this technique requires an exhaustive exploration of the state space, which hinders its use on large and complex systems. Reduction or decomposition techniques simplify the system by means of reduction rules which preserve the properties under study.

Structural analysis studies the relationship between the system behaviors and the underlying net structure and characterizes behavior properties of a Petri net system by its topological properties like invariants [Lautenbach 74, 87a] or siphons and traps [Lautenbach 87b, Ezpeleta 91, Esparza 92]. The advantage of structural analysis is that the complexity of the analysis techniques is independent of the size of the state space. It can deal with large and complex or even unbounded systems.

Synthesis techniques deal with the question of how to construct systems with a given set of good behavior properties. This problem is strongly related to system design methodologies [Esparza 91b, Souissi 91, 92].

This thesis studies new structural analysis techniques to assure that a Petri net model has a good structure.

1.1 Problem Statement

Hardware and software have advanced in recent years and it is now easy to build large Petri net models using graphic tools such as $\mathcal{GSPN}$ [Chiola 87] and Design/CPN [Meta

92]. The structural analysis of a large Petri net model has become routine. The real bottle-neck is no longer the ability to verify the structural property of large nets but the ability to repair the models if they are found to have problems.

A problem may have global symptoms, but the causes are local. An appropriate analogy is program compilation, where syntax errors on specific lines cause numerous error messages throughout the program. The first step of model debugging is to locate the problem.

When a large and complex Petri net model contains errors, whether introduced during model formulation or pre-existing in the design, it might be difficult to find them. Problem diagnosis may exceed human cognitive analysis capacity. Even a simple mistake such as reversing an arc accidentally is hard to find manually in a complex graph. In these cases, the modelers need to know not only what the problem is but also where the problem is in order to fix the model. Some Petri net tools can check if a Petri net has structural problems, for example structural unboundedness, but they can not localize the sources causing the problems. Therefore we need a kind of intelligent debugging tool which can isolate a problem from a large system. The size of the problem source located should be as small as possible for easy diagnosis. We also need to classify bad structures of Petri net model and supply modelers with information about what is wrong with the structures to assist model repair. To achieve these purposes, this thesis applies the viability and infeasibility analysis techniques, which have shown their advantages in linear programming (LP) model debugging, to structural debugging of Petri net models.

1.2 Outline of the Thesis

Viability theory is defined for processing networks [Chinneck 90a, 92]. If the flow in at least one edge in a processing network is forced to zero by the structure of the network, then the network is nonviable. Chinneck's algorithms can be used to isolate and diagnose the problem. Infeasibility analysis is a more general method. Given a system of linear equations which is infeasible, various methods can be applied to isolate the portion of the model that contains the source of the infeasibility and to diagnose the problem [Greenberg 83, 91].

Based on the fact that structural properties of Petri nets can be expressed as a set of linear equations or represented by a processing network, the thesis defines viability and infeasibility

bility of Petri net models and interprets some common structural problems such as structural unboundedness, inconsistency, nonconservativeness and nonrepetitiveness by viability and infeasibility analysis. This interpretation makes it possible to apply those powerful techniques to Petri net model debugging.

In Chapter 2, structural and behavioral properties of Petri nets are introduced. The state of the art of structural analysis techniques is reviewed to show that these techniques can pre-screen certain behavior problems such as deadlock and unboundedness of a Petri net model. These techniques can only detect structural problems but can not localize them or indicate causes of the problems.

In Chapter 3, viability and infeasibility analysis theory is reviewed. Techniques for localizing nonviability or infeasibility are investigated. Chinneck's technique of isolating irreducible inconsistent systems [Chinneck 91] is presented.

Chapter 4 shows that there is topological and logical similarity between processing networks and Petri nets. A Petri net can be represented by a processing network with the same topology.

In Chapter 5, structural problems of Petri nets such as inconsistency, nonconservativeness, nonrepetitiveness and structural unboundedness are interpreted using viability and infeasibility theory. Inconsistency or nonconservativeness is a kind of nonviability in a Petri net or in its reverse and dual net. Moreover, structural unboundedness or nonrepetitiveness is an infeasible problem of a Petri net or in its reverse and dual net. Four algorithms for localizing these structural problems are given in this section. The relationship between siphons and traps and nonviability is discussed to help in problem diagnosis. Experiments are done to show how the technique works in several example models. A reader/writer system and a user-server protocol are analyzed and diagnosed to show how the debugging technique can be used to repair Petri net models and verify protocols.

The concept of Petri boxes [Best 91] is used in the Petri nets research community to support modelling and analysis of systems designed by a modular approach or object oriented method. For a large Petri net model composed of a number of smaller nets or Petri boxes, we want to know what kind of composition can preserve the good structural properties of the boxes and how to localize problems in such a large system by analyzing a smaller sub-

system instead of a large net to ease the diagnosis. In Chapter 6, an algorithm for viability and infeasibility analysis of Petri box models and an algorithm to localize a smaller source in a large net are developed.

Finally, a Petri net model of a distributed simulation system is chosen as a case study to show how viability and infeasibility analysis works in localizing structural problems and leads to a diagnosis of the problems in large Petri net models.

Chapter 2 Structural Analysis of Petri Nets

A Petri net model is analyzed to understand the modeled system behavior. Structural analysis techniques can explain certain kinds of model behavior efficiently. In this chapter, these techniques are investigated after introducing basic concepts of Petri nets.

2.1 P/T Nets and Their Properties

A Petri net is a kind of directed graph with an initial state called the *initial marking* M_0 . A Petri net consists of two kinds of nodes: places and transitions. In the graphic representation, places are drawn as circles and transitions as bars. The places and transitions are linked by arcs. A marking (state) assigns to each place a nonnegative integer k , which is represented by k tokens in the place. To model various kinds of systems, several categories of Petri nets have been developed such as elementary nets (condition/event nets) [Reisig 85, Thiagajian 87], place/transition (P/T) nets [Reisig 87, Peterson 81, Murata 89a] and high level Petri nets including Predicate/Transition nets [Genrich 81, Genrich 87] and Colored Petri nets [Jensen 87]. Here we are only concerned with the properties of P/T nets.

2.1.1 Fundamental Concepts of P/T Nets

For convenience of discussion, we give formal definitions and notations for P/T nets used in the thesis.

Definition 2.1 (P/T nets)

A five-tuple $N=(P,T, F,W,M_0)$ is called a place/transition (P/T) net iff

1. $P=\{p_1, p_2, \dots, p_m\}$ is a finite set of places;
2. $T=\{t_1, t_2, \dots, t_n\}$ is a finite set of transitions;
3. $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation);
4. $W: F \rightarrow \{1, 2, 3, \dots\}$ is a weight function and $W(p, t)$ is the weight for the arc from p to t and $W(t, p)$ is the weight of the arc from t to p ;
5. $M_0: P \rightarrow \{0, 1, 2, 3, \dots\}$ is the initial marking;
6. $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$;
7. A P/T net structure $PN=(P,T,F,W)$ without any specific initial marking is denoted by PN .

Let us define the following symbols for a pre-set and a post-set:

- t stands for the set of input places of t ;
- t • stands for the set of output places of t ;
- p stands for the set of input transitions of p ;
- p • stands for the set of output transitions of p .

P/T nets consist of several subclasses which have special interconnections between the places and transitions.

Definition 2.2 (Subclasses of Petri Nets)

1. A P/T net is said to be **ordinary** if all of its arc weights are 1's;
2. A P/T net $PN=(P,T, F,W)$ is said to be **pure** iff $\forall (p, t) \in (P \times T): (p, t) \in F$ implies $(t, p) \notin F$.

We assume that the nets considered in this thesis are pure.

3. A **state machine** (SM) is an ordinary P/T net such that each transition t has exactly one input place and exactly one output place, i.e., $|t \bullet| = |\bullet t| = 1$ for all $t \in T$.
4. A **marked graph** (MG) is an ordinary P/T net such that each place p has exactly one input transition and exactly one output transition, i.e., $|p \bullet| = |\bullet p| = 1$ for all $p \in P$.
5. A **free choice net** (FC) is an ordinary Petri net such that every arc from a place is either a unique outgoing arc or a unique incoming arc to a transition, i.e., for all $p_1, p_2 \in P$, if $p_1 \bullet \cap p_2 \bullet \neq \emptyset$ then $|p_1 \bullet| = |p_2 \bullet| = 1$.

Figure 2.1 includes examples of several special P/T nets.

FIGURE 2.1 Subclasses of Petri Nets

The behavior of many systems can be described in terms of system states and their changes. In order to simulate the dynamic behavior of a system, a state or marking of a Petri net is changed according to the dynamic properties defined as follows.

Definition 2.3 (Dynamic Properties)

Let $N=(P,T;F,W,M_0)$ be a P/T net

1. a function $M: P \rightarrow \{0,1,2,3,\dots\}$ is called a marking of PN;
2. a transition t is enabled at M iff $\forall p \in P: M(p) \geq W(p,t)$;
3. A *firing* of an enabled transition t removes $W(p,t)$ tokens from each input place p of t and adds $W(t,p)$ tokens to each output place p of t .
4. if $t \in T$ is a transition which is enabled at a marking M , then firing t will yield a new marking M' given by the equation:

$$M'(p) = M(p) - W(p,t) + W(t,p) \quad \text{for } \forall p \in P;$$

5. A matrix $A: P \times T \rightarrow \mathbb{Z}$ (integers) indexed by P and T such that $A(p_i, t_j) = W(t_j, p_i) - W(p_i, t_j)$ is called the incidence matrix of N .

Example 2.1

The marking and incidence matrix of Figure 2.2 (we assume the weights of unmarked arcs are 1) are

$$M_0^T = \{1, 0, 0, 0\}$$

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

If t_1 fires, then the new marking is $M^T = \{0, 1, 1, 0\}$ under which t_2 is enabled.

FIGURE 2.2 A P/T Net

2.1.2 Behavior Properties

The objective of modelling systems by Petri nets is to analyze the original system. This analysis may lead to some important insights into the behavior of the original system. There are two types of properties which can be studied with a Petri net model: behavior properties and structural properties. The former depend on the initial marking, the latter are independent of the initial marking. Here we define some common behavior properties.

Definition 2.4 (Reachability)

A marking M_n is said to be *reachable* from a marking M_0 if there exists a sequence of firings that transforms M_0 to M_n . The set of all possible markings reachable from M_0 in a net is denoted by $R(M_0)$.

It has been shown that the reachability problem is decidable [Mayr 84], but it takes at least exponential time and space to verify in the general case.

Deadlock has been the subject of a number of studies of computer systems. The situation can be modeled by a Petri net. A *deadlock* in a Petri net is a transition or a set of transitions which can not fire. This problem is taken as a liveness problem defined below.

Definition 2.5 (Liveness)

A Petri net PN is said to be *live* or M_0 is said to be a live marking for PN, if, no matter what marking has been reached from the initial marking, it is possible to fire any transition of the net by progressing through some further firing sequence.

Places in a Petri net are often used to model buffers or registers for storing shared data. To verify if the overflow will happen in the buffers or registers, safeness and boundedness are important properties of a Petri net system.

Definition 2.6 (Boundedness)

A Petri net is said to be *bounded* if the number of tokens in each place does not exceed a finite number k for any marking reachable from M_0 , i.e., $M(p) \leq k$ for all places $p \in P$ and all markings $M \in R(M_0)$.

Although there are some “nice” characterizations of liveness and boundedness for some subclasses of Petri nets like marked graph, state machines [Murata 89a], and free choice nets [Best 87, Esparza 92], there are no necessary and sufficient conditions to characterize liveness and boundedness for generalized Petri nets because of the nondeterministic nature of state transitions in Petri nets. So the reachability analysis technique is used to verify if a Petri net system is live or bounded.

2.1.3 Structural Properties of Petri Nets

Structural properties are those that depend only on the topological structures of Petri nets. They are independent of initial markings in the sense that these properties hold for any initial marking or are concerned with the existence of certain firing sequences from some initial marking. Thus some structural properties, which are deterministic, are a subset of nondeterministic behavior properties. They can detect or prescreen certain kinds of deadlock and unboundedness of a Petri net system. Most of the structural properties of a Petri net can be characterized by the incidence matrix and its associated homogeneous equations or inequalities.

Definition 2.7 (*s and t vectors*)

Let $PN=(P,T,F,W)$ be a P/T net and $|P|=m$ and $|T|=n$.

1. A column vector $v: P \rightarrow \mathfrak{R}$ (real number) indexed by $\{P: p_1, p_2, \dots, p_m\}$ is called an *s-vector*,
2. A column vector $w: T \rightarrow \mathfrak{R}$ indexed by $\{T: t_1, t_2, \dots, t_n\}$ is called a *t-vector*.

In a Petri net, if a marking M is reachable from M_0 , then the following equation

$$M=M_0+AF$$

holds, where A is the incidence matrix of the net, F is a t-vector indicating the transition occurrences. The analysis of incidence matrix provides a useful tool to verify some structural properties. Among the following five structural properties, four can be characterized by the incidence matrix.

Definition 2.8 (*Structural Liveness*)

A P/T net is said to be *structurally live* if there exists a live initial marking for the net.

By this definition, if a P/T net is structurally nonlive then it is nonlive as well. But the converse is not true. The characterization condition of structural liveness for a general P/T net is still unknown.

Definition 2.9 (Structural Boundedness)

A P/T net is said to be *structurally bounded* if it is bounded for any finite initial marking.

If a P/T net is structurally bounded, then it is also bounded by Definition 2.6.

Theorem 2.1 [Murata 89a]

A P/T net PN is structurally bounded iff there exists an s-vector y of positive integers such that $A^T y \leq 0$. ■

Definition 2.10 (Conservativeness)

A P/T net is said to be *conservative* if there exists an s-vector i of positive integers such that the weighted sum of tokens $M^T i = M_0^T i = (\text{a constant})$ for every reachable marking set from M_0 and for every fixed initial marking M_0 .

Theorem 2.2 [Murata 89a]

A P/T net PN is conservative iff there exists an s-vector y of positive integers such that $A^T y = 0$. ■

A conservative Petri net means that it does not lose or gain tokens but merely moves them around. It is a special case of structurally bounded net.

Definition 2.11 (Repetitiveness)

A Petri net is said to be *repetitive* if there exists a marking M_0 and a firing sequence σ such that every transition occurs infinitely often in σ .

Theorem 2.3 [Murata 89a]

A P/T net PN is repetitive iff there exists a t-vector X of positive integers such that $AX \geq 0$. ■

Definition 2.12 (Consistency)

A P/T net is said to be *consistent* if there exists a marking M_0 and a firing sequence σ from M_0 back to M_0 such that every transition occurs at least once in σ .

Theorem 2.4 [Murata 89a]

A P/T net PN is consistent iff there exists a t-vector X of positive integers such that $AX=0$.

Consistency is a special case of repetitiveness.

We define the above four properties as *static structural properties*. Table 2.1 summarizes the necessary and sufficient conditions for these properties.

TABLE 2.1 Necessary and Sufficient Conditions for Static Structural Properties

Properties	Necessary and Sufficient Conditions
Structurally Bounded	$\exists y > 0, A^T y \leq 0$
Conservative	$\exists y > 0, A^T y = 0$
Repetitive	$\exists x > 0, Ax \leq 0$
Consistent	$\exists x > 0, Ax = 0$

Where A is incidence matrix of the P/T net. y is an s-vector, x is a t-vector.

Definition 2.13 (Reverse and Dual Nets)

Let $PN=(P,T,F,W)$ be a P/T net. The net $PN_d=(T,P,F,W)$ is the dual net of PN. The net $PN_r=(P,T,F^{-1},W)$ is the reverse net of PN. The net $PN_{rd}=(T,P,F^{-1},W)$ is the *reverse and dual net* of PN.

Corollary 2.1 (Duality Theorem)

1. A P/T net $PN=(P,T,F,W)$ is conservative iff $PN_{rd}=(T,P,F^{-1},W)$ is consistent;
2. A P/T net $PN=(P,T,F,W)$ is consistent iff $PN_{rd}=(T,P,F^{-1},W)$ is conservative;

Proof: It follows easily from properties in Table 2.1 and the definition of reverse-dual net that $A_{rd} = -A^T$, where A and A_{rd} are the incidence matrices of PN and PN_{rd} , respectively. ■

Although there is no good characterization condition of structural liveness for general Petri nets, the following theorem implies that some structurally nonlive Petri nets can be found by checking if they are nonconservative and inconsistent.

Theorem 2.5 [Ramchandani 73]

If a P/T net PN is structurally bounded and structurally live, then the net is both conservative and consistent. ■

This theorem implies that: (1) if a structurally bounded P/T net is nonconservative then the net is structurally nonlive; (2) if a structurally bounded P/T net is inconsistent then the net is structurally nonlive.

2.2 Current Techniques to Detect Structural Problems in Petri Nets

Solving invariants, and siphons and traps are the most frequently used techniques to detect if a Petri net has some structural problems such as structural unboundedness, nonconservativeness, inconsistency and nonrepetitiveness. In this section, these techniques are reviewed.

2.2.1 Invariant Analysis

Definition 2.14 (Invariants and semiflows)

- Let $I = (i_p)_{p \in P}$ be an s -vector of integers. I is a linear *s-invariant* iff for all markings M reachable from M_0 :

$$I^T M = I^T M_0$$

- I is an *s-semiflow* (or semiflow) iff

$$I^T A = 0$$

where A is the incidence matrix.

- I is a *p-semiflow* iff I is a semiflow and $I \geq 0$;
- A *p-semiflow* $I \neq 0$ of PN is called minimal iff there exists no non-negative *p-semiflow* I' such that $I \geq I'$.

- Let $J=(J_i)_{i \in T}$ be a t-vector of integers. J is a *t-invariant* iff $AJ=0$, where A is the incidence matrix.

Definition 2.15 (Support)

The *support* of a vector V , denoted by $\|V\|$, is defined by $\|V\|=\{i|V_i \neq 0\}$.

S-invariants can be used to tell whether a marking is a reachable marking of M_0 . If

$$I^T M \neq I^T M_0$$

then we can say M is not reachable from M_0 without any simulation. It can also tell whether a marked graph is live by the following theorem.

Theorem 2.6 [Lautenbach 87a]

Let N be a marked graph and M its marking, then M is a live marking of N iff $I^T M \geq 1$ for all minimal (non-negative) s-invariants I of N . ■

Semiflows can be used to tell if a net is conservative and structurally bounded. A Petri net is conservative iff the support of the semiflow $\|I\|$ covers all the places in a Petri net.

Therefore the invariant analysis technique can detect nonconservativeness and which place is not included in the support of the semiflows as a result of nonconservativeness by the following steps:

1. Solve all the minimal s-semiflows I ;
2. Find the support of $\|I\|$;
3. If the support covers all the transitions of the Petri net, then the net is conservative. Otherwise, select the places not covered by the support.

On the other hand, t-invariants are closely related to consistency. The support of t-invariants $\|I\|$ is a consistent component. If the $\|I\|$ covers all the transitions in the net, then the net is consistent. Murata [89b] developed an algorithm to detect the inconsistency deadlock by solving all the minimal t-invariants of a Petri net and find the deadlocks by identifying the transitions which are not covered by the support of the t-invariants. The calculation of a minimal support set of generators of positive invariants is complex and these invariants are obtained by means of exponential algorithms [Martinez 82, Kruckeberg 88, Colom 91].

2.2.2 Siphons and Traps

Besides invariants, siphons and traps have importance in verifying structural properties of Petri net systems.

Definition 2.16 (Siphon and Trap)

Let $S \subseteq P$ be a set of places of a P/T net.

1. S is a *trap* iff $S^\bullet \subseteq {}^\bullet S$;
2. S is a *siphon* iff ${}^\bullet S \subseteq S^\bullet$;
3. S is a *siphon/trap-component* (st-component) iff ${}^\bullet S = S^\bullet$, i.e. S is a set of places being a siphon and trap at the same time;
4. S is called *minimal* if there is no siphon (trap) contained in S as a proper subset.

By the firing rules, siphons are sets of places which remain empty once they have lost all tokens. So it is a potential deadlock. Once it becomes empty, no transition $t \in {}^\bullet S$ can be fired or they are dead. Traps, on the contrary, are sets of places which remain marked once they have gained at least one token. Then we have the following two observations:

1. In a dead marking there is at least one empty siphon, so it is necessary for the liveness of markings that all siphons remain marked. For free choice nets, this is a necessary and sufficient condition for liveness [Esparza 92].
2. In a bounded Petri net system, a trap must be marked under a live initial marking. The reason is that if it is empty under the initial marking, once it gets a token, the system can not go back to the initial marking (state) and thus it is inconsistent. According to Theorem 2.5, the system is structurally nonlive.

The most frequently used computation method of siphons and traps was developed by Lautenbach [87b], in which siphons and traps are calculated as supports of p -semiflows of a transformed net and its time complexity is exponential. Another algorithm to obtain a generating family of siphons and traps was recently shown in [Ezpeleta 91], where siphons and traps are obtained as solutions of a set of linear inequalities.

Chapter 3 Debugging Processing Networks by Viability and Infeasibility Analysis

When an abnormal behavior is found in a model, it could be due to a modelling problem introduced during the modelling stage or a legitimate feature of the system itself. In either case, the model needs to be "debugged". The first step of debugging is to *isolate* or *localize* the cause of the problem. In a large model, the isolation can hardly be done manually. Moreover, considering the limitation of human diagnostic capacity, the sources of errors or problems located should be as small as possible with less residual information or "noise". That is, an isolation should find a portion of the model, preferably of minimal dimensions or sizes, that contains the source of the problem. The second step following the isolation is *diagnosis*, which explains the problem in terms of the modeled system. A diagnosis is good if it leads to the correction of the problem in a reasonable amount of time.

Research in model debugging techniques has been quite fruitful in the operations research field, where several infeasibility analysis techniques [Greenberg 91] have been developed to localize the source of infeasibility in a network model or a general linear programming model.

Recently, another technique, viability analysis, has been developed to localize nonviable flow problems in processing network models [Chinneck 90a, 92]. We will see in the next chapter that a processing network corresponds to a Petri net, so processing network model debugging techniques can be applied to structural debugging of Petri net models.

In this chapter, the concepts of processing networks and the techniques of processing network model debugging are presented.

3.1 Processing Network Concepts

Processing networks are modelling tools for certain real life systems which have flows of messages, materials, objects and concurrent tasks. A brief survey of the theory and application of processing networks can be found in [Koene 84] and [Chen 86].

A processing network has two kinds of nodes, *regular nodes* and *processing nodes* and the nodes are linked by arcs. A processing node is constrained by fixed ratios of flow in the terminals. There are two major processing network models developed separately by [Koene 84] and [Chinneck 90a]. Both models are capable of representing the same class of systems. In Koene's model, there are two kinds of processing nodes: a refining node (with a single inflow and multiple outflows) and a blending node (with a single outflow and multiple inflows). We use Chinneck's model in this thesis because it does not restrict the number of inflows and outflows on a processing node.

Definition 3.1 (processing networks)

A graph $G=(RV, PV; E, P)$ is called a processing network iff

1. RV is a set of regular nodes of the network on which flow conservation holds;
2. PV is a set of processing nodes of the network;
3. E is a set of arcs linking the nodes;
4. $P: E \rightarrow \mathbb{R}^+$ (positive real number) is a processing ratio function;
5. The flows X incident on a processing node are restricted by the flow ratio equation

$$x_i/x_j = p_i/p_j$$

where p_i, p_j are the processing ratios of arcs e_i and e_j ; x_i, x_j are nonnegative flows on arcs e_i and e_j .

A processing node has t terminals as shown in Figure 3.1 and it is represented by a set of $t-1$ independent ratio equations.

FIGURE 3.1 A Processing Node [Chinneck 90a]

Figure 3.2 shows a processing network example, in which p_i , $i = 1, \dots, 6$, are processing ratios. rv_1 , rv_2 and rv_3 are regular nodes on which flow conservation equations hold, that is sum of the inflows equals the sum of outflows for a given regular node:

$$\sum_{out} x_i - \sum_{in} x_i = 0 \quad \text{for } \forall rv_i \in RV.$$

pv_1 and pv_2 are processing nodes on which flow ratio equations hold, for example,

$$x_1/x_2 = p_1/p_2.$$

FIGURE 3.2 A Processing Network

If for a processing node

$$\sum p_i^- = \sum p_i^+$$

holds, where p_i^- is the processing ratio of an inflow arc and p_i^+ is the processing ratio of an outflow arc, then the processing node is flow-conserving. A processing network is said to be a **pure processing network** iff all its processing nodes are flow-conserving. Otherwise the processing network is said to be a nonconserving processing network. Figure 3.3 gives an example of a nonconserving processing node. A nonconserving processing network can be converted to a pure processing network by introducing artificial environment nodes [Chinneck 92].

In both pure processing networks and nonconserving processing networks, flow conservation always holds at regular nodes.

FIGURE 3.3 A Nonconserving Processing Node

3.2 Processing Network Representation of LPs

One of the most important advantages of processing networks is their power in visualizing a system of linear equations [Koene 84]. This property, as we will see in Chapter 4, makes it possible to transform a Petri net into a processing network in order to use processing network model debugging techniques. In this section, we discuss how to convert a general LP into a processing network.

Definition 3.2 (Bipartite network)

1. A processing network is said to be a **bipartite processing network** iff all the arcs are those linking a processing node to a regular node and there are no arcs between processing nodes or between regular nodes.

The following theorem shows that a linear program (LP) can be converted to a bipartite processing network.

Theorem 3.1 (Transformation of an LP to a processing network)

A linear program model of

$$\begin{aligned} AX + S &= b \\ X, S &\geq 0, \end{aligned} \tag{3-1}$$

where S is a vector of slack variables, can be transformed into a bipartite processing network $G=(RV, PV, E, P)$ or vice versa under the following mapping:

1. All the variables X_i and S_k are mapped to processing nodes PV_i or PS_k in G , where $i = 1, \dots, n$; n is the number of variables, $k = 1, \dots, m$; m is the number of constraints.

2. Each row of equation j is represented by a regular node R_j , where the number of regular nodes equals to m , the number of equations in the model;
3. For each a_{ij} in A , if $a_{ij} < 0$, then there is an edge e_{ij} from a regular node RV_i to processing node PV_j with the processing ratio w_{ij} equal to the absolute value of a_{ij} ; if $a_{ij} > 0$ then there is an edge from processing node PV_j to regular node RV_i with the processing ratio w_{ij} equal to a_{ij} . For each PS_k representing S_k there is an arc from the PS_k to RV_k representing the constraint i where S_k belongs and the processing ratio of the arc is 1.
4. If $b_j > 0$ then RV_j is a source with flow supply b_j , if $b_j < 0$ then RV_j is a sink with flow demand b_j .

Proof: According to the mapping, let the flow through a processing node PV_i be X_i , and the flow of an arc linking RV_j and PV_i is $a_{ij}X_i$. Then we have the flow equations of the processing network

$$AX + S = b$$

where A is the incidence matrix of the network and X is a vector corresponding to the processing nodes. This is exactly the same as the model (3-1). ■

For example, given the linear program in (3-2):

$$\begin{aligned} 2X_1 - X_2 + 5X_3 &\leq 3 \\ X_1 + 5X_2 - X_3 &\leq 1 \end{aligned} \quad (3-2)$$

By adding slack variables, (3-2) is transformed to:

$$\begin{aligned} 2X_1 - X_2 + 5X_3 + S_1 &= 3 \\ X_1 + 5X_2 - X_3 + S_2 &= 1 \end{aligned} \quad (3-3)$$

And model (3-3) can be transformed to a processing network according to the above mapping rules as shown in Figure 3.4 in which dash outflows stand for sinks (dash inflows stand for sources).

FIGURE 3.4 A processing network representation of an LP

For a set of linear homogeneous equations or inequalities, there are no source and sink regular nodes in the transformed processing network. If the flow variables have nonzero lower bound and finite upper bound, the transformation is the same as above except each arc has a capacity restriction. For example, if the variable X_1 in model (3-3) is bounded by $1 \leq X_1 \leq 3$, then the flow capacity range of the arc from " X_1 " to " $row1$ " is $[2, 6]$.

3.3 Localization of Infeasibility by IIS Technique

Once a processing network has no feasible flow, we want to know what and where the problem is. Since a processing network and a general LP are mutually convertible, we can use infeasibility analysis of LPs to isolate and diagnose the problem. In this section, infeasibility analysis techniques are introduced.

The first step of infeasibility analysis is the isolation of a source which causes global infeasibility. There are various techniques to achieve this. Phase 1 dual price methods are useful but cannot give an isolation less than the entire linear system of equations [Murty 83]. For easy diagnosis, we want to minimize the number of equations in an infeasible subset or localize the cause to as few as possible.

Definition 3.3 (IIS)

An *irreducibly inconsistent system (IIS)* is a minimal set of inconsistent constraints. [van Loon 81]

By this definition, *IIS* is an ideal source of infeasibility to be diagnosed. Chinneck and Dravnieks [91] have developed filtering algorithms, which are implemented in *MINOS(IIS)*, the only software able to isolate *IIS*s. The filtering algorithms obtain a minimal infeasible set by gradually eliminating constraints from the original LP model until those remaining constitute a minimal infeasible set. There are three component filtering routines which are integrated into an overall algorithm.

Given an infeasible LP, *deletion filtering* tests the effect on the feasibility of the model of removing a single constraint. If the reduced LP is infeasible, then remove the constraint permanently; if the reduced LP becomes feasible, then return the constraint to the LP. Continue in this fashion until all of the constraints have been tested once. This deletion filtering method will find exactly one *IIS* in an infeasible LP.

Elastic filtering makes use of elastic programming, in which all constraints in the model are provided with elastic variables which permit them to “stretch” to provide a feasible solution. An iterative procedure enforces some constraints back to nonelastic form by removal of the elastic variables, eventually resulting in a set of nonelastic constraints which are collectively infeasible; this set contains at least one *IIS*. This method can eliminate large numbers of uninvolved constraints quickly.

Sensitivity filtering uses the fact that stretching a constraint is equivalent to altering the right hand side (*rhs*) of the constraint. The phase I or elastic solution will show sensitivity to an infinitesimal adjustment of the *rhs*'s of some of the *IIS* constraints, but never to the *rhs* of a non-*IIS* constraint. Basically, if a constraint or variable bound has a nonzero shadow price or reduced cost, then it must be part of some *IIS*. The constraint set identified by sensitivity filtering contains at least one *IIS*. Sensitivity filtering is very inexpensive, requiring only the calculation of shadow prices and reduced costs; its major advantage is the immediate elimination of large number of uninvolved constraints.

These techniques have been implemented in *MINOS(IIS)* version 4.01 [Chinneck 93]. *MINOS* [Murtagh 87] is used to test if a model is feasible by Phase I LP. Once the model is found to be infeasible, then *MINOS(IIS)* is invoked to localize the *IIS*s.

3.4 Localization of Nonviability

Nonviability is a kind of network model error, which was first defined for processing networks by Chinneck [90]. A network model is usually designed to transfer flow along its arcs. If the flow in one or more of the arcs is restricted to zero solely by the structural relationships of the model, then it is *nonviable*. That is, it is *nonviable* iff, when all arc flows are restricted to the range $(0, \infty)$, $\exists e \in E$ s.t the flow of e , x_e , is always restricted to zero. By this definition, we have the following theorem:

Theorem 3.2 [Chinneck 90a]

A processing network is viable iff it is feasible for all variables to be positive simultaneously. ■

To test whether a processing network is viable we let all arc flows have a lower bound of some arbitrary small positive number, for example 1, and no upper bound and then use the phase 1 simplex method [Chvatal 83] to test the feasibility of the transformed model:

$$\begin{aligned} AX &= 0 \\ X &\geq 1 \end{aligned} \tag{3-4}$$

Mathematically speaking, nonviability of a system of homogenous linear equations is due to one of the following two reasons: equation rank nonviability or constraint nonviability defined below.

Theorem 3.3 [Porter 66]

A set of homogeneous linear equations has zero solution iff it has full rank. ■

Definition 3.4 (Equation rank nonviability)

Nonviability due to full rank is called *equation rank nonviability*.

Definition 3.5 (Constraint nonviability)

In a system of homogeneous linear equations:

$$\begin{aligned} AX &= 0 \\ X &\geq 0 \end{aligned} \tag{3-5}$$

with rank r less than the number of variables ($r < n$), if there is a variable x_0 which is a linear combination of r independent variables such that

$$x_0 = k_1x_1 + k_2x_2 + \dots + k_rx_r \quad (3-6)$$

where $k_i < 0$ ($i=1, \dots, r$), then x_0 is forced to zero because of the nonnegativity constraints. Nonviability caused by the nonnegativity constraints as defined in (3-6) is called *constraint nonviability*.

Therefore, from (3-5) we can tell that equation rank nonviability and constraint nonviability are the only two possible causes of nonviability.

Topologically speaking, nonviabilities of pure processing networks are due to a number of classes of "bad" structural relations such as directed cutsets, full rank cycle and induced cutsets [Chinneck 90a].

There are three techniques to localize nonviability in processing network, one for pure processing networks [Chinneck 90a], and two for nonconserving processing networks [Chinneck 92]. Here the one that localizes the nonviability by finding *IIS*s is introduced because it is more general. Theorem 3.2 implies a method of using the existing *IIS*-identification tool *MINOS(IIS)* for nonviability localization. Simply add the positivity constraints to the structural equations defining the processing network and use *MINOS(IIS)* to identify an *IIS*. The constraints in the *IIS* will constitute a nonviable set which is called an *irreducible nonviable system* or *INS* for short. The algorithm is given below:

1. Change a processing network model

$$\begin{aligned} AX &= 0 \\ L &\leq X \leq U \end{aligned} \quad (3-7)$$

by setting the lower bound of the flow variables to one and upper bound to infinity:

$$\begin{aligned} AX &= 0 \\ X &\geq 1 \end{aligned} \quad (3-8)$$

2. Test the feasibility of (3-8) by Phase I LP;

3. IF (3-8) is infeasible THEN

Localize the *IIS* in (3-8)

This is the *INS* of model (3-7) when the positivity constraints are replaced by the original nonnegativity constraints.

This algorithm has been implemented in a processing network model debugging tool called PROFLOW [Chinneek 90b] which has a user friendly interface and a specialized language for describing processing networks. PROFLOW translates the specifications of a processing network model into a MPS file which can be used by *MINOS(II)* to localize the *INS*s.

Chapter 4 Processing Network Models of Petri Nets

In this chapter, we will show that where the static structural properties of a Petri net are concerned, the Petri net can be mapped to a processing network.

From the modeling point of view, Petri nets are different from processing networks. The behavior of a Petri net model depends not only on the structure but also on the markings which define the states of the system. The flows of tokens are lumpy, discrete and nondeterministic in terms of the nondeterministic firing choice. On the other hand, the flows are deterministic in terms of the underlying net structure such that some movements of tokens are never allowed in a net. Petri nets model both structure and behavior of concurrent systems, such as parallelism, choice and conflict (see Figure 4.1 in which transitions t_j and t_k are in conflict since firing either will remove the token from p_i , disabling the other transition). By comparison, the flows in a processing network are steady, continuous and deterministic. Once the structure of a processing network is given, all the feasible flows are defined as well.

FIGURE 4.1 Conflict

From the structural analysis point of view, processing networks are similar to Petri nets. Chapter 2 showed that static structural properties of Petri nets can be represented by a system of linear equations or inequalities which can be visualized as a processing network as well. Mathematically, the deterministic semiflows in Petri nets are the same as the steady flows in a processing network, as we can see from the definition of the semiflow. The structural analysis of Petri nets can also be viewed as describing the behavior of token movements by underlying semiflows.

Moreover, given the topological similarities between the two, such as both processing nodes and transitions having "AND" logic and both obeying flow ratio equations, the transformation of a Petri net structure to a processing network is quite straightforward.

Because a processing network can represent an LP, the structural equations or semiflows of a Petri net can be represented by a processing network as well.

Given a Petri net $PN=(P,T,F,W)$, its structural equation is

$$AX=0$$

where A is the incidence matrix or flow matrix of PN and X is a t-vector of flow variables of PN .

Definition 4.1 (feasible semiflows)

Given a Petri net $PN=(P,T,F,W)$ and its incidence matrix A ,

1. A t-vector X is said to be a *feasible t-semiflow* iff

$$\begin{aligned} AX &= 0 \\ X &\geq 0 \end{aligned} \tag{4-1}$$

holds.

2. An s-vector Y is said to be a *feasible p-semiflow* iff

$$\begin{aligned} A^T Y &= 0 \\ Y &\geq 0 \end{aligned} \tag{4-2}$$

holds.

The t-semiflow equations (4-1) of a Petri net can be mapped to a processing network $G=(RV,PV, E,W')$ as follows:

1. Replace each place p by a regular node such that $RV=P$
2. Replace each transition t by a processing node such that $PV=T$;
3. Replace each arc in PN by an arc in the processing network such that $E=F$;
4. Replace the weight function by the processing ratio such that $W'=W$.

Figure 4.2 is an example of the transformation between a Petri net and a processing network.

The structural equations (4-1) of this Petri net and the processing network are identical to each other. Same mapping techniques can be applied to the p-semiflow equations (4-2) of a Petri net by first converting the net into its reverse and dual net and then mapping the reverse and dual net into a corresponding processing network by the above rules. Once the mapping is done, the t-semiflow (or p-semiflow) is the same as the steady flow in the processing network.

FIGURE 4. 2 Transformation between a Petri net and a Processing Network

(a) A Petri Net

(b) A Processing Network

$$\begin{aligned} rv_i &= p_i \text{ for } i=1, \dots, 5; \\ pv_i &= t_i \text{ for } i=1, 2; \end{aligned}$$

Chapter 5 Analyzing Petri Net Structure by Viability and Infeasibility Analysis

We saw in the last chapter that Petri net structural properties can be mapped to processing network flow models. The transformation paves the way for the application of viability and infeasibility analysis methods to prescreen and localize certain structural problems such as structural nonliveness and unboundedness. In this chapter, we define viability and infeasibility in Petri nets and relate them to static structural properties. The nonviability and infeasibility localization algorithms are designed for Petri net model debugging. If a Petri net model has structural problems, the algorithms can tell modelers or designers where the problems are. To assist diagnosis, some bad structures (subclasses of siphons and traps) are studied to interpret why these structures cause nonviability and infeasibility of Petri net models.

5.1 Nonviability and Infeasibility Interpretation of Structural Problems in Petri Nets

Both semiflows in a Petri net and flows in a processing network are deterministic. Once the structures are given, their feasible flow solutions are defined. In this section we use this similarity to interpret structural properties of Petri nets by viability and infeasibility theory.

Given the system of linear equations of a t-semiflow in a Petri net $PN=(P, T, F, W)$:

$$\begin{aligned} AX &= 0 \\ X &\geq 0 \end{aligned} \tag{5-1}$$

where A is the incidence matrix of the net and X is a t-vector of flow variables, we can define nonviability as follows.

Definition 5.1 (Nonviability of Petri nets)

A P/T net $PN=(P, T, F, W)$ is said to be **nonviable** if its t-semiflow equations (5-1) are nonviable, i.e. certain components of the t-semiflow are restricted to zero. Mathematically, $\exists t \in T$ such that $j(t)=0$ in all J for $AJ=0, J \geq 0$.

Nonviability is a global symptom of a Petri net model. Loosely speaking, the “victim” transition $t_{j(t)=0}$ is called a **nonviable transition**.

The same definition can be applied to the dual and reverse net of PN, i.e., a Petri net is said to be **dual nonviable** iff its p-semiflow equations

$$\begin{aligned} A^T Y &= 0 \\ Y &\geq 0 \end{aligned} \quad (5-2)$$

are nonviable, i.e., certain components of the p-semiflow are restricted to zero. Mathematically, $\exists p \in P$ such that $i(p)=0$ in all I for $A^T I=0, I \geq 0$.

In accordance with the definitions of consistency and conservativeness in Table 2.1, it is easy to make the following observations:

1. A Petri net is nonviable iff it is inconsistent;
2. A Petri net is dual nonviable iff it is nonconservative.

Definition 5.2 (Structural Infeasibility)

A Petri net $PN=(P,T,F,W)$ is said to be **structurally infeasible** iff the system of linear inequalities

$$\begin{aligned} AX &\leq 0 \\ X &> 0 \end{aligned} \quad (5-3)$$

is infeasible, where A is the incidence matrix of PN and X is a t-vector of flow variables.

Similarly, we define that a Petri net is **structurally dual infeasible** iff the system of linear inequalities of its reverse and dual net

$$\begin{aligned} A^T Y &\leq 0 \\ Y &> 0 \end{aligned} \quad (5-4)$$

is infeasible, where A is the incidence matrix and Y is an s-vector of flow variables.

Comparing formula (5-1) with formula (5-3), we know that if a Petri net is structurally infeasible it must be nonviable. So in a structurally infeasible net, there are also some transitions on which have zero component in all t-semiflows. These transitions are nonviable transition as well. Comparing the definitions of structural boundedness and repetitiveness

in Table 2.1 with the definition of structurally infeasible, we make the following observations:

1. A Petri net is structurally infeasible iff it is nonrepetitive;
2. A Petri net is structurally dual infeasible iff it is structurally unbounded.

The structural inequalities of a Petri net (5-3) can be converted to a processing network as well by adding a slack variable to each row of the inequalities as shown in the following equations:

$$\begin{aligned} AX+S &= 0 \\ X > 0, S &\geq 0 \end{aligned} \quad (5-5)$$

In this processing network, the lower bound of flow in the arcs incident to the processing nodes corresponding to X is no longer zero. If it has feasible flow, then the corresponding Petri net is structurally feasible or repetitive. Similarly if the following processing network model (5-6) has feasible flow then the corresponding Petri net is structurally bounded.

$$\begin{aligned} A^T X+S &= 0 \\ X > 0 \text{ and } S &\geq 0 \end{aligned} \quad (5-6)$$

So an immediate result of these observations is that the nonviability and infeasibility algorithms for processing networks can be used to detect and localize inconsistency, nonconservativeness, nonrepetitiveness and structural unboundedness in Petri net models.

Once we find that a processing network model (5-5) or (5-6) is infeasible, the ISS causing the structural unboundedness and nonrepetitiveness in the corresponding Petri net can be isolated by the techniques in section 3.3.

5.2 Structural Debugging of Petri Net Models

The interpretation of structural problems of Petri net models by nonviability and infeasibility theory suggests that a processing network model debugging tool like PROFLOW can be applied to Petri net model debugging. In this section, algorithms for structural debugging of Petri net models are presented.

5.2.1 Algorithms

The debugging algorithms given below detect and localize structural problems of Petri net models. Once a Petri net model is checked by the algorithms, the output model is without static structural problems. This is the major contribution of viability and infeasibility analysis to Petri net model debugging.

There are four algorithms for structural debugging of Petri net models. A modeler can select which algorithms he/she wants in accordance with the kind of structural property he/she is interested in. These four algorithms can be run in parallel or sequentially.

Algorithm 1: Checking and locating inconsistency problems

1.1 Convert $PN=(P, T, F, W)$ to a processing network $G=(P, T, F, W)$ of model (5-1);

1.2 Test viability of the model

IF G is viable THEN

stop

ELSE

localize the *INS*;

fix the problem and modify the net PN ;

go to 1.1;

Algorithm 2: Checking and locating nonconservative problems

2.1 Convert $PN=(P, T, F, W)$ to its reverse-dual net $PN_{rd}=(T, P, F^{-1}, W)$

2.2 Convert PN_{rd} to a processing network $G=(T, P, F^{-1}, W)$ of model (5-2);

2.3 Test viability of the model

IF G is viable THEN

stop

ELSE

localize the *INS*;

fix the problem and modify the net PN ;

go to 2.1;

Algorithm 3: Checking and locating nonrepetitive problems

- 3.1 Convert $PN=(P,T,F,W)$ to a processing network $G=(P \cup S, T', F', W')$ of model (5-5);
- 3.2 Test the feasibility of the model;
 - IF G is feasible THEN
 - stop
 - ELSE
 - localize the IIS ;
 - fix the problem and modify the net PN ;
 - go to 3.1;

Algorithm 4: Checking and locating structurally unbounded problems

- 4.1 Convert $PN=(P,T,F,W)$ to its reverse and dual net $PN_{rd}=(T,P,F^{-1},W)$;
- 4.2 Convert the PN_{rd} to a processing network $G=(P \cup S, T', F', W')$ of model (5-6)
- 4.3 Test feasibility of the model
 - IF G is feasible THEN
 - stop
 - ELSE
 - localize the IIS ;
 - fix the problem and modify the net PN ;
 - go to 4.1

5.2.2 Advantages of the Algorithms

Compared with using invariants to find inconsistency or nonconservativeness problems, the method we use here is more efficient because it can locate problems in polynomial time [Chinneck 91].

Another major difference between our technique and those already known in the Petri net community is that our technique localizes a source which causes problems instead of find-

ing only “victims” of the problems. For example, Murata’s technique [Murata 89b] finds all the transitions which are not covered by t-invariants, but the real problem is why they are not covered. It is a bad structure embedded in the net that causes the problem. The technique developed here can find the bad structure (*INS* and *IIS*) and give the modeler structural information for model debugging. If no change is made to the *INS* or *IIS*, the model will never function as expected.

Furthermore, if there are several independent structural errors in a Petri net model, the invariant technique can not separate them into different groups. The resulting error message is quite confusing as we can not tell which “victim” is the result of which specific bad structure. The technique we use here is able to classify all the nonviable transitions into different groups. Therefore, it can guide debugging of Petri net models.

If all *INS*s or *IIS*s in a Petri net are independent of each other, we have to find all of them and fix them separately and the order is not important. In real cases, several *INS*s or *IIS*s are usually dependent on each other or overlap. Thus it is better to find an *INS* or *IIS* which is most easily diagnosed (normally the one with the fewest rows) and our technique can help here [Chinneck 93].

5.2.3 Implementation

The implementation of this Petri net model debugging algorithm is based on *MINOS(IIS)* 4.01 and *PROFLOW*. We use *GSPN* to draw a Petri net model. When a Petri net model is debugged, the first step is to convert the Petri net into a corresponding processing network model according to transformation rules in Chapter 4. This is automatically done by a transformation program and the result is analyzed by *PROFLOW*. Once an *INS* or *IIS* is found by *PROFLOW*, it is mapped back to the corresponding subnet in the original Petri net. One can then diagnose manually in accordance with the principles of the modeled system. In the following example, we see the inputs and outputs of the tools.

Example 5.1

Given the Petri net in Figure 5.1, the transformation program converts it to a processing network described in *PROFLOW* language. When the model is debugged by *PROFLOW*, we get the *INS* listed in Figure 5.2.

FIGURE 5.1 A nonviable Petri net

In Figure 5.2, the edges on which flow is forced to zero are called lower bound columns and $t6[K2:K1]$ means the terminals $K1$ and $K2$ on transition $t6$. The information in Figure 5.2 can be represented as a graph as shown in Figure 5.3

FIGURE 5.2 *INS* generated by nonviability

----- Columns -----
edge(t1,P2) lower bound
edge(t3,P6) lower bound
edge(t2,P5) lower bound
----- Rows -----
t6[K2:K1] fixed value
t5[K2:K1] fixed value
t4[K2:K1] fixed value
P6 fixed value
P5 fixed value
P2 fixed value

5.3 Nonviability, Infeasibility and Siphons and Traps

To assist in problem diagnosis, we classify common bad structures of Petri nets and show that nonviability and infeasibility are closely related to subclasses of siphons and traps in this section.

Definition 5.3 (pure trap)

A trap S is called a *pure trap* if $\bullet S S \bullet$ is not empty.

A pure trap might get a token from another part of the net under some initial marking through a certain firing sequence. This is dangerous if structural boundedness and structural liveness are desired. To find all of the minimal siphons and traps is of exponential time complexity, but viability analysis can find certain pure traps efficiently.

Theorem 5.1

If a trap S in a Petri net $PN=(P, T, F, W)$ is a pure trap and the subnet $PN'=(S, S \bullet, F', W')$ is conservative, then the trap S causes nonviability and infeasibility, where $F'=S \times S \bullet \cup S \bullet \times S$, W' is a weight vector of F' .

Proof: As S is a pure trap, all the arcs linking the places in S with the transitions outside of the subnet PN' are incoming arcs. So if any transition $t \in \bullet S S \bullet$ fires, the subnet PN' gets tokens from the rest of the net. Let $M_0(S)$ and $M_1(S)$ denote the markings of the PN' before and after any transition $t \in \bullet S S \bullet$ fires. By the definition of conservativeness, $M_0(S)i = M(S)i = a \text{ constant}$, where i is an s -vector of positive integers, holds in PN' for any reachable marking from $M_0(S)$ by firing solely the transitions belonging to $S \bullet$. Because some components of $M_1(S)$ are greater than the corresponding components in $M_0(S)$ and the remaining components of these two marking vectors are the same, then we have $M_0(S)i \neq M_1(S)i$. Therefore once a $t \in \bullet S S \bullet$ fires, the net can not go back to its original marking, i.e., the transitions belonging to $\bullet S S \bullet$ are not covered by the support of t -invariants. The net is nonviable (or inconsistent) by definition.

Where feasibility is concerned, we simply add a slack transition to each place in S . All the arcs to the places in S from the rest of the net to the subnet are still incoming arcs. So S is still a pure trap and by the same proof we can find that all the transitions in $\bullet S S \bullet$ are not covered by the support of t -invariants. As long as there is one transition belonging to $\bullet S S \bullet$ in the original net, then the net is structurally infeasible. ■

Example 5.2

The Petri net in Figure 5.1 is an example of nonviability due to a pure trap $\{P2, P5, P6\}$. The subnet that $P2, P5, P6, t4, t5$ and $t6$ formulate is a conservative one. So the net is non-

viable and the flows on transitions $t1, t2, t3$ are forced to zero. The *INS* is shown in Figure 5.3.

FIGURE 5.3 An *INS* of Figure 5.1

Definition 5.4 (Generating and absorbing cycles)

If there is a cycle in a Petri net $PN=(P,T,F,W)$ such that $|\bullet p|=|p\bullet|=1$ for all the places on the cycle, then the set of the places on the cycle is an st-component S . If the weights of the arcs on the cycle have the following relation:

$$\prod_{i=1}^{|S|} \frac{W(\bullet p_i, p_i)}{W(p_i, \bullet p_i)} > 1 \quad \text{for } \forall p_i \in S$$

then the cycle is called a **generating cycle**; if

$$\prod_{i=1}^{|S|} \frac{W(\bullet p_i, p_i)}{W(p_i, \bullet p_i)} < 1 \quad \text{for } \forall p_i \in S$$

holds, then the cycle is called an **absorbing cycle**.

Theorem 5.2

If there is a generating cycle or an absorbing cycle $(S, S\bullet, F', W')$ in a Petri net $PN=(P,T,F,W)$, where $F'=S \times S\bullet \cup S\bullet \times S$, $S \in P$ and W' is weight vector of F' , is an st-component, then the net is nonviable.

Proof: For a place $p_k \in S$, we have the flow relationship $X_{out}^k - X_{in}^k = 0$, $k=1, \dots, |S|$.

Because on the transition $t_k = \bullet p_k$, we have $X_{in}^k = q_k X_{out}^{k-1}$, where

$q_k = (W(\bullet p_k, p_k)) / (W(p_{k-1}, \bullet p_{k-1}))$, then we have the following formula:

$$X_{in}^k = X_{in}^1 \prod_{i=1}^{|S|} q_i$$

As the product of q_i is greater than 1 (or less than 1) for a generating cycle (or an absorbing cycle), all the X^k are forced to zero, i.e., all the transitions on the cycle are nonviable transitions and the net is nonviable. ■

Example 5.3

Figure 5.4 is an example of a nonviable Petri net due to a generating cycle, where $\{P1, P2, P3\}$ is an st-component. All the transitions are nonviable transitions and the whole net is an *INS*.

FIGURE 5.4 A nonviable P/T net due to a generating cycle

On the other hand, if a siphon can lose tokens even though it is marked under the initial marking, then this is dangerous when structural liveness and boundedness are desired. Our nonviability analysis technique can localize this kind of siphon efficiently.

Definition 5.5 (pure siphon)

A siphon S is said to be a *pure siphon* if $S \bullet \bullet S$ is not empty.

Theorem 5.3

Let S be a pure siphon in a Petri net $PN=(P, T, F, W)$. If $(S, \bullet S, F', W)$ is a conservative subnet, where $F'=S \times \bullet S \cup \bullet S \times S$ and W' is weight vector of F' , then the net is nonviable and all the transitions in $\bullet S \setminus S \bullet$ are nonviable transitions.

Proof: Similar to the proof of theorem 5.1. Simply consider firing any transitions belonging to $\bullet S \setminus S \bullet$, the subnet will lose tokens. ■

The difference between a pure siphon and a pure trap is that if a pure trap causes nonviability, then it also causes structural infeasibility. But a pure siphon does not cause infeasibility as it can get as much flow as it wants through the slack variables (corresponding to transitions) in order to let the flow in the transitions $\bullet S \setminus S \bullet$ be positive.

Example 5.4

Figure 5.5 is an example of a nonviable Petri net due to a pure siphon $\{P3, P4\}$. From the *INS* in Figure 5.6, we can see that the flow in arc $(P3, t2)$ linking to a conservative component $(P3, t3, P4, t4)$ is forced to zero.

FIGURE 5.5 A nonviable P/T net due to a pure siphon

FIGURE 5.6 An *INS* of Figure 5.5

Note that not all traps or siphons cause nonviability. For example, the Petri net in Figure 5.7 is viable even though $\{P1, P2, P3\}$ and $\{P4, P5\}$ are st-components and $\{P1, P2, P5\}$ is a siphon. The reason might be due to the subnet $(S, S^\bullet, F')$ or $(S, \bullet S, F')$ being nonconservative.

By viability analysis and the siphon and trap solving techniques mentioned in Chapter 2, we can differentiate two kinds of traps and siphons: those causing nonviabilities or infeasibilities and those which do not. A bounded Petri net system can not include those traps and siphons which cause nonviabilities in order to be live. Finding the minimal siphons and traps which cause nonviability by viability analysis technique is more efficient than the method of finding general minimal siphons and traps which is of exponential time complexity. In this sense, *the viability analysis technique is an efficient method to find certain subclasses of siphons and traps which cause nonviability or infeasibility.*

FIGURE 5.7 A viable P/T net with traps and a siphon

5.4 Experiments

In this section, experiments are done in two aspects: finding all the four kinds of structural problems using the structural debugging algorithms to show how they work; and localizing inconsistency deadlocks in a large net is taken as an example to show that our algorithms can find all the sources of structural problems (usually finding a smaller *INS* or *IIS* first).

5.4.1 Check Nonrepetitiveness

Since a Petri net is consistent only if it is repetitive, we have to fix the nonrepetitive problems first. Before using viability analysis techniques to detect and localize problems, the original Petri net model of Figure 5.8 is transformed to a processing network model (5-5) with slack variables (slack processing nodes $t_9 - t_{16}$) in Figure 5.9. The *IIS* of the model is the bold part of Figure 5.9, which is a *pure trap* $\{P_7, P_8\}$ in the original Petri net. If we remove the arc (t_6, P_7) , then the new processing network model (5-5) becomes feasible or the Petri net is structurally feasible.

FIGURE 5.8 A Petri net with structural problems

FIGURE 5.9 A processing network model (5-5) of the P/T net in Figure 5.8

5.4.2 Check Inconsistency

When checking inconsistency, the processing network model of (5-1) is exactly the same as the Petri net in Figure 5.8. The *INS* found by PROFLOW is shown in Figure 5.10, where the transitions *t5* and *t6* are nonviable. The problem is due to a pure siphon {*P2*, *P3*, *P4*, *P5*} in the original Petri net. When we reverse the arc of (*P3*, *t6*) then the net is viable or consistent.

5.4.3 Check Structural Unboundedness

Given the Petri net model in Figure 5.11, when checking structural unboundedness, the net is converted to its reverse and dual net first, which then is transformed to a processing network model of (5-6) by adding slack variables (processing nodes *P9* - *P16*) as in Figure 5.12. The *INS* (bold in Figure 5.12) found by PROFLOW is a sink node *t5* in the processing network. This corresponds to a pure trap {*t5*} in the reverse and dual net. So the structural relation of (*P2*, *t5*, *P6*) in the original net is the cause of structural unboundedness. If the arc of (*t5*, *P2*) is reversed in the original net, then the net is structurally bounded.

FIGURE 5.10 An *INS* of the P/T net in Figure 5.8

FIGURE 5.11 A Petri net model

5.4.4 Check Nonconservativeness

After replacing arc $(t5, P2)$ by $(P2, t5)$ in Figure 5.11, the viability test follows. The reverse and dual net of the new P/T net is nonviable, i.e., the original net is nonconservative. The *INS* found by the tool in the reverse and dual net is shown in Figure 5.13, in which $P4$ is reported as a nonviable transition. The problem is caused by an absorbing cycle in the *INS*. If the weight on the arc $(P4, t4)$ is changed to 1, then the net is dual viable or conservative.

FIGURE 5.12 The processing network model (5-6) of Figure 5.11

FIGURE 5.13 The Dual INS of the P/T net in Figure 5.11

5.4.5 A Localization Example

The net with 36 places, 66 transitions and 260 edges whose structure is shown in Table 1 in *Appendix A* is the basis of the example. Thirty independent nonviabilities, as listed in

Table 2 in *Appendix A*, were introduced one at a time. And 30 *INS*s resulted by the problems introduced are shown in the table. Then we put all of them into the net at the same time and used the inconsistency check algorithm to localize the nonviabilities. When a source is located, we take the action to fix the problem according to the problem information in *Appendix A* Table 2, then locate the next problem. In the end, we located 23 of them. The others were no longer nonviabilities because of the interaction of the structures (e.g. problems 15 and 16 do not cause nonviability when they are put into the net together) makes them logical errors instead of structural problems. The result of the localization shows that our method locates a small source first (see Table 3 in *Appendix A*).

5.5 Diagnosis of Nonviability

Once a nonviable set or an infeasible set is located, the diagnosis operation follows. For different problems or systems, the interpretation of *INS*s or *IIS*s is different and it depends on the real problem in the original system. The problems detected by the structural debugging tool could be modeling problems or design problems. For the former kind of problems, we can check some simple modeling errors before looking into more complicated errors resulted by misunderstanding the modeled system. These simple errors include: (1) missing an outflow (or inflow) from a subnet which is reported as an *INS* or *IIS* caused by a pure trap (or siphon); (2) adding an extra outflow (or inflow) to a subnet which is reported as an *INS* or *IIS* caused by a pure siphon (or trap).

In this section, two case studies are done to show how the localization technique helps the problem diagnosis.

5.5.1 A Case of a Reader/Writer System

Figure 5.14 is a Petri net model of a simple version of the reader/writer problem from [Genrich 81]. Assume there are three readers which may share the resource and only one writer which can use the resource exclusively.

Let $H(u)$ stand for the state that user u has nothing to do with the resource;

$W(u,m)$ stand for the state that user u wants to use the resource exclusively;

$U(u,m)$ stand for the state that user u is using the resource exclusively;

$D(u,m)$ stand for the state that user u has finished using the resource exclusively;
 R stand for the number of times the resource is still available for shared usage;
 s stand for the shared mode and e stands for the exclusive mode;
 a, b, c stand for three readers/writers;

FIGURE 5.14 A Petri net model of a reader/writer system

Unfortunately, the modeler made a mistake by missing the arc from the transition $3ae$ to the place R , i.e., the user a does not send an acknowledge message after he/she finishes using the resource exclusively. When the model is debugged, an *INS* is obtained as in Figure 5.15 in which the flow in the arc $(R, 2ae)$ is forced to zero or the transition $2ae$ is non-viable. The structure of the *INS* is a siphon which causes deadlock, i.e., once the transition $2ae$ fires (user a using the resource exclusively), the system can not go back to the original marking under which the resource is available for one of the three users to access exclusively, and it will eventually deadlock. To fix the *INS*, we have to add one more arc from a

transition outside of the *INS* to a place in the siphon. This means that user *a* should send acknowledge information back once he finishes using the resource. After adding arc $(3ae, R)$ the model is viable.

FIGURE 5.15 An *INS* of the reader/writer model

5.5.2 A Case of a Deadlock in a User Server Protocol

As a second example in Figure 5.16, we choose a user server protocol from [Sajkowski 86]. When the protocol is verified by viability analysis, it shows that the protocol model is nonviable or the protocol has deadlocks. The debugging tool localizes two *INS*s, which are shown in Figure 5.17 and Figure 5.18. Both of them have a siphon which causes deadlock. The first *INS* shows that the “lossack” and “lossalarm” are two nonviable transitions, which means once these two transitions fire or the two events happen, the protocol will deadlock eventually. The same information can be found in the second *INS* which shows that the “lossreq” and “lossdone” are two nonviable transitions. Once they fire or these two events occur, the protocol will eventually deadlock. This example shows that our techniques can prescreen certain faults existing in communication protocols and thus can be applied to protocol verification.

FIGURE 5.16 The Petri net model of a user server protocol **FIGURE 5.17 An /NS of the protocol model**

FIGURE 5.18 Another *INS* of the protocol model

Chapter 6 Compositional Viability and Infeasibility Analysis of Large Petri Nets

A large Petri net model is usually composed of a number of smaller Petri net models with their own functions. When such models are tested, we may find that although each subsystem is free of structural problems, the global system still has structural problems due to the interaction between the subsystems. Also, the viability and infeasibility analysis may end up with a very large *INS* or *IIS*, which is difficult to understand if the analysis is done in the global context. So we need to know (1) what kind of composition can preserve the good structural properties, such as consistency and conservativeness, from subsystems to the global system; (2) If the global system is nonviable or infeasible, is there any more effective way to localize a source to a subnet which is as small as possible to assist diagnosis of the problems?

Decomposition is an approach to answer these questions. In this chapter we will show that (1) For the purpose of studying preservation of structural properties, a global system can be broken down into a number of subsystems communicating with each other through communication media. Also some special communication media, which allow subnets to communicate with each other through places, will preserve good structural properties of subsystems. (2) For the purpose of locating structural problems, a global Petri net is decomposed into a number of subnets communicating with transitions. We develop a special localization algorithm for large Petri nets using the concept of the composition of Petri boxes and the monotonicity property of feasibility in a Petri net obtained by the composition of subnets via transitions.

The composition of the subsystems is often recursive or their relationship is of a tree structure. We assume the composition specification of the modules is given in the design stage. If a global Petri net system is composed of k subnet systems, it can be seen as at most $k-1$ compositions of 2 nets.

6.1 Viability and Infeasibility Analysis of Petri Nets Composed via Shared Places

A communication medium is a third part through which two nets communicate with each other. The medium could be composed solely of places, solely of transitions or of both. In

this section, we only discuss the media consisting of places and prove that this kind of medium can preserve good structural properties.

Definition 6.1 (Composition by shared places)

A Petri net $PN=(P,T,F,W)$ is said to be obtained by the composition of two nets $PN_1=(P_1,T_1,F_1,W_1)$ and $PN_2=(P_2,T_2,F_2,W_2)$ via shared places $\{P_s\}$ iff

1. $P=P_1 \cup P_2$;
2. $T=T_1 \cup T_2$;
3. $F=F_1 \cup F_2$;
4. $P_1 \cap P_2 = P_s \in P$;
5. $T_1 \cap T_2 = \emptyset$.

If two nets are composed of shared places, they interact with each other by sending and receiving tokens via the shared places. Figure 6.1(b) is obtained by the composition of two nets in Figure 6.1(a).

FIGURE 6.1 Composition of nets by shared places

To understand the structural relationship of the composition, let us define the t-semiflow equations of PN_1 and PN_2 to be

$$A_1 X_1 = 0$$

$$X_1 \geq 0$$

and

$$A_2 X_2 = 0$$

$$X_2 \geq 0$$

respectively, where A_i is the incidence matrix of net PN_i , $i=1,2$.

The t-semiflow equations of PN are

$$AX = 0$$

$$X \geq 0$$

where

$$A = \begin{bmatrix} A_{1c} & 0 \\ A_{1s} & A_{2s} \\ 0 & A_{2c} \end{bmatrix}$$

in which A_{1s} and A_{2s} are the subsets of A_1 and A_2 covering the shared places and they have the following relationship:

$$A_1 = A_{1s} + A_{1c}$$

$$A_2 = A_{2s} + A_{2c}$$

This composition preserves the good structural properties of subnet PN_1 and PN_2 as proved by the following theorem.

Theorem 6.1

If a Petri net $PN=(P,T,F,W)$ is obtained by the composition of two nets $PN_1=(P_1,T_1,F_1,W_1)$ and $PN_2=(P_2,T_2,F_2,W_2)$ via shared places $\{P_s\}$, then the following properties hold.

1. If PN_1 and PN_2 are viable then PN is viable;
2. If PN_1 and PN_2 are structurally feasible then PN is structurally feasible.

Proof: Given the above notations and relationships, $AX=0$ in PN can be represented by

$$\begin{aligned} A_{1c}X_1 &= 0 \\ A_{1s}X_1 + A_{2s}X_2 &= 0 \\ A_{2c}X_2 &= 0 \\ X_1, X_2 &\geq 0 \end{aligned}$$

As PN_1 and PN_2 are viable, then we can always find positive X_1 and X_2 which satisfy the following equations:

$$A_1 X_1 = \begin{bmatrix} A_{1c} \\ A_{1s} \end{bmatrix} X_1 = 0$$

$$A_2 X_2 = \begin{bmatrix} A_{2s} \\ A_{2c} \end{bmatrix} X_2 = 0$$

And then $X=[X_1, X_2]^T$ will make $AX=0$ hold. Therefore, PN is viable as well.

As for feasibility of Petri net models, the system of linear equations is changed to a system of linear inequalities $AX \leq 0$. The same proof follows. ■

Corollary 6.1

If a Petri net $PN=(P,T,F,W)$ is obtained by $n-1$ compositions of n subnets $PN_1=(P_1, T_1, F_1, W_1), \dots, PN_n=(P_n, T_n, F_n, W_n)$ via shared places $\{P_{s1}\}, \{P_{s2}\}, \dots, \{P_{sn-1}\}$, where $\{P_{s1}\}, \{P_{s2}\}, \dots, \{P_{sn-1}\}$ are the shared places of $n-1$ compositions of the subnets, then the following properties hold:

1. If $PN_1, \dots, PN_n$ are viable then PN is viable;
2. If $PN_1, \dots, PN_n$ are structurally feasible then PN is structurally feasible.

Proof: When $n=2$, it is true by theorem 6.1. Assume it is true for $n=k$. Let $n=k+1$ and the shared places between PN_{k+1} and PN^k (obtained by the composition of k nets $PN_1, \dots, PN_k$) be $\{P_{sk}\}$. By the assumption, PN^k and PN_{k+1} are both viable (or feasible), then the net PN is also viable (or feasible) by theorem 6.1. ■

The above corollary tells us that the composition by shared places will preserve the structural properties of subnets. For example, if $PN_1, \dots, PN_n$ are consistent, PN is consistent as well. The same conclusion can be applied to its reverse and dual net to define the preservation of dual viability and feasibility by shared places.

These results suggest that we do not need to do any viability or infeasibility testing in the global net obtained by the composition of n subnets via shared places as long as each subnet is viable or feasible.

6.2 Viability and Infeasibility Analysis of Petri Nets Composed via Shared Transitions

Another special communication medium is the one consisting of only transitions, i.e., two nets communicating with each other by rendezvous or merging transitions.

Definition 6.2 (Composition by shared transitions)

A Petri net $PN=(P,T,F,W)$ is said to be obtained by the composition of two nets $PN_1=(P_1,T_1,F_1,W_1)$ and $PN_2=(P_2,T_2,F_2,W_2)$ via shared transitions $\{T_s\}$ iff

1. $P=P_1 \cup P_2$;
2. $T=T_1 \cup T_2$;
3. $F=F_1 \cup F_2$;
4. $P_1 \cap P_2 = \emptyset$;
5. $T_1 \cap T_2 = T_s \in T$;

By this composition, the structural equations of PN are

$$AX=0$$

$$X \geq 0$$

where

$$A = \begin{bmatrix} A_{1c} & A_{1s} & 0 \\ 0 & A_{2s} & A_{2c} \end{bmatrix}$$

in which A_{1s} and A_{2s} are the subsets of A_1 and A_2 covering the shared transitions. The structural equations of PN become:

$$AX = \begin{bmatrix} A_{1c} & A_{1s} & 0 \\ 0 & A_{2s} & A_{2c} \end{bmatrix} \begin{bmatrix} X_{1c} \\ X_s \\ X_{2c} \end{bmatrix} = 0$$

Then we have

$$\begin{aligned} A_{1c}X_{1c} + A_{1s}X_s &= 0 \\ A_{2s}X_s + A_{2c}X_{2c} &= 0 \end{aligned} \quad (6-1)$$

In equation (6-1), X_s is in both sets, which implies the possibility that we may not find an $X_s > 0$ which makes both equations hold. This means that even if the two subnets are viable or feasible, the global net obtained by the composition of these two nets might be nonviable or infeasible. For example, in Figure 6.2, (a) and (b) are both viable, but net (c), which is composed of (a) and (b) by sharing transitions (shaded), is not viable because of a pure trap inside (c). The *INS* is shown in (d).

Because the composition via shared transitions does not necessarily preserve the structural properties of the subsystem, the communication medium needs verification as well. A communication medium PN_0 is obtained by including the shared transitions and the places attached to the transitions in PN_1 and PN_2 as defined in Figure 6.3.

FIGURE 6.2 A nonviable net composed by shared transitions

Shaded transitions are $\{T_S\}$

FIGURE 6.3 Shared by Transitions

Corollary 6.2

Let a Petri net $PN=(P,T,F,W)$ be obtained by the composition of two nets

$PN_1=(P_1,T_1,F_1,W_1)$ and $PN_2=(P_2,T_2,F_2,W_2)$ via shared transition $\{T_s\}$ and let

$\{P_{1s}\}=\{p \in P_1 | W(p,t)>0 \text{ or } W(t,p)>0, t \in T_s\}$ and $\{P_{2s}\}=\{p \in P_2 | W(p,t)>0 \text{ or } W(t,p)>0,$

$t \in T_s\}$ and $PN_0=(P_{1s} \cup P_{2s}, T_s, F_s, W_s)$. If PN_0 ,

$PN_1'=(P_1 \setminus P_s, T_1 \setminus T_s, F_1 \setminus F_s, W_1 \setminus W_s)$ and $PN_2'=(P_2 \setminus P_s, T_2 \setminus T_s, F_2 \setminus F_s, W_2 \setminus W_s)$ are viable or feasible, then PN is viable or feasible.

Proof: By corollary 6.1 it is easy to get the result. In Figure 6.3, the problem of composition of two subnets via shared transitions is converted to a problem of composition of three subnets by shared places P_{1s} and P_{2s} . ■

This corollary implies that when two nets are merged via transitions, the viability or infeasibility testing can be done in three smaller models first. If all the three are found to be viable or feasible, then the global system will preserve the good structural properties of the subsystems.

6.3 Viability and Infeasibility Analysis of Petri Nets Composed via Shared Places and Transitions

In this section, we discuss the most general composition mode: composition of nets by sharing both places and transitions.

Definition 6.3 (Shared places and transitions)

A Petri net $PN=(P,T,F,W)$ is said to be obtained by the composition of two nets $PN_1=(P_1,T_1,F_1,W_1)$ and $PN_2=(P_2,T_2,F_2,W_2)$ via shared places $\{P_s\}$ and transitions $\{T_s\}$ iff

1. $P=P_1 \cup P_2$;
2. $T=T_1 \cup T_2$;
3. $P_1 \cap P_2 = P_s \in P$;
4. $T_1 \cap T_2 = T_s \in T$.

Figure 6.4 is an example of composition by sharing places and transitions. Like sharing by transitions, this kind of composition may also result in nonviability or infeasibility of the global net even though the subnets are viable or feasible. In terms of viability or feasibility preservation, there is no general characterization for this kind of composition. In Figure 6.4, (a) and (b) both are viable nets, but the global net (c) composed of (a) and (b) via the shared transitions and places (shaded) is nonviable. Figure 6.5 is the *INS*.

The structural equations of PN become formula (6-2), if it is composed by shared transitions and places. For some special cases, such as *sequential processes* and *well formed blocks* [Souissi 91], A_{1sp} and A_{2sp} in (6-2) become zero. The structure of the PN is shown in Figure 6.6.

$$AX = \begin{bmatrix} A_{1c} & A_{st} & \\ A_{1sp} & A_{pt} & A_{2sp} \\ & A_{2st} & A_{2c} \end{bmatrix} \begin{bmatrix} X_{1c} \\ X_s \\ X_{2c} \end{bmatrix} = 0 \quad (6-2)$$

FIGURE 6.4 Composition of nets via shared transitions and places

FIGURE 6.5 An *INS* of Figure 6.4(c)

FIGURE 6.6 Composition via a well formed medium

Definition 6.4 (Well-formed medium)

A Petri net $PN=(P,T,F,W)$ is said to be obtained by the composition of two nets

$PN_1=(P_1,T_1,F_1,W_1)$ and $PN_2=(P_2,T_2,F_2,W_2)$ by a *well-formed medium*

$PN_0=(P_s \cup K_1 \cup K_2, T_s, F_0, W_0)$ as shown in Figure 6.6 iff

1. $P=P_1 \cup P_2$, $T=T_1 \cup T_s$;
2. $P_s=P_1 \cap P_2$, $T_s=T_1 \cap T_2$;
3. $K_1=((\bullet T_s \cup T_s \bullet) \setminus P_s) \cap P_1$, $K_2=((\bullet T_s \cup T_s \bullet) \setminus P_s) \cap P_2$
4. $F_0=T_s \times P_s \cup P_s \times T_s \cup T_s \times K_1 \cup K_1 \times T_s \cup T_s \times K_2 \cup K_2 \times T_s$.

This is to express that besides the fact that PN_1 and PN_2 can access a common part of the system (the subnet PN_0), each of them can access a particular set of buffers, K_1 and K_2 respectively. If we view such a net PN as a model of a concurrent program, where PN_1 and PN_2 model two sets of tasks of this program, then P_s can be seen as a set of global variables (since they belong to both PN_1 and PN_2) which can be accessed only via PN_0 . K_1 and K_2 can be seen as a set of local variables belonging to PN_1 , PN_2 respectively. In that case, PN_0 represents the synchronization mode between PN_1 and PN_2 . This special communication medium also preserves the structural properties of the subnets by the following corollary.

Corollary 6.3

Let a Petri net $PN=(P,T,F,W)$ be obtained by the composition of two nets $PN_1=(P_1,T_1,F_1,W_1)$ and $PN_2=(P_2,T_2,F_2,W_2)$ via a well-formed medium $PN_0=(P_s \cup K_1 \cup K_2, T_s, F_0, W_0)$. If PN_0 , $PN_1'=(P_1 \setminus P_s, T_1 \setminus T_s, F_1', W_1')$ and $PN_2=(P_2 \setminus P_s, T_2 \setminus T_s, F_2', W_2')$ are viable or feasible then PN is viable or feasible.

Proof: Follows easily from corollary 6.1. ■

This property also holds when the net is obtained by the composition of n subnets via $n-1$ well-formed media by corollary 6.1.

By this corollary, if a large system obtained by the composition of a number of subsystems via well-formed media, we do not need to analyze the viability or feasibility of the whole system as long as the medium PN_0 and the submodels are viable or feasible.

6.4 Petri Box Analysis Approach

When we localize problems in a large Petri net, sometimes we may get a very large *IIS* or *INS*, which is difficult to understand, due to the propagation of the local problem into the global system. In order to find a smaller *INS* or *IIS*, if it really exists, in these kinds of nets, an immediate approach is to decompose the net in a certain way and isolate the source within a smaller subnet. There are two kinds of large Petri net models: those composed of a number of smaller nets (or modules) and those without modules. In this section, we develop decomposition localization techniques for these two kinds of large models.

6.4.1 Decomposition

A *Petri box* consists of an internal Petri net structure and interface nodes through which the internal structure communicates with other boxes as shown in Figure 6.7, where p_1 and p_2 are *interface places*, t_1 and t_2 are *interface transitions* and the arcs to or from other boxes are called *interface arcs*. A global Petri net can be seen as the composition of a number of Petri boxes as in Figure 6.8, in which p_1, p_2, p_3 and p_4 are the interface places, and t_1, t_2, t_3 and t_4 are the interface transitions.

FIGURE 6.7 A Petri Box

For a large Petri net composed of a number of modules, each module can be modeled by a Petri box. The boundaries of these Petri boxes are defined during the design stage. To analyze the structural properties of this kind of Petri net model, we develop the following rules to decompose the global net into a so called block and link structure [Williams 90].

Petri box decomposition rules

1. Assign each shared place and each shared transition (which appear in more than one box) to one box such that no place or transition appears in more than one box.
2. Change the boundary of a box by adding all the transitions which are in the other boxes and linked to the interface places of the box; delete the interface arcs incident on the interface transitions in the box.

FIGURE 6.8 Composition of Petri Boxes

For example, the model in Figure 6.8 is decomposed into three submodels shown in Figure 6.9. Then the model can be seen as the result of merging the shared transitions. The incidence matrix of the global net has a block and link structure of Figure 6.10, where A_i is the incidence matrix of an internal structure which does not include any interface transitions and I_i is the incidence matrix of interfaces.

Definition 6.5 (Viability and Feasibility of a Petri box)

A Petri box is said to be viable or feasible if a Petri net obtained by the above *Petri box decomposition rules* is viable or feasible.

The reason for this definition is simple. At an interface place, if we delete interface arcs, then the place may become a source place (without incoming arcs) or a sink place (without leaving arcs) and the module would be nonviable. This is contradict to the real function of the module which is viable when it sends or receives flows to and from other modules.

This decomposition makes it possible to exploit the relationship between viability or feasibility of the boxes and viability or feasibility of the global net as described in the following theorem. For each block, define a linear program composed of the equations in the block together with the interface (link) variables appearing as slack or surplus variables. The block is said to be feasible if this linear program is feasible.

FIGURE 6.9 Decomposition of Petri Box Model

Theorem 6.2 [Murphy 86]

In a block and link structure system of linear equations, if one or more blocks are infeasible, then the system is infeasible. ■

By the theorem, we can conclude that if one or more Petri boxes are infeasible or nonviable, then the global net is infeasible or nonviable. The result can be used to debug a large Petri net system which is composed of a number of Petri boxes. When a large Petri net is

found to be nonviable or infeasible and the *INS* or *IIS* is too large to understand, we can localize the problem in a box (or block) first. If a block is infeasible, the objective of isolation is achieved and we can confine further analysis to the block. If all the blocks are viable or feasible, then the *INS* or *IIS* must include some interface transitions. Each interface transition is shared by a number of boxes. We can start from those shared by the least number of boxes and then move up to those shared by more boxes until we localize the problem within a subsystem which is as small as possible. This idea can be illustrated by the following diagram.

FIGURE 6.10 A Block and Link Structure Petri Net

FIGURE 6.11 Localizing an *INS* or *IIS* to a smaller subnet

6.4.2 Algorithm

The following algorithm is used to localize the structural problems of a large Petri net composed of a number of Petri boxes. Let $NIT(i)$ stand for an interface transition belong-

ing to an *INS* or *IIS* and shared by i boxes. The algorithm to test Petri box models is as follows:

Algorithm for nonviability and infeasibility analysis of Petri box models

1. Convert a Petri net model $PN=(P,T,F,W)$ into a processing network model.
2. Test the viability or feasibility of the processing network model;
3. IF the model is nonviable or infeasible THEN
 - 3.1 Localize the *INS* or *IIS*;
 - 3.2 IF the *INS* or *IIS* is too large to diagnose the problem THEN
 - 3.2.1 Identify the nonviable transitions $\{NIT(i)\}$ in the *INS* or *IIS*;
 - 3.2.2 Find the minimum i , i.e. $j=\min\{i\}$;
 - 3.2.3 Select a subsystem $B(j)$ obtained by the composition of the boxes sharing the transition $NIT(j)$;
 - 3.2.4 Test the viability or feasibility of $B(j)$;
 - 3.2.5 IF $B(j)$ is nonviable or infeasible THEN

Localize the *INS* or *IIS*;

Fix the problem and modify the net;

go to step 1;

ELSE

$\{i\}=\{i\}\setminus j$;

go to step 3.2.2;
 - ELSE

fix the problem in the global system;
- ELSE

Stop.

Another localization technique for the block and link structure LP models has been developed by Murphy[86]. It finds the minimum and maximum levels of each link variable. This can result in tightening the bounds on the link variables. Then process the second box in the same way. Finally we see on one or a few link variables, the lower bound value obtained from one block is greater than its upper bound obtained by another block or vice versa. This technique can be directly applied to the localization of nonviable interface transitions in Petri box models, but its disadvantages are that it can not present structural information and sometimes fails to work.

6.4.3 Localizing a smaller *INS* or *IIS* in a Large Petri net without Modules

If a system or a module within a system itself has a large size and a large *INS* or *IIS* is obtained when this single net is analyzed, the above algorithm can not be applied to this situation to reduce the size of *INS* or *IIS*. We can still exploit theorem 6.2 to get a smaller *INS* or *IIS* if it exists. The idea is to decompose a large Petri net into two Petri boxes. The localization operation is conducted in one box which includes the edge on which the flow is forced to zero. The breadth first search method described in the following algorithm is used to expand the boundary of this box until we find it is nonviable or infeasible.

The "victim" information (zero-flow edges) in the global *INS* or *IIS*, the one obtained when the whole net is tested, is used to guide the decomposition. We select the row constraint to which a zero-flow edge belongs as the starting point of decomposition. When the boundary of the box is expanded (the worst case is to include the whole net again), it will finally include a bad structure (*INS* or *IIS*) to indicate what is wrong with the structure.

Algorithm to localize a smaller *INS* or *IIS*

1. Find the place P_b to which a zero-flow edge is incident;
2. Find all the transitions T_b incident to the place P_b ;
3. Test the viability or feasibility of the Petri box (P_b, T_b, F_b, W_b) ,

where $F_b = P_b \times T_b \cup T_b \times P_b$;

4. IF the box is viable or feasible THEN

Find places $\{P'\} = \{p | W(t,p) > 0 \text{ or } W(p,t) > 0 \text{ for } \forall t \in T_b\}$;

Find all the transitions T' incident to P' ;

$P_h = P_h \cup P'$;

$T_h = T_h \cup T'$;

GOTO step 3;

ELSE

locate the *INS* or *IIS*;

stop.

The following example shows the advantage of this algorithm in getting a smaller *INS* for easy diagnosis.

When an edge (p2,t43) is added into the net given in Table 1 in Appendix (problem 17 in Table 2 in Appendix), we get a large *INS* with 18 places, 64 transitions and 163 arcs. In the *INS*, the edge (p2, t43) is reported as a zero-flow edge but we don't know what is wrong with the structure. Using the above algorithm, we extract a Petri box from the net as shown in Figure 6.12. The box is viable. Then we expand it according to the algorithm and get another box in Figure 6.13 in which all the interface arcs incident to p23, p24, p25, p34, p35 and p36 are collapsed to one inflow and one outflow (slack variable and surplus variable) on each of them. The net is nonviable and the *INS* is shown in Figure 6.14 which includes a pure siphon. The *INS* has only two places, 4 transitions and 9 arcs, much smaller than the global one.

FIGURE 6.12 Petri Box 1

FIGURE 6.13 Petri Box 2

FIGURE 6.14 An INS of Box 2

7.0 A Case Study

In this chapter, we select a GSPN model of a distributed discrete event simulator [Balbo 92] as a case study because it is reasonably large with 70 places, 80 transitions and 215 arcs and it also consists of six modules which can be represented by Petri boxes. The size and complexity of the model are appropriate for us to show how the debugging algorithms in section 5.2.1 and 6.4.2 work in a real case.

7.1 A Distributed Simulation System

The simulation program is composed of the receiver, the simulator, and the transmitter. The simulation is parallelized. The simulated system is decomposed into loosely coupled subsystems; each simulation module performs a standard discrete event simulation of the corresponding subsystem. Interaction with the other simulating modules takes place through external arrivals (the receiver collects from external transmitters and forwards to the simulator) and departures toward the external world that the transmitter sends to the other subsystems. We consider a system composed of two subsystems, each with three modules. The structure of the program is shown in Figure 7.1. The transmitter of module i sends messages to the receiver of module $(i+1) \bmod n$, using channel $\text{chan}T_iR_{i+1}$.

Each local simulator has its own time called local virtual time (LVT). The process can exchange four types of information: arrival (A), antmessages (AM), departure (D) and rollback (RB). Arrivals and antmessages are sent by a transmitter to the next receiver and by the receiver to the corresponding simulator. Departure messages are sent by the simulator to the corresponding transmitter and are then forwarded to the receiver after conversion into arrivals for the receiving subsystems. Copies are kept in the transmitter as antmessages. A rollback message is sent by the simulator to the transmitter when a rollback occurs. The transmitter then forwards to the next receiver all the stored antmessages with time stamp greater than the LVT value; this has the function of propagating the effect of rollback (i.e., the cancellation of previous events).

Each local simulator reaches its end when the LVT is greater than a given value TEOS common to all the modules. When a module reaches the TEOS it goes into a "Coma" state, an almost dead state from which the module can start running again only because of a rollback. The simulation ends when all the simulators reach the Coma state.

The code of the simulator, transmitter and receiver are listed in Figures 1 and 2 in *Appendix B*.

FIGURE 7.1 An example of distributed simulation

7.2 Petri Net Model of the System

Figure 7.2, Figure 7.3 and Figure 7.4 are the Petri net models of the programs. Each of them is a Petri box. The global system consists of 6 boxes and altogether it has 88 transitions, 70 places and 215 arcs. The empty boxes are timed transitions [Chiola 87] which can be treated as normal (immediate) transitions where structural analysis is concerned.

FIGURE 7.2 A Petri box of receiver module [Balbo 92]

FIGURE 7.3 A Petri box of simulation module [Balbo 92]

FIGURE 7.4 A Petri box of transmitter module [Balbo 92]

7.3 Model Debugging Experiments

In this section, model debugging experiments are carried out by introducing a structural problem into the model and seeing if we can localize it by the algorithms developed in chapter 5 and chapter 6 and whether the localization leads to a meaningful diagnosis of the problem.

Example 7.1

The power of our debugging tool is that it can efficiently and effectively localize nonviability and infeasibility in a large Petri net model. For example, the author made a mistake by missing the arcs from transitions "endtxR2Arr" and "endtxR2AM" in the transmitter box to the place "REC2if" in the receiver box. When the global model was tested, we had an *INS* as shown in Figure 7.5. The *INS* shows that there is a siphon {REC2if} in the model. If this siphon is not marked in the initial marking or once it has lost its tokens, then the transitions "cond1", "cond2", and "cond3" can never fire again. To fix the model, the place needs some inflow arcs, i.e., the receiver module needs messages from the transmitter by the channel chanS_iT_i as described in the code.

FIGURE 7.5 A *INS* of the model

Example 7.2

The author made another mistake by adding one arc from the transition "timeout" to the place "rxRi" in a simulator box. When the global model has gone through the viability test, we get an *INS* with 52 places, 83 transitions and 182 arcs which is difficult to understand. The *INS* shows that there is a nonviable transition "timeout" in the simulator box. Now we want to know why the transition is nonviable. The simulator box, where the transition is, is chosen and tested. We get the much smaller *INS* of Figure 7.6. The pure trap {rxRi} causes nonviability and the place is unbounded. If the arc is translated into code, it causes an infinite loop in the first "case" section of the simulator code, i.e., the program may always check the current time. After removing the arc, the global net is free of nonviability. From this example, we can see that if the module test were done first, this kind of situation would not happen.

FIGURE 7.6 An *INS* of the model **Example 7.3**

In the simulation module, the statement “if (rollback occurred) chanSiTi ! RBmsg” translating (see Figure 2 in *Appendix B*) requires that transmitter should receive a “rollback” message from the simulator. Suppose the modeler made a mistake in interpreting this statement and did not connect the transition “endtxTiRB” to the place “IfRollBack” in the transmitter module. After testing the global model, we get an *INS* with 34 places, 54 transitions and 117 arcs, too large to diagnose. The *INS* shows that the transition “endtxTiRB” is nonviable in the simulator box. By the Petri box analysis algorithm, the simulator box is tested and it is viable. The transition “endtxTiRB” is shared by both simulator and transmitter boxes. By the Petri box model debugging algorithm, we test the subsystem composed of the two boxes and find that it is nonviable. The *INS* obtained is shown in Figure 7.7. The nonviability is due to a siphon consisting of all the places in the figure. From the flow point of view, there is an imbalance between the box transmitter and box simulator. This means that if the simulation is in “Coma” state, it can not restart running by receiving a roll back message.

8.0 Conclusions

8.1 Contributions

The Petri net model debugging techniques developed here are quite helpful for isolating certain structural problems in models which are too large for manual diagnosis. The major contributions of this thesis are as follows.

- It links processing networks with Petri nets according to their topological similarities and paves the way for the application of processing network model debugging techniques to Petri net model debugging.
- The research defines viability and infeasibility of Petri net models and uses the definitions to interpret structural problems of Petri net models including inconsistency, non-conservativeness, structural unboundedness and nonrepetitiveness. This relationship makes it possible to do viability and infeasibility analysis in Petri net models.
- The thesis introduces viability and infeasibility analysis into Petri net model debugging and shows that these techniques are better than those being used now in three aspects: isolating a smaller problem set; giving information about bad structures embedded in the model; localizing one problem at a time. All three advantages are helpful in diagnosis.
- Four debugging algorithms are designed in the thesis to localize sources of inconsistency, nonconservativeness, structural unboundedness and nonrepetitiveness. The algorithms have been implemented and their effectiveness has been shown by examples.
- Siphons and traps are discussed in the context of nonviability and infeasibility and we show that certain subclasses of siphons and traps, which cause nonviability and/or infeasibility can be localized in polynomial time. If we want a good Petri net model, these kinds of siphons and traps should be removed.
- Viability and infeasibility analysis are extended to analyze large Petri nets composed of a number of subnets. We point out methods to verify the preservation of structural properties from subnets to the global net obtained by some special communication media such as shared places, transitions, and a well-formed medium.

- A Petri box viability and infeasibility analysis approach is presented to debug systems designed by a modular or compositional approach. A special algorithm is designed to localize an *INS* or *IIS* which is as small as possible in Petri box models.
- For a large Petri net without modules, an algorithm is designed to reduce the size of an global *INS* or *IIS*, which is too large to understand, by a Petri box decomposition method.

8.2 Future Research

We expect that there is a nice characterization between viability and structural liveness in some special classes of Petri nets such as marked graphs and free choice nets. If so the viability and infeasibility analysis techniques can be used to localize structural nonliveness problem in these nets. This could be an interesting research area in the future.

An interesting and application oriented research area is to apply our Petri box analysis method to object oriented testing. In CO-OPN model, each object is modeled by a Petri box and each box communicates with other boxes via external transitions (or methods) [Buchs 91]. Our Petri box analysis method can find and localize all the structural problems caused by the interaction of the objects and lead to understanding of the structural relationship between objects.

This thesis applies model debugging techniques developed in the linear programming area to Petri net model debugging. On the other hand, introducing Petri net structural properties into general network models or general LP models is another interesting topic. In the LP model debugging area, infeasibility and nonviability need to be explained by structural properties instead of simply by mathematical definitions. We can expect that structural unboundedness, siphons and traps, inconsistency and nonconservativeness have their counterparts in general LP models and they may not only explain some LP model errors but can also help visualize the model errors. This visualization may shed more light on the causes of the model errors and help model repair.

Appendix A

In this appendix, we list the structure of the Petri net which are used in section 5.4.5 for localization testing and the testing results.

TABLE 1 The structure of a Petri net

p35,p1 [t1] p36,p2 ;	p36,p22 [t24] p21,p35 ;	p35,p19 [t47] p36,p20;
p33,p1 [t2] p34,p2 ;	p35,p25 [t25] p36,p26 ;	p33,p19 [t48] p34,p20;
p34,p2 [t3] p1,p33 ;	p33,p25 [t26] p34,p26 ;	p36,p16 [t49] p15,p35 ;
p36,p2 [t4] p1,p35 ;	p34,p26 [t27] p25,p33 ;	p34,p16 [t50] p15,p33 ;
p35,p5 [t5] p36,p6 ;	p36,p26 [t28] p25,p35 ;	p35,p15 [t51] p36,p16 ;
p33,p5 [t6] p34,p6 ;	p35,p29 [t29] p36,p30 ;	p33,p15 [t52] p34,p16 ;
p34,p6 [t7] p5,p33 ;	p33,p29 [t30] p34,p30 ;	p36,p12 [t53] p11,p35 ;
p36,p6 [t8] p5,p35 ;	p34,p30 [t31] p29,p33 ;	p34,p12 [t54] p11,p33 ;
p35,p9 [t9] p36,p10 ;	p36,p30 [t32] p29,p35 ;	p35,p11 [t55] p36,p12 ;
p33,p9 [t10] p34,p10 ;	p35,p31 [t33] p36,p32 ;	p33,p11 [t56] p34,p12 ;
p34,p10 [t11] p9,p33 ;	p33,p31 [t34] p34,p32 ;	p36,p8 [t57] p7,p35 ;
p36,p10 [t12] p9,p35 ;	p34,p32 [t35] p31,p33 ;	p34,p8 [t58] p7,p33 ;
p35,p13 [t13] p36,p14 ;	p36,p32 [t36] p31,p35 ;	p35,p7 [t59] p36,p8 ;
p33,p13 [t14] p34,p14 ;	p36,p28 [t37] p27,p35 ;	p33,p7 [t60] p34,p8 ;
p34,p14 [t15] p13,p33 ;	p34,p28 [t38] p27,p33 ;	p36,p4 [t61] p3,p35 ;
p36,p14 [t16] p13,p35 ;	p35,p27 [t39] p36,p28 ;	p34,p4 [t62] p3,p33 ;
p35,p17 [t17] p36,p18 ;	p33,p27 [t40] p34,p28 ;	p35,p3 [t63] p36,p4 ;
p33,p17 [t18] p34,p18 ;	p36,p24 [t41] p23,p35 ;	p33,p3 [t64] p34,p4 ;
p34,p18 [t19] p17,p33 ;	p34,p24 [t42] p23,p33 ;	p35 [t65] p33 ;
p36,p18 [t20] p17,p35 ;	p35,p23 [t43] p36,p24 ;	p33 [t66] p35;
p35,p21 [t21] p36,p22 ;	p33,p23 [t44] p34,p24 ;	
p33,p21 [t22] p34,p22 ;	p36,p20 [t45] p19,p35 ;	
p34,p22 [t23] p21,p33 ;	p34,p20 [t46] p19,p33 ;	

In Table 1, the places before the transitions are the pre-places and the places after the transitions are the post-places. The weight of all the arcs is 1.

TABLE 2 Experimental structural problems

Problems	Size of the <i>INSs</i>
1. delete edge (t3, p1) and (t4, p1)	one place and two arcs
2. add edge (t3, p20), (t3, 21)	2 places, 3 transitions and 8 arcs
3. delete edge (t7, p5)	2 places, 3 transitions and 8 arcs
4. delete edge (t15, p13)	2 places, 3 transitions and 8 arcs
5. delete edge (t19, p17)	2 places, 3 transitions and 8 arcs
6. delete edge (t27, p25)	2 places, 3 transitions and 8 arcs
7. delete edge (t31, p29)	2 places, 3 transitions and 8 arcs
8. delete edge (t34, p34)	4 places, 65 transitions and 162 arcs
9. delete edge (t37, p27)	2 places, 3 transitions and 8 arcs
10. delete edge (t41, p23)	2 places, 3 transitions and 8 arcs
11. delete edge (t46, p19)	2 places, 3 transitions and 8 arcs
12. delete edge (t62, p3)	2 places, 3 transitions and 8 arcs
13. add edge (t20, p3)	2 places, 4 transitions and 11 arcs
14. add edge (t24, p23)	2 places, 4 transitions and 11 arcs
15. add edge (t28, p30)	2 places, 4 transitions and 11 arcs
16. add edge (t38, p4)	2 places, 4 transitions and 11 arcs
17. add edge (p2, t43)	18 places, 64 transitions and 161 arcs
18. add edge (p1, t50)	18 places, 64 transitions and 163 arcs
19. add edge (p13, t65)	2 places, 4 transitions and 11 arcs
20. add edge (p10, t41)	2 places, 4 transitions and 11 arcs
21. add edge (p33, t53)	4 places, 66 transitions and 161 arcs
22. add edge (p20, t33)	2 places, 4 transitions and 11 arcs
23. add edge (p22, t14)	2 places, 4 transitions and 11 arcs
24. delete edge (p33, t56)	4 places, 66 transitions and 164 arcs
25. delete edge (t20, p3)	4 places, 65 transitions and 162 arcs
26. add edge (p11, t28)	2 places, 4 transitions and 11 arcs
27. add edge (p6, t25)	2 places, 4 transitions and 11 arcs
28. add edge (p9, t23)	2 places, 4 transitions and 11 arcs
29. add edge (p21, t48)	2 places, 4 transitions and 11 arcs
30. delete edge (p8, t58)	2 places, 3 transitions and 8 arcs

The left column lists thirty problems, which were introduced one at a time into the net in Table 1.
The right column shows the isolation results.

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

TABLE 3 Localization result

Sources	Size of the <i>INSs</i>
Problem 9	2 places, 3 transitions and 8 arcs
Problem 6	2 places, 3 transitions and 8 arcs
Problem 5	2 places, 3 transitions and 9 arcs
Problem 4	2 places, 4 transitions and 10 arcs
Problem 19	2 places, 4 transitions and 11 arcs
Problem 26	2 places, 4 transitions and 12 arcs
Problem 28	2 places, 4 transitions and 11 arcs
Problem 20	2 places, 3 transitions and 8 arcs
Problem 30	2 places, 3 transitions and 9 arcs
Problem 3	2 places, 4 transitions and 10 arcs
Problem 27	2 places, 4 transitions and 10 arcs
Problem 1	1 place, 3 arcs
Problem 2 (t3,p20)	2 places, 4 transitions and 10 arcs
Problem 11	2 places, 3 transitions and 9 arcs
Problem 22	2 places, 4 transitions and 10 arcs
Problem 2 (t3,p21)	2 places, 4 transitions and 12 arcs
Problem 29	2 places, 4 transitions and 11 arcs
Problem 23	2 places, 4 transitions and 11 arcs
Problem 18	18 places, 64 transitions and 167 arcs
Problem 8	18 places, 63 transitions and 160 arcs
Problem 16	18 places, 64 transitions and 161 arcs
Problem 15	18 places, 64 transitions and 160 arcs
Problem 17	18 places, 64 transitions and 160 arcs

These are the localization results when introducing all the problems in Table 2 at the same time. They are ordered according to the sequence that they were found.

Appendix B

In this appendix, the code of receiver, transmitter and simulator modules used in Chapter 7 are given. These three programs are transformed into three Petri net models shown in Chapter 7.

FIGURE B.1 Receiver and transmitter code [Balbo 92]

<pre> Receiver module Rec_i seq COMA = false EndOfSim = false while true seq chanT_jR_i ? msg if not COMA if msg.time ≥ TEOS seq COMA = true chanR_iS_j ! msg endseq endif else { Filter successive messages with time stamp greater than TEOS } if msg.time < TEOS seq COMA = false chanR_iS_j ! msg endseq endif endseq endwhile endseq </pre>	<pre> Transmitter module Tran_i while true chanS_iT_i ? msg if msg.type=rollback while (there are anti-messages) chanS_iT_k ! anti-msg endwhile else chanT_iR_k ! msg endif end while </pre>
---	--

FIGURE B.2 Simulator code [Balbo 92]

```

Simulation module Simi
seq
LVT = 0
while true
  while (LVT < TEOS)
    seq
    if (EventList is not Empty)
      {get event from event list}
      LVT = Event.time
      case Event.type
      A: {update the event list and server queue}
      D: {update the event list and server queue}
      msg = Event
      msg.type = A
      chanSiTi ! Event
      clock ? timenow
      end case
      alt
        chanRiSi ? msg
        seq
        case msg.type
        AM: {delete message from queue}
        A: {insert arrival in event list}
        end case
        if LVT > msg.time
          {perform rollback}
        endseq
        clock ? after timenow plus timeout
        if (rollback occurred)
          chanSiTi ? RBmsg
        endseq
      endalt
    end while
  endwhile
endwhile

{Coma management}
while LVT ≥ TEOS
  seq
  chanRiSi ? msg
  case msg.type
  AM: {delete message from
        received message queue}
  A: {Insert arrival in event list}
  end case
  if LVT > msg.time
    seq
    {perform rollback}
    chanSiTi ! RBmsg
  endseq
  endif
endseq
end while
endwhile
endseq

```

References

- [Balbo 92]** Balbo, G., Donatelli, S., and Franceschinis, G., "Understanding Parallel Program Behavior through Petri Net Models," *Journal of Parallel and Distributed Computing*, Vol.15, 171-187 (1992)
- [Best 87]** Best, E., "Structure theory of Petri nets: the free choice hiatus," in Rozenberg, G. (ed) *Advances in Petri Nets 1986*, LNCS 254, Springer-Verlag, 168-205 (1987)
- [Best 91]** Best, E., "The Petri Box Calculus for Concurrent Programs," in *Proceedings of 12th International Conference on Application and Theory of Petri Net*, Aarhus, Denmark, 33-42, 1991
- [Buchs 91]** Buchs, D., and Guelfi, N., "CO-OPN: A Concurrent Object Oriented Petri Net Approach for System Specification," in *Proceedings of 12th International Conference on Application and Theory of Petri Net*, Aarhus, Denmark, (1991)
- [Chen 86]** Chen, C-H. J., and Engquist, M., "A Primal Simplex Approach to Pure Processing Networks," *Management Science*, 32, 1582-1598 (1986)
- [Chvatal 83]** Chvatal, V., *Linear Programming*, W.H. Freeman and Company, 1983
- [Chinneck 90a]** Chinneck, J. W., "Formulating Processing Network Models: Viability Theory," *Naval Research Logistics*, Vol. 37, 245-261 (1990)
- [Chinneck 90b]** Chinneck, J. W., "Processing Network Models of Energy/Environment Systems," *Technical Report*, Systems and Computer Engineering, Carleton University, (1990)
- [Chinneck 91]** Chinneck, J. W., and Dravnieks, E. W., "Locating Minimal Infeasible Constraint Sets in Linear Programs," *ORSA Journal on Computing*, Vol. 3, No.2, 157-168 (1991)
- [Chinneck 92]** Chinneck, J. W., "Viability Analysis: A Formulation Aid for All Classes of Network Models," *Naval Research Logistics*, Vol.39, 531-543 (1992)

[Chinneck 93] Chinneck, J. W., "Finding the Most Useful Subset of Constraints for Analysis in an Infeasible Linear Program," *Technical Report SCE-93-07*, Department of Systems and Computer Engineering, Carleton University, 1993

[[Chiola 87] Chiola, G., "GreatSPN Users' Manual", 1987

[Colom 91] Colom, J.M. and Silva, M., "Convex Geometry and Semiflows in P/T nets. A Comparative Study of Algorithms for Computation of Minimal P-semiflows," in Rozenberg, G. (ed) *Advances in Petri Nets 1990*, LNCS 483, Springer-Verlag, 79-112 (1991)

[Esparza 91a] Esparza, J. and Silva, M., "On the Analysis and Synthesis of Free Choice Systems," in Rozenberg, G. (ed) *Advances in Petri Nets 1990*, LNCS 483, Springer-Verlag, 243-286 (1991)

[Esparza 91b] Esparza, J. and Silva, M., "Compositional Synthesis of Live and Bounded Free Choice Petri Nets," in Baeten, J.C.M. and Groote, J.F. (eds) *CONCUR' 91*, LNCS 527, Springer-Verlag, 172-187, (1991)

[Esparza 92] Esparza, J., "A Polynomial-time Algorithm to Decide Liveness of Bounded Free Choice Nets," *Theoretical Computer Science* 102, 185-205, (1992)

[Ezpeleta 91] Ezpeleta, J. and Couvreur, J.M., "A New Technique for Finding a Generating Family of Siphons, Traps and st-components. Application to Colored Petri Nets," in *Proceedings of 12th International Conference on Application and Theory of Petri Nets*, Aarhus, Denmark 1991, 145-164

[Genrich 81] Genrich, H. J., and Lautenbach, K., "System Modelling with High-Level Petri Nets," *Theoretical Computer Science* 13 (1981) 109-136

[Genrich 87] Genrich, H. J., "Predicate/Transition Nets," in Rozenberg, G. (ed) *Advances in Petri Nets 1986*, LNCS 254, Springer-Verlag, 207-247 (1987)

[Greenberg 83] Greenberg, H. J., "A Computer-Assisted Analysis System for Linear Programming Models," *ACM Transactions on Mathematical Software*, Vol. 1, No.9, 18-56 (1983)

[Greenberg 91] Greenberg, H. J. and Murphy, F. H., "Approaches to Diagnosing Infeasible Linear Programs," *ORSA Journal on Computing*, Vol. 3, No.3, 253-261 (1991)

[Jensen 87] Jensen, K., "Colored Petri Nets," in Rozenberg, G. (ed) *Advances in Petri Nets 1986*, LNCS 254, Springer-Verlag, 248-299 (1987)

[Koene 84] Koene, J., "Minimal Cost Flow in Processing Networks, a Primal Approach," *CWI Tract*, 4, (1984)

[Kruckeberg 88] Kruckeberg, F. and Jaxy, M., "Mathematical Methods for Calculating Invariants in Petri Nets," in Rozenberg, G. (ed) *Advances in Petri Nets 1987*, LNCS 266 Springer-Verlag, 104-131, (1988)

[Lautenbach 74] Lautenbach, L., "Use of Petri Nets for Proving Correctness of Concurrent Processing Systems," *Information Processing* 74, 187-191 (1974)

[Lautenbach 87a] Lautenbach, L. "Linear Algebraic Techniques for Place/Transition Nets," in Rozenberg, G. (ed) *Advances in Petri Nets 1986*, LNCS 254, Springer-Verlag, 142-167 (1987)

[Lautenbach 87b] Lautenbach, L. "Linear Algebraic Calculation of Deadlocks and Traps," in Voss, K., Genrich, H. J., and Rozenberg, G. (eds) *Concurrency and Nets*, Springer-Verlag, 315-335, (1987)

[Martinez 82] Martinez, J., and Silva, M., "A Simple and Fast Algorithm to Obtain All Invariants of a Generalized Petri Net," in Girault, C., and Reisig, W. (eds), *Application and Theory of Petri Nets*, Informatik-Fachberichte, Springer-Verlag, 301-310 (1982)

[Mayr 84] Mayr, E. W., "An algorithm for the general Petri net reachability problem," *SIAM Journal of Computing*, Vol.13, no.3, 441-460, (1984)

[Meta 92] Meta Software Corporation, "Design/CPN Manual", 1992

[Murata 89a] Murata, T. "Petri Nets: Properties, Analysis and Applications", *Proceeding of IEEE*, Vol. 77 No. 4, 541-580, (1989)

- [Murata 89b]** Murata, T., Shenker, B. and Shatz, S. M., "Detection of Ada Static Deadlocks Using Petri Net Invariants", *IEEE Transactions on Software Engineering*, Vol. 15, No.3, 314-326, (1989)
- [Murphy 86]** Murphy, F. H. and Stohr, E. A., "An Intelligent System for Formulating Linear Programs" *Decision Support Systems* 2, 39-47 (1986)
- [Murtagh 87]** Murtagh, B. A. and Saunders, M. A., "MINOS 5.1 User's Guide," *Technical Report SOL 83-2OR*, Systems Optimization Laboratory, Department of Operations Research, Stanford University, 1987
- [Murty 83]** Murty, K.G., *Linear Programming*, Wiley, New York, (1983)
- [Peterson 81]** Peterson, J. L., *Petri Net Theory and the modelling of Systems*, Prentice-Hall, Inc. (1981)
- [Porter 66]** Porter, W. A., *Modern Foundations of Systems Engineering*, The Macmillan Company, New York, (1966)
- [Ramchandani 73]** Ramchandani, C., "Analysis of Asynchronous Concurrent Systems by Time τ Petri Nets", Ph.D. dissertation, MIT, (1973)
- [Reisig 85]** Reisig, W., *Petri Nets*, EATCS Monographs on Theoretical Computer Science, Vol. 4, Springer-Verlag, (1985)
- [Reisig 87]** Reisig, W., "Place/Transition Systems", in Rozenberg, G. (ed) *Advances in Petri Nets 1986*, LNCS 254, Springer-Verlag, 117-141 (1987)
- [Sajkowski 86]** Sajkowski, M., "Protocol Verification in the Presence of Time", in Sarikaya, B. and Bochamann, G. V., (eds) *Protocol Specification, Testing, and Verification, VI, IFIP*, North-Holland, 269-280, (1986)
- [Souissi 91]** Souissi, Y. and Memmi, G., "Composition of Nets via a Communication Medium", in Rozenberg, G. (ed) *Advances in Petri Nets 1990*, LNCS 483, Springer-Verlag, 457-470, (1991)

[Souissi 92] Souissi, Y., "On Liveness Preservation by Composition of Nets via a Set of Places", in Rozenberg, G. (ed) *Advances in Petri Nets 1991*, LNCS 524, Springer-Verlag, 277-295, (1992)

[Thiagarajan 87] Thiagarajan, P.S., "Elementary Net Systems", in Rozenberg, G. (ed) *Advances in Petri Nets 1986*, LNCS 254, Springer-Verlag, 26-59 (1987)

[Treves 90] Treves, N., "A Comparative Study of Different Techniques for Semi-flows Computation in Place/Transition Nets", in Rozenberg, G. (ed) *Advances in Petri Nets 1989*, LNCS 424, Springer-Verlag, 433-452 (1990)

[van Loon 81] van Loon, J., "Irreducibly Inconsistent Systems of Linear Inequalities", *European Journal of Operational Research*, 8, 283-288, (1981)

[Williams 90] Williams, H. P., *Model Building in Mathematical Programming*, 3rd Edition, John Wiley & Sons (1990)

END

'20-07-94

FIN